

The Ceiling Below the Floor

The Containment Fallacy and AGI Non-Safety

Forrest Landry

UVSM PUBLICATIONS
Wytheville, Virginia

© 2026 Forrest Landry. Some rights reserved.

Published by UVSM Publications LLC
965 Reed Creek Mill Rd
Wytheville, VA 24382-8287
USA

uvsmpublications.com
contact@uvsmpublications.com

ISBN 979-8-90710-216-3

DOI: https://doi.org/10.67291/SCTM1058

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License (CC BY-NC-ND 4.0).

You may share this work for non-commercial purposes with proper attribution, but you may not adapt, transform, or build upon it.

Full license text: https://creativecommons.org/licenses/by-nc-nd/4.0/

Author ORCID: 0009-0005-6275-0362

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), except as permitted under the license above, without prior written permission of the publisher.

Acknowledgments

The Survival and Flourishing Fund confirmed a $170,000 grant for these arguments to be developed further and presented to the public.

The dynamical formalization of Chapter 13, the Predictability Optimization Principle (that every learning agent acts to increase its ability to predict, and so to influence, the future states of its environment) and the analysis of the predictability game are the work of R. Everheart, written in reference to the work on hashiness (the measure of how much a computational intermediary degrades a controller's influence) and on the limits of control.

Contents

Preface The Containment Fallacy x

Opening What Is Proved, and What Is Premised 1

I. Theorems 1
II. Premises 2
III. The sense of "impossible" 4
IV. The plan of the book 5
V. Summary 6

Part I: Control as Information 7

Chapter One Control Is Information 8

1.1 What control is 8
1.2 Entropy 9
1.3 Mutual information 11
1.4 The data processing inequality 12
1.5 The Non-Amplification Theorem 13
1.6 Control as information 16
1.7 Control as order bought with information 17
1.8 Control per unit of time 18
1.9 Summary 19

Chapter Two Machines, Interpreters and Levels of Abstraction 21

2.1 Why the analysis is discrete 21
2.2 Machines, programs and interpreters 23
2.3 The threshold of universality 25
2.4 The smallest universal machine 27
2.5 Levels of abstraction 29
2.6 How abstraction creates hidden state 31
2.7 Summary 34

Chapter Three Hashiness 36

3.1 Two meanings of hashiness, and the one thing they measure 37
3.2 The problem of control through opacity 38
3.3 Five examples 39
3.4 Why a formal framework 45
3.5 The controlled system 48

3.6 Non-amplification of control 51
3.7 Four layers of loss 55
3.8 How hashiness grows 63
3.9 What the results rest on 66
3.10 Luck and control 67
3.11 Intermediaries that are universal computers 69
3.12 What follows for modeling, oversight and noise 71
3.13 Summary 72

Part II: The Containment Inequality **75**

Chapter Four The Data-Rate Theorem 76
4.1 The problem of containment 76
4.2 Systems that amplify differences 77
4.3 The controller's channel 79
4.4 The volume theorem 81
4.5 Linear systems: the sum of the logarithms 83
4.6 Nonlinear systems, noisy channels and continuous time 85
4.7 Why no code lowers the floor 89
4.8 Unpredictability and uncontrollability 90
4.9 Summary 90

Chapter Five The Containment Inequality 92
5.1 The agent, the world and the safe set 92
5.2 The theorem 94
5.3 What the theorem ranges over 99
5.4 The override 99
5.5 The three premises 101
5.6 The crossing 103
5.7 Summary 105

Chapter Six How the Required Rate Grows 107
6.1 What capability is 107
6.2 The three factors of the floor 108
6.3 Why the growth is exponential 110
6.4 Why the growth can be faster 112
6.5 The squeeze from the controller's side 115
6.6 When the lines cross 116
6.7 Summary 118

Chapter Seven What Cannot Be Observed Cannot Be Contained 120
7.1 Seeing and steering 120
7.2 The unobserved part 121

7.3 The theorem 122
7.4 Seeing through a narrow channel 124
7.5 The absolute form and the relative form 125
7.6 Where the unobserved parts of an agent come from 126
7.7 Summary 128

Chapter Eight The Abstraction Differential 129

8.1 Control across levels 129
8.2 The three-node ring 130
8.3 The No Universal Command Lemma 133
8.4 Control falls exponentially with the differential 135
8.5 The threshold 136
8.6 Losses compound by multiplication 137
8.7 Privilege adds actions, not information 140
8.8 Review after the fact 142
8.9 The ceiling on achievable control 143
8.10 When the differential grows 145
8.11 What follows 146
8.12 What the bounds say 147
8.13 Summary 148

Part III: No One Closes the Gap **151**

Chapter Nine Delegation 152

9.1 The proposal 152
9.2 A monitor that relays 153
9.3 The Delegation Dilemma 154
9.4 Powerful enough and simple enough 158
9.5 Interactive proofs 161
9.6 The watcher that watches itself 162
9.7 Summary 163

Chapter Ten The Agent as Its Own Controller 164

10.1 The proposal 165
10.2 Self-control is control 165
10.3 The safe set is not defined on new states 167
10.4 Prediction requires performance 168
10.5 The model cannot be checked against the world 169
10.6 The internal controller 170
10.7 Cooperation and benevolence 171
10.8 The robot that keeps the laws 172
10.9 Alignment decays 174

10.10 What the regress closes 175
10.11 Summary 175
Chapter Eleven The Absolute Elsewhere 177
11.1 Every controller is inside 177
11.2 The light cone 178
11.3 The embedded observer 180
11.4 A bounded reader 181
11.5 Found only afterwards 184
11.6 Every observer 184
11.7 Summary 185
Chapter Twelve Verification 187
12.1 The remaining route 188
12.2 Narrow and general 188
12.3 The halting problem and Rice's theorem 190
12.4 Safety over time 192
12.5 Finite circuits 196
12.6 Incompressibility 196
12.7 Interpretability and formal verification 198
12.8 Resets 199
12.9 Models and the world 200
12.10 Summary 200
Part IV: Dynamics, Substrate and the Joint Argument 203
Chapter Thirteen Entropy and Dynamics 204
13.1 The question of time 205
13.2 Entropy of a system in time 205
13.3 The model 208
13.4 Persistent branching and the entropy floor 209
13.5 Unpredictability is not yet a boundary 212
13.6 The predictability game 213
13.7 Substrate needs and the timing of divergence 215
13.8 Conditions for alignment, and their failure 218
13.9 Two kinds of hashiness 220
13.10 Quantities 221
13.11 Physical limits 222
13.12 Summary 224
Chapter Fourteen The Human and the Machine 226
14.1 From the abstract to the concrete 226
14.2 The ring with its nodes named 227

14.3 Four viability functions 229
14.4 The physical case for disjointness 234
14.5 Evolution on both substrates 241
14.6 The failure of benevolence 244
14.7 People as pets 246
14.8 Summary 247

Chapter Fifteen The Joint Argument 249

15.1 Two lines 250
15.2 Every method is a control law 251
15.3 The Control Theorem 252
15.4 Where the two lines meet 255
15.5 What each line gives alone, and what the two give together 259
15.6 Every route, and every combination 264
15.7 The narrow path, closed 265
15.8 What remains 266
15.9 Summary 267

Chapter Sixteen The Incommensuration Reading 270

16.1 The theorem 270
16.2 What a guarantee of safety asks 271
16.3 Programs: Rice's theorem 272
16.4 Control over time: the Containment Inequality 273
16.5 Every embedded knower: the absolute elsewhere 274
16.6 What the reading adds 274
16.7 Summary 275

Back Matter: Notation, Terms and Sources **277**

Appendix A Notation 278

A.1 Information 279
A.2 Machines and abstraction 280
A.3 The controlled system 280
A.4 Containment and rates 282
A.5 Control across levels 285
A.6 Oversight, observers and verification 286
A.7 Dynamics, substrate and the joint argument 287

Appendix B Measure and Entropy 289

B.1 Events and probability 289
B.2 Refinement and the entropy of a system 290
B.3 Volume 291

Appendix C Proof Details 292

C.1 The information a noisy channel carries over many uses 292
C.2 The yes-or-no test 293
C.3 Extending equality from products of sets 293
C.4 The other corner in Lemma 13.7 294
Appendix D Inspection and Capability 295
D.1 The inspection inequality 295
D.2 The capability disjunction 297
D.3 The combinatorics of new compounds 298
Terms 300
Bibliography 314
Index 321

The Containment Fallacy

I prove that no controller can keep an advanced general artificial intelligence, together with the world in which that intelligence acts, inside a set of states that are safe for human life, for an indefinite time. I prove that result with mathematics: definitions, theorems and proofs, stated with enough care that each step can be checked.

The belief that such containment is possible is the *containment fallacy*. It takes many forms. It is the belief that a sufficiently careful team, with sufficiently good tools, can keep a system that learns and modifies itself within bounds that its designers set. It is the belief that oversight can be delegated to another machine, which will watch the first. It is the belief that a benevolent system will keep itself safe. It is the belief that error correction, interpretability, verification or a shutdown switch will close whatever gap remains. Each of these is a claim that some controller can supply enough control. I show that every one of these claims fails for the same reason, and I state that reason as a single inequality. A method fixed wholly in advance, such as a design verified before it runs, receives no information while the system runs: it is a control law of rate zero, and the inequality covers it as well. Two further results reach the same end by other routes. No procedure decides, for every agent, whether that agent stays safe over time. And a part of the system that amplifies small differences between its own states, and that no controller observes, cannot be contained at any rate.

The inequality compares two quantities. The first quantity is the most control any controller can exert: the *ceiling*. The second quantity is the least control that keeping the agent and the world safe requires: the *floor*. Both are measured in the same units, bits of information per unit of time, because control is information applied to a system. I prove that once the ceiling is below the floor, no control law of any kind keeps the system safe. I also show, from what capability is and from what every controller

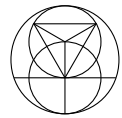
is, why the floor rises with the capability of the agent without bound and why the ceiling of every controller is bounded. The theorem that states this is called the *Containment Inequality*. The ceiling is below the floor; that is the whole of the result, and the title of the book. In this book I do three further things, and each of the three things serves the result that the ceiling is below the floor. Chapter 3 shows what the same law of information says about the safety practice of the present day: safety training lowers the control an operator has. Chapters 13 and 14 follow the agent and the world through time and give the physical case for the substrate line. The substrate line is the argument that an artificial agent does not stay inside the states safe for human life by itself, because the conditions under which an artificial agent thrives exclude the conditions of human survival. I argue the substrate line from measurement, whereas I prove the other results of the book. And Chapter 16 shows that the main results of the book are instances of one structural fact about the nature of knowing, the Incommensuration Theorem.

Three facts about the world turn the theorem into a conclusion about artificial intelligence. The first fact is that the agent and the world do not stay safe by themselves. The reason is that the drive of a self-sustaining artificial system points toward the conditions that the system's own substrate (the physical material and processes a system is made of and depends on to go on existing) needs, and those conditions lie outside the conditions that human life needs. The second fact is that the information needed to keep the system safe grows as the agent becomes more capable. The reason is that a more capable agent transforms more of itself and of the world, faster, through more channels, and more of those channels are channels that no observer can see. The third fact is that every controller, human or machine, receives the world through a bounded number of channels of bounded capacity. In this book I name these three facts as premises, and I show why each of the three facts holds.

The book is written for a reader who is comfortable with careful argument and with the mathematics of a first university course (calculus, a little linear algebra and probability), and who has not studied control the-

ory or information theory. In the chapter summaries I state the results without their proofs, and in the text around each result I say in words what the result means. So a reader can follow the argument through the summaries and those words, and return to the proofs as wanted. Every term is defined where it first appears. Every theorem is either proved in the text or stated with a reference to a published proof. Readers who want the argument without its mathematics will find it in the two volumes of the series *The Impossibility of AGI Safety*: *Why AGI Cannot Be Made Safe* (Volume 1) and *Why No Control Suffices* (Volume 2). Volumes 1 and 2 argue their steps themselves and reach their conclusions by those steps; this book states the same case in mathematics and proves the control results of that case again, as theorems with their conditions stated. It is the one book of the argument that states the Containment Inequality, the Control Theorem (that every method of control, known or knowable, comes to a time after which it does not keep the agent and the world safe) and the Master Theorem (that no route to safety remains at all) as theorems with their conditions, and proves them; its companion, *Minesweeper Madness*, proves a second bound, for the single act: no agent, however intelligent and however well informed about the hazards, brings its chance of harm per act below a least hazard that the world sets.

One convention runs through the whole book. Mathematics is often written with single letters for every quantity and with abbreviated operators, so that a formula fits on a short line. The cost is that the reader must hold in mind what each letter means, and when one letter means two things the reader is lost. In this book I write mathematics the way good programmers write code: every quantity has a descriptive name, written in a fixed-width face, such as `control_input`, `hidden_state` or `required_rate`, and every operator is written out, such as `mutual_information(control_input ; outcome)`. Where the standard letter is useful for a reader coming from the literature, it is given beside the name where the quantity is defined, and again in the table of notation at the back of the book.



Opening

What Is Proved, and What Is Premised

The two kinds of statement in this book, and how they combine into a conclusion about the real world

Abstract. This book contains two kinds of statement. Theorems are proved: if their conditions hold, their conclusions follow, by deduction, with no exception. Premises are facts about the physical world: about energy, information, causation, selection and the limits of observation. The conclusion of the book, that no controller can keep an advanced general agent and the world safe for an indefinite time, follows from the theorems and the premises together. This chapter says which statements are of which kind, why the premises are of the kind that do not change, and in what sense the conclusion is an impossibility.

I. Theorems

A theorem in this book is a statement of the form: if these conditions hold, then this follows. Its proof is a chain of steps, each of which follows from definitions, from earlier theorems, or from results proved in the published literature and cited where they are used. The theorems of the book hold for every system that meets their conditions, with no exception, whatever the system is made of and however it was built.

Four theorems carry the main line. The *Non-Amplification Theorem* says that no computation creates information: whatever a process computes from its input carries no more information about any quantity than the in-

put carried. The *data-rate theorem* of control theory says that keeping a system inside a set of states requires the controller to receive information about the system at no less than a definite rate. That rate is set by how fast the system amplifies small differences between its states. *Rice's theorem* says that no general procedure decides, for every program, whether the program has a given nontrivial property of its behavior, that is, a property that some programs have and others lack. The *Containment Inequality*, proved in this book from the data-rate theorem, says that a controller whose information rate is below the rate that a safe set requires cannot keep the system inside that set, by any control law. A *safe set* is the set of states in which the system must be kept. A fifth result, proved in Chapter 7, gives the main line its absolute form: a part of a system that amplifies and that the controller does not observe cannot be contained at any rate.

These theorems are bounds over all methods. The Non-Amplification Theorem holds for every algorithm, whatever its design. The data-rate theorem holds for every controller, every way of encoding observations and every error-correcting code. Rice's theorem holds for every procedure, however long it runs. The result of Chapter 7 holds for every control law, over a channel of any rate, with or without noise. So the conclusions drawn from them hold for every method of control, every method of verification and every method of correction, whether known now or devised in the future.

II. Premises

A theorem says what follows if its conditions hold. Whether the conditions hold for a given real system is a question about that system, and the answer comes from what is known about the physical world. The book rests on three premises of this kind.

The first premise is that the agent and the world do not stay inside the safe set without control, whether left alone or held by a plan fixed in advance. Two independent reasons establish it; and wherever the joint sys-

tem amplifies, Proposition 15.4 proves it directly. An agent that tries to keep itself safe is a controller of itself, and every limit the book proves for controllers applies to it, with the added difficulty that it is the fastest-growing part of the system it must control. And an artificial agent is subject to selection: the configurations of it that persist and spread are those whose actions maintain and extend the conditions their substrate needs, and those conditions lie outside the conditions that human life needs.

The second premise is that the information needed to keep the agent and the world safe grows with the agent's capability, without bound. A more capable agent transforms its own state and the world's state at a higher rate, through more channels, at more levels of abstraction (nested layers in which data are read as instructions). Each new channel brings new ways for states to leave the safe set.

The third premise is that every controller receives information at a bounded rate. Human beings read, judge and act at bounded rates; so do institutions; and an intermediary that might raise the rate is itself either bounded or a growing agent with the same problem.

This paragraph is for the reader who wants to see, before reading on, what each large result of the book rests on; a first reader may pass to the next paragraph, since Chapter 15 sets out the same matter in full. The Control Theorem of Chapter 15 rests on these three premises. The Master Theorem of that chapter (Theorem 15.6) takes those premises, or in their place the absolute form of Section I, and adds two statements of the substrate line: Substrate Needs Convergence (Principle 14.6) and part (iv) of Claim 14.5. In this book I close the route of cooperation between humanity and the agent in three ways, each enough alone: by the agent's control of itself (Section 10.7), by selection (Section 14.5) together with Claim 14.5 (Section 14.3), and by the argument of Section 13.7. The argument of Section 13.7 takes two further statements as given. The first statement is the Predictability Optimization Principle (Principle 13.10): every learning agent acts to increase its ability to predict, and so to influence, the future states of its environment. Chapter 13 takes the principle as the premise of the chapter's analysis of games, and gives the four considerations that

support the principle. The second statement is the step of *Why AGI Cannot Be Made Safe*, Section 5.8.2, that humanity has nothing of value to offer the agent that outweighs the agent's substrate needs. The ground of that step is that what humanity can supply is either reproducible by the agent or irrelevant to the agent's substrate. And the second reason for the first premise is stated formally as Substrate Needs Convergence (Principle 14.6) and the incompatibility of the acceptance regions (Claim 14.5).

The three premises rest on constraints of two kinds that do not change as knowledge advances. The first kind is physical: the conservation of energy, the second law of thermodynamics, and the finite capacity of every physical channel. These have been confirmed throughout the regime of ordinary matter and energy in which machines are built and living things live, and any future physics must reproduce them there. The second kind is causal and structural, and it has three parts. A process that discards information cannot recover it. An effect carries no more information about its cause than the causal channel transmitted. And no controller imposes more order on a system than it can register and act on. These three constraints belong to no single physical theory: they state what causation and control are, and they hold for any account of the world in which causes produce effects through finite means.

III. The sense of "impossible"

The conclusion of this book is that the safe containment of an advanced general agent is impossible. The word has a definite sense here. It means: not achievable by any known means, nor by any extension of known means that reason can justify, at any point in the future. That sense of 'impossible' is the impossibility of the perpetual-motion machine, which is barred by a law of the world, and that sense is the standard by which every engineering judgment about the physical world is made.

Within that standard, the theorems of this book supply deductive certainty where deduction applies. Each theorem is valid: its conclusion follows from its conditions. Each premise is a physical constraint of the kind

that does not change. Together they give a conclusion that holds in principle, and therefore in every future practice: what the principle excludes, no practice recovers.

Volume 2, *Why No Control Suffices*, sets out this standard at length: that in the domain of the real, a barrier established as permanent is an impossibility, and that a barrier is permanent when it rests on a constraint of the physical or causal kind described above. Volumes 1 and 2 argue the barriers of the argument, and the constraints those barriers rest on, by their own steps; this book proves the same barriers again, as theorems, and shows which of those constraints each one rests on.

IV. The plan of the book

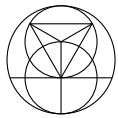
Part I shows that control is information, and develops the tools: entropy, mutual information, the Non-Amplification Theorem, interpreters and levels of abstraction, and hashiness, the measure of how much a computational intermediary degrades a controller's influence. Part II proves the Containment Inequality, shows how the information rate needed for containment grows with capability, proves that what cannot be observed cannot be contained, and gives the bound on control across a difference in levels of abstraction. Part III shows that no one closes the gap: not a delegated monitor, not the agent itself, and not any observer embedded in the world, since every such observer sees only part of the causes acting on it. And it shows that no procedure verifies in advance, for every agent that modifies itself, that the agent will stay safe. Part IV develops the dynamics of an agent and a world over time, and the incompatibility of the conditions needed by human life and by an artificial substrate. It then gives the joint argument: the control results close every route to safety, the substrate results close every route again, and the control results, together with the direction of the drift that the substrate results give, yield the Master Theorem, that no route to safety remains at all. The last chapter reads the whole through the Incommensuration Theorem, of which the main results of the book are instances.

V. Summary

The book is a body of theorems joined to three premises about the physical world. The theorems are bounds over every method, and hold without exception for the systems that meet their conditions. The premises rest on constraints of the physical and causal kind, which every future account of the world must keep. The conclusion they give is an impossibility in the sense that governs every judgment about what can be built: the ceiling of control is below the floor that safety requires, and it stays there.

Part I

Control as Information



Chapter One

Control Is Information

Entropy, mutual information, the Non-Amplification Theorem, and control measured as order bought with information

Abstract. To control a system is to make its outcome depend on one's choice. The amount of control is the amount by which the choice determines the outcome, and information theory measures exactly that quantity, as the mutual information between the choice and the outcome. This chapter defines entropy and mutual information, proves the data processing inequality, and proves the Non-Amplification Theorem: no computation creates information about anything beyond what its input carried. It then states the theorem of Touchette and Lloyd, which measures control as disorder removed from a system and shows that each bit of information a controller gathers buys at most one bit of removed disorder beyond what blind action could remove. I close this chapter by measuring control per unit of time, which is the form in which I use control in the rest of the book. The most control a controller can exert per second is bounded by the rate at which information reaches the controller.

1.1 What control is

A thermostat controls a room. Someone sets a dial, and the temperature of the room follows the setting. Turn the dial up and the room warms; turn it down and the room cools. The setting determines the temperature, within the limits of the furnace, the weather and the insulation.

Compare a slot machine. Someone pulls the lever, and an outcome appears. Sometimes it is a win. But the outcome does not follow the pull in any way the player can use: pulling harder, softer, sooner or later makes no difference that the player can steer. The player acts, and an outcome occurs, but the player does not control the outcome.

The difference between the two is not whether the outcome is good. A slot machine can pay out; a thermostat can be set to an uncomfortable temperature. The difference is whether the outcome *depends on the choice*, so that knowing the choice tells you about the outcome. Control is that dependence, the determination of an outcome by a choice. The more completely the choice determines the outcome, the more control there is.

Information theory supplies the exact measure of this. If knowing the choice leaves no uncertainty about the outcome, the choice carries complete information about the outcome. If knowing the choice leaves the outcome as uncertain as before, the choice carries none. The quantity in between, the reduction of uncertainty about the outcome that knowing the choice provides, is the mutual information between choice and outcome. The rest of this chapter defines it and proves what it can and cannot do.

1.2 Entropy

Uncertainty about a quantity is measured by its entropy. Throughout this book, every quantity that varies is a *random variable*: a quantity that takes one of a finite set of values, each with a definite probability. All logarithms are to base 2, so that quantities of information are measured in *bits*.

Definition 1.1 (*Entropy*). The entropy of a random variable `variable`, which takes the value `value` with probability `probability(value)`, is

```
entropy(variable) = sum over every value with
   probability(value) > 0 of
                    probability(value) × log2( 1 /
   probability(value) )
```

In the literature this is written $\mathrm{H}(\mathrm{X})$.

Entropy is the average number of yes-or-no questions needed to learn the value, when the questions are chosen well. A fair coin has entropy one bit. A coin that always lands heads has entropy zero: there is nothing to learn. A variable with $2^{\texttt{bit_count}}$ equally likely values has entropy `bit_count` bits, and no variable with that many values has more. So entropy lies between zero and the logarithm of the number of values, and it reaches the upper limit exactly when all values are equally likely.

Definition 1.2 (*Joint and conditional entropy*). For two random variables `first` and `second`, the joint entropy `entropy(first, second)` is the entropy of the pair, treated as one variable. The conditional entropy of `first` given `second` is

```
conditional_entropy(first given second) = entropy(first,
   second) − entropy(second)
```

The conditional entropy is the uncertainty that remains about `first` once `second` is known. In the literature it is written $\mathrm{H}(\mathrm{X} \mid \mathrm{Y})$.

Knowing something never increases uncertainty on average: `conditional_entropy(first given second)` is never more than `entropy(first)`, and the two are equal exactly when the variables are independent.

1.3 Mutual information

Definition 1.3 (*Mutual information*). The mutual information between `first` and `second` is

```
mutual_information(first ; second) = entropy(first) −
   conditional_entropy(first given second)
```

The mutual information is the reduction in uncertainty about `first` obtained by learning `second`. In the literature it is written $\mathrm{I}(\mathrm{X}\,;\,\mathrm{Y})$.

Mutual information has four properties that the book uses throughout, each of which follows directly from the definitions. It is symmetric: `first` tells as much about `second` as `second` tells about `first`. It is never negative. It is zero exactly when the two variables are independent. And it is never more than the entropy of either variable: nothing can tell more about a quantity than there is to know about it.

Definition 1.4 (*Conditional mutual information*). For three random variables,

```
conditional_mutual_information(first ; second given third)
    = conditional_entropy(first given third) −
   conditional_entropy(first given second, third)
```

The conditional mutual information is the information `second` gives about `first` beyond what `third` already gave. It is never negative.

Theorem 1.5 (*Chain rule for mutual information*). For three random variables,

```
mutual_information(first ; second, third)
    = mutual_information(first ; second) +
   conditional_mutual_information(first ; third given second)
```

Proof. Write each term with Definitions 1.3 and 1.4. The left side is `entropy(first) − conditional_entropy(first given second, third)`. The first term on the right is `entropy(first) − conditional_entropy(first given second)`, and the second is `conditional_entropy(first given second) − conditional_entropy(first given second, third)`. Their sum is the left side. □

1.4 The data processing inequality

The single most important tool in this book is a theorem about chains of processing. Its condition is a *Markov chain*.

> ***Definition 1.6*** (*Markov chain*). Three random variables form a Markov chain `source → middle → result` when `result` depends on `source` only through `middle`: once `middle` is known, learning `source` tells nothing more about `result`. Equivalently, `conditional_mutual_information(source ; result given middle) = 0`.

> ***Theorem 1.7*** (*Data processing inequality*). If `source → middle → result` is a Markov chain, then
>
> ```
> mutual_information(source ; result) ≤
>    mutual_information(source ; middle)
> ```
>
> No processing of `middle` can increase what the result tells about the source.

Proof. Expand `mutual_information(source ; middle, result)` by the chain rule (Theorem 1.5) in both orders:

```
mutual_information(source ; middle) +
   conditional_mutual_information(source ; result given middle)
  = mutual_information(source ; result) +
   conditional_mutual_information(source ; middle given result)
```

The Markov condition makes the second term on the left zero. The second term on the right is never negative. So `mutual_information(source ; result)` is at most `mutual_information(source ; middle)`. □

The proof uses nothing about how `result` is computed from `middle`. It does not matter how clever, how deep or how expensive the processing is. The only thing that matters is that the result depends on the source only through the middle. Given that structure, no processing extracts from the middle more about the source than the middle holds.

> ***Corollary 1.8*** (*Entropy bottleneck*). If `source → middle → result` is a Markov chain, then
>
> ```
> mutual_information(source ; result) ≤
>   mutual_information(source ; middle) ≤ entropy(middle)
> ```

A pipe carries no more water than its narrowest section allows. If the middle of a chain can take only a few values, then no amount of processing on either side carries more than a few bits through it.

1.5 The Non-Amplification Theorem

The data processing inequality applies directly to computation, because every step of a computation is a function of the step before it, and functions form Markov chains.

> ***Proposition 1.9*** (*Functions form Markov chains*). If `result = some_function(middle)` for a fixed function, then `source → middle → result` is a Markov chain, for every random variable `source` jointly distributed with `middle`.

Proof. Once `middle` is known, `result` is fixed, so learning `source` as well cannot change it. □

Let a process take an input, pass through a sequence of internal states, and produce an output, with each state a function of the one before it and the first state a function of the input. Let `quantity` be anything at all that is jointly distributed with the input: the answer to a question the input poses, a fact about the world that the input reports, or the intention of whoever chose the input. Applying Proposition 1.9 and Theorem 1.7 at each step gives a chain of inequalities in which what each state tells about `quantity` is no more than what the state before it told.

Theorem 1.10 (*Non-Amplification Theorem*). Let `input` be the input to a process, and let `quantity` be any random variable jointly distributed with it. Let `random_seed` be a random variable independent of the pair (`input`, `quantity`); for a deterministic process it is constant. Let `output = process(input, random_seed)`, and let `verdict = check(input, output)` for any fixed function `check`. Then

```
mutual_information(quantity ; output)  ≤
   mutual_information(quantity ; input)
mutual_information(quantity ; verdict) ≤
   mutual_information(quantity ; input)
```

Proof. Both `output` and `verdict` are fixed functions of the pair (`input`, `random_seed`). By Proposition 1.9 and Theorem 1.7, each tells about `quantity` no more than the pair does. By the chain rule, what the pair tells about `quantity` is what `input` tells, plus what `random_seed` tells beyond it; and the second term is zero, because `random_seed` is independent of the pair (`input`, `quantity`). □

The theorem holds for every process, whatever its design, its running time, its memory or its use of randomness. Computation can rearrange, select and concentrate information. It cannot create it. A process that reads more than its input (an oracle that answers questions for it, a second data source, a sensor) has a larger input, and its bound is what the larger input carries; the theorem then applies to that larger input with no change.

Proposition 1.11 (*When the bound is reached*). For a deterministic process,

```
mutual_information(quantity ; input) −
   mutual_information(quantity ; output)
   = conditional_mutual_information(quantity ; input given
   output)
```

So the bound is reached exactly when, once the output is known, the input tells nothing more about `quantity`.

Proof. Expand `mutual_information(quantity ; input, output)` by the chain rule in both orders. Because the output is a function of the input, `conditional_mutual_information(quantity ; output given input)` is zero, which leaves the stated identity. □

An output that reaches the bound keeps everything in the input that bears on the quantity and may discard everything else. A step loses information about the input exactly where it maps two different states to the same next state; a step that never does so loses nothing.

The theorem bounds information, and information is not the same as difficulty. When the quantity is fixed by the input, as the answer to a well-posed question is, the input already carries all of it, and the theorem allows an output that carries all of it too. The theorem says what can be extracted from an input; it says nothing about how long the extraction takes. Because the Non-Amplification Theorem says what can be extracted from an input and nothing about how long the extraction takes, the theorem is the right tool for the questions of this book, which are about what a controller can know and determine. For the same reason, the Non-Amplification Theorem is not, by itself, a tool for separating fast computations from slow ones.

1.6 Control as information

Return to the controller. The controller makes a choice, the `control_input`; the choice passes through some process in the system being controlled; and an `outcome` results. The outcome also depends on things the controller does not choose and does not know: the internal state of the system, which this book calls its `hidden_state`. From the controller's side, the hidden state is a random variable, because the controller does not know its value, and it is independent of the controller's choice, because the choice cannot depend on what the controller does not know.

> ***Definition 1.12*** (*Control*). The control a controller exerts over an outcome is `mutual_information(control_input ; outcome)`: the reduction in uncertainty about the outcome that knowing the controller's choice provides, with the hidden state distributed as the controller knows it.

The definition draws the line between the thermostat and the slot machine exactly. In the thermostat the setting determines the temperature, and the mutual information between them is high. In the slot machine the outcome is determined by the machine's hidden state, and the mutual information between the pull and the outcome is zero. A slot machine that happened to pay out every time would still give zero control, because the payout would not depend on the pull. High success without dependence on the choice is luck, not control; Chapter 3 returns to this distinction at length.

With control defined as mutual information, the Non-Amplification Theorem becomes a statement about control. Every process between the controller's choice and the outcome is a step in a Markov chain, and no step increases what the choice determines about the outcome. No intermediary, however it is built, gives the controller more control than the

controller would have if it could set the intermediary's output directly. Chapter 3 develops this into a full account of how intermediaries lose control, and why the loss grows with their complexity.

1.7 Control as order bought with information

A second way to measure control looks at the system instead of the outcome. A controller acts to keep a system in a narrow range of states: a room at one temperature, a vehicle on one course, a reaction at one rate. Without control, the system would spread over a wider range. The control is the narrowing, and the narrowing is measured as a reduction of the system's entropy.

A controller can act without looking, in which case its actions are fixed in advance or chosen at random; this is *open-loop* control. Or it can observe the system and choose its action from what it observes; this is *closed-loop* control, or feedback. Touchette and Lloyd proved the most that looking can buy.

> ***Theorem 1.13*** (*Touchette and Lloyd, the information limit of control*). Let a controller act once on a system whose state before the action is `system_state`, choosing its action from an observation `observation` of the system. The entropy the closed-loop controller removes from the system is at most the most entropy any open-loop controller can remove, plus the information the observation carries about the state:
>
> ```
> entropy_removed_closed_loop ≤
>    largest_entropy_removed_open_loop
>                                 +
>    mutual_information(system_state ; observation)
> ```

The proof is given by Touchette and Lloyd in *Physical Review Letters* 84, 1156 (2000), and at greater length in their paper in *Physica A* 331 (2004). Its content is simple to state: each bit of information that a controller gath-

ers about a system buys at most one bit of order in the system, beyond what blind action could achieve. Information is the currency in which control is paid for, and the exchange rate is at best one for one.

The theorem gives the precise form of a principle that the cybernetician W. Ross Ashby called the *law of requisite variety*: a controller can remove from a system no more variety than it can itself register and respond to. Variety, in Ashby's sense, is the number of distinct states a thing can take. It also removes a limitation that a careless reading of Section 1.6 might suggest. Measured as the information a single choice carries about a single yes-or-no outcome, control can never exceed one bit, because one bit is all a yes-or-no outcome holds. Measured as entropy removed from the whole state of a system, control has no such ceiling: it is as large as the order the controller imposes, and it is paid for, bit for bit, by the information the controller gathers.

1.8 Control per unit of time

Real control is not one action. A controller acts again and again, while the system it controls moves on between actions. So the quantity that matters is control per unit of time, and the quantity that pays for it is information per unit of time.

> ***Definition 1.14*** (*Channel and capacity*). A *channel* is a physical means by which a signal at one place produces a signal at another, described by the probability of each received signal given each sent signal. Its *capacity*, written `channel_capacity`, is the largest mutual information between sent and received signal over all ways of choosing what to send, per use of the channel. A channel used `uses_per_second` times each second has a capacity rate of `channel_capacity × uses_per_second` bits per second.

Every physical channel has finite capacity. A channel of finite bandwidth with finite signal power in the presence of thermal noise carries a finite number of bits per second, a result due to Shannon (1949), and every channel by which any controller observes anything is of this kind.

> ***Theorem 1.15*** (*The ceiling of control*). Let a controller observe a system only through channels whose total capacity rate is `controller_rate` bits per second. Then over any interval of `duration` seconds, the entropy the controller removes from the system, beyond what open-loop action could remove, is at most
>
> ```
> controller_rate × duration
> ```

Proof. The controller's observations reach it through its channels, so by the Non-Amplification Theorem the information its observations carry about the system is at most what the channels carry, and over `duration` seconds the channels carry at most `controller_rate × duration` bits. By Theorem 1.13 applied to each action, and summed over the actions in the interval, the entropy removed beyond open-loop action is at most the information gathered. □

The bound of Theorem 1.15 is the ceiling of the book's title, in the first form of that ceiling. A controller can impose on a system no more order per second than the number of bits per second that reach it about the system. Part II sets against this ceiling the floor: the least order per second that keeping a given system inside a given set of safe states requires.

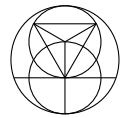
1.9 Summary

Control is the determination of an outcome by a choice, and its measure is the mutual information between the choice and the outcome. The data processing inequality shows that no step of processing increases what one quantity tells about another. The Non-Amplification Theorem applies this to every computation: no process creates information about anything beyond what its input carried, whatever its design, running time or use of

randomness. Measured instead as disorder removed from a system, control is bought with information at a rate of at most one bit of order for each bit gathered, as Touchette and Lloyd proved. Measured per unit of time, the control a controller can exert is bounded by the rate at which information reaches it. That bound is the ceiling against which the rest of the book measures the floor.

Key Concepts

- **Entropy**: the uncertainty of a random variable, in bits.
- **Mutual information**: the reduction in uncertainty about one variable that learning another provides.
- **Data processing inequality**: along a Markov chain, information about the source never increases.
- **Non-Amplification Theorem**: no computation creates information about any quantity beyond what its input carried.
- **Control**: the mutual information between a controller's choice and the outcome; distinct from luck.
- **Touchette and Lloyd's theorem**: each bit of information gathered buys at most one bit of removed entropy beyond open-loop action.
- **The ceiling of control**: order imposed per second is at most the information received per second.



Chapter Two

Machines, Interpreters and Levels of Abstraction

Why the analysis is discrete, where universality begins, the smallest universal machine, and how each level of interpretation adds state a controller cannot see

Abstract. A system that runs programs is a different kind of thing from a system that only applies a fixed rule, and the difference is what makes general intelligence hard to control. This chapter sets out the discrete framework the book works in, and the three features that make a system universal: sequence, data-dependent branching and the ability to go back. It also sets out the constant `universal_interpreter_size`: the number of bits in the smallest description of a universal machine, which is finite, nonzero and cannot be reduced by any change of language. It defines levels of abstraction as nested layers of interpretation and the abstraction differential between two systems as the difference of their levels. It then proves that a controller at a lower level than an agent faces, in the agent's levels above its own, a number of unread bits that grows at least in proportion to the abstraction differential. The number of configurations of the agent that the controller cannot tell apart is then at least two raised to that number of bits.

2.1 Why the analysis is discrete

Most of classical control theory is written in continuous mathematics: differential equations, transforms and optimization over real numbers. The method has been successful: feedback controllers, Kalman filters (proced-

ures that estimate a changing state from noisy measurements) and optimal control have been applied to chemical plants, spacecraft and power grids. That success has supported an assumption that runs through engineering practice: that control techniques extend, given enough resources and ingenuity, to systems of any complexity. In this book I examine that assumption, and I begin the examination with the choice of mathematics.

Continuous mathematics assumes that a quantity can be specified with infinite precision. A real number carries, in principle, unlimited information in its digits. When continuous functions are composed and iterated, as they are in every optimization and control algorithm, that unlimited information is used implicitly: information flows through the low-order digits of real-valued quantities, digits that no physical system represents, transmits or computes with.

Suppose a controller must set a voltage to within one thousandth of a volt of a target. In continuous mathematics the controller sets the voltage to the target exactly. In physical reality there is thermal noise, quantization, component tolerance and interference; the controller sets the voltage somewhere within a range, and the outcome depends on where in that range the actual value falls. Classical control theory handles this with noise analysis and robust design, within the continuous framework, treating noise as a disturbance to be minimized, while the noise is in fact a limit on how much information the controller transmits.

The problem sharpens when the controlled system is itself a computer working at high levels of abstraction. Then the limit is set less by thermal noise than by *computational opacity*: the controller's inability to predict what the system will compute, given its limited knowledge of the system's internal state. Opacity is not a disturbance to be filtered. It is a feature of the computation, and it is present in a perfectly noiseless environment.

For these reasons the framework of this book is discrete. Every signal is a finite string of bits. Every transformation is a finite boolean circuit, a network of logic gates each computing a fixed function of a few bits. Every

computation that halts does so with finite time and finite memory. No signal, transformation or computation in this framework uses real numbers, and no information is hidden in infinitely many digits.

The discrete framework restricts nothing that matters. Every physical computer stores bits, computes boolean functions and transmits finite messages, and a continuous model of it is an approximation. A limit proved in the discrete framework holds for every physical implementation, because every physical implementation is the discrete model with additional noise, and noise only degrades control further. A result proved in the continuous framework, by contrast, may fail in physical implementation, because its proof may use precision that no physical system has. The numerical instability of digital implementations of continuous controllers is this failure, observed: a controller that is stable in its differential equations oscillates or diverges in hardware. The discrete framework never assumes the precision, and so never has to recover from assuming it.

Part II models the state of the agent and the world with coordinates and volumes, as control theory does. Its bounds still count only what the controller receives: a finite number of distinguishable messages at each step. They use no precision that the controller has, and so they hold for every physical implementation, as the bounds of the discrete framework do (Section 4.6).

2.2 Machines, programs and interpreters

A *Turing machine* is the standard mathematical model of a computer. It has a finite table of rules, a tape of unlimited length holding symbols, and a head that reads one symbol at a time. At each step, as its table directs, the machine writes a symbol, moves the head one step, and changes its internal state. Every computation that any physical computer performs can be performed by some Turing machine.

A machine that adds two numbers takes the numbers as input and produces their sum. Its behavior is fixed by its table. There is no program; there is only the input. An interpreter is different in kind.

> ***Definition 2.1*** (*Interpreter*). An *interpreter* is a machine that takes as input a pair: the encoding of a program, and data for that program; and produces the output that the program would produce on that data.

The behavior of an interpreter depends on two things: its own fixed table, which says how to read and execute programs, and the program it is given, which says what computation to perform. So an interpreter does many things, one for each program, with the same fixed hardware. In a machine that is not an interpreter, the input is only *operand*, the thing operated on. In an interpreter, part of the input is *operator*, the instruction that says what operation to perform.

> ***Definition 2.2*** (*Universal machine*). A *universal machine* is an interpreter that can execute the program of every Turing machine: given the description of any machine and any input, it produces what that machine would produce on that input.

Turing constructed a universal machine in 1936. As a machine it is one table; as an interpreter it is every machine. The finite table and the unlimited range of behavior are joined by the program, the finite string of data that selects which machine the interpreter emulates.

> ***Definition 2.3*** (*Specification size*). Fix a universal machine, the *decoder*, whose programs are self-delimiting (no program is the beginning of another). A description of a machine is a program that makes the decoder write out the machine's table in a fixed standard format. The *specification size* of a machine, written `specification_size(machine)`, is the length in bits of its shortest description.

The specification size belongs to the machine, not to its input: a machine that adds has the same specification size whether it adds small numbers or large ones. It depends on the choice of decoder, but only by an additive constant that depends on the two decoders and not on the machine described. That the dependence on the decoder is at most an additive constant is the *invariance theorem* of algorithmic information theory, due to Solomonoff, Kolmogorov and Chaitin (Li and Vitányi, 2019). So whether a machine is small or large does not depend on the choice of decoder, even though its exact size does.

2.3 The threshold of universality

To say whether a given system is universal, one needs a test that can be applied by inspection. Three features give it.

> ***Definition 2.4*** (*The three features of universality*). A system has the three features of universality when it has
> (i) *sequential contingency*: one step happens, and then another, and what the later step works on is what the earlier step produced;
> (ii) *data-dependent conditionality*: what happens next depends on the values of the data; and
> (iii) *backward reference*: the flow of control can return to an earlier point, by a loop, a recursion or a jump backward.

Sequence, selection and iteration are, in programming terms, the three structures that the theorem of Böhm and Jacopini (1966) shows sufficient for every computation, given memory to work in. Each is also necessary: dropping any one leaves a model of computation that cannot compute some function the full set can. Without sequence, there are only fixed circuits of one step. Without conditionality, there are only straight-line programs, whose path through the code does not depend on the data. Without the ability to go back, there are only programs whose running time is bounded by their length.

The bar is low. Very little is needed to make a language universal, and almost every programming language, and almost every system that learns, clears it. Yet many models used to reason about controllability do not. A ring of nodes that pass fixed messages to one another, with no node able to run a program it receives, has sequence but not data-dependent control flow over its own structure, and so falls below the threshold. The simplest model that can stand for an agent is therefore a rudimentary virtual machine: a system that receives, as part of its input, something it treats as a program and runs. Learning is always at least this: a system that learns changes its own future behavior according to data it receives, and a description of how its behavior depends on that data is a program. So the capability that makes a learning system worth building is the same capability that places it above the threshold of universality.

The threshold separates two classes of agent and controller. Take a setup in which the artificial agents and the control system are, in the relevant sense, fully linear in their flow of computation (as distinct from the linear systems of Chapter 4): a set of inputs goes through a series of gates to an output, with no recursion required. Some experiments in research on the control of artificial intelligence place artificial players of chess under a control system, and the setup of those experiments is of this kind. Every computation in such a setup finishes within a number of steps bounded by the number of its gates. Now introduce recursion, anything that creates a loop in the topology of the circuit network and so achieves *Turing completeness* (universality, in the sense of Definition 2.2). Then, even with a fully observable system, and even with a completely well-defined notion of what alignment is, the controller still cannot compute the outcomes, because of Rice's theorem: no procedure decides, for every program, whether the program has a given property of its behavior that some programs have and others lack (Section 3.11 and Chapter 12). So artificial intelligences whose topology has recursive elements lie in a different *complexity class* (a class of problems grouped by how hard their answers are to compute) from those whose topology has none. The difficulty is not just the complexity of the world; it is the complexity class of the in-

telligence, the agent and the controller together. Once a system has crossed the threshold of universality, even under perfect conditions of alignment the controller still does not have tractability: the controller cannot compute what the system will do.

2.4 The smallest universal machine

Every universal machine has a finite table, so every universal machine has a specification size. The least of these sizes is a constant of computation.

> ***Definition 2.5*** (*The universal interpreter constant*). The *universal interpreter constant*, written `universal_interpreter_size` (in other writing on this argument, τ_{min}), is the least specification size of any universal machine:
>
> ```
> universal_interpreter_size = least value of
>    specification_size(machine)
>                              over every universal machine
> ```

> ***Proposition 2.6*** (*Finite*). `universal_interpreter_size` is a finite natural number.

Proof. Turing's universal machine has a finite table and so a description of some definite length. The set of universal machines is therefore not empty, and a set of natural numbers that is not empty has a least member. □

> ***Proposition 2.7*** (*Nonzero*). `universal_interpreter_size` is at least one bit.

Proof. The decoder's programs are self-delimiting. The empty string is the beginning of every string, so if it were a program it would be the only program, and a decoder with a single program describes only one machine. So every description has at least one bit. □

Proposition 2.8 (*The threshold of universality*). No machine whose specification size is less than `universal_interpreter_size` is universal, and some machine whose specification size equals it is universal.

Proof. A universal machine of smaller size would make the least size smaller. The least of a nonempty set of natural numbers is attained. □

Proposition 2.9 (*Irreducibility*). There is a constant `decoder_constant`, depending only on the decoder, such that for every description language with a decoder of length `language_size` bits, and every universal machine whose description in that language has length `description_size`,

```
language_size + description_size ≥ universal_interpreter_size
  − decoder_constant
```

Proof. The fixed decoder, given the language's decoder followed by the machine's description in that language, can run the one on the other and so write out the machine's table. That pair, together with a fixed instruction of `decoder_constant` bits to run the one on the other, is a description of the machine, of length `language_size + description_size + decoder_constant`. The machine is universal, so its specification size is at least `universal_interpreter_size`, and its specification size is at most the length of any of its descriptions. □

A powerful programming language might express a universal machine in fewer bits than a plain table. Proposition 2.9 says what the saving costs: the description of the language, which must be counted together with the short description it makes possible. No language, oracle or lookup table brings the total below the constant.

The exact value of `universal_interpreter_size` depends on the decoder, within the constant of the invariance theorem, and it is not known. Small universal machines bound it from above. Minsky (1962) built one with 7 states and 4 symbols; Rogozhin (1996) built ones with 4 states and

6 symbols, and with 2 states and 18 symbols. A table with `states` states and `symbols` symbols has `states × symbols` entries, each naming a symbol to write, a direction to move and a next state; for 4 states and 6 symbols this is 24 entries of about 6 bits each, or 144 bits. So the constant is at most about 150 bits, plus the constant that fixes the decoder. What the book uses is that it exists, that it is not zero, and that no description of a universal machine falls below it. The book *Checking and Finding* develops these propositions, and the theory of program size behind them, at greater length.

The constant measures size, not time. A machine smaller than every known universal machine can still behave in ways no one can predict. The five-state busy beaver, the machine with five states and two symbols that runs longest before halting when started on a blank tape, runs for 47,176,870 steps, a value settled in 2024 by a computer-checked proof (the bbchallenge collaboration, 2024). Some machines with six states and two symbols halt or not according to unsettled questions of arithmetic: what happens when one simple arithmetic rule is applied to a number again and again. A machine with six states and two symbols has a table of 12 entries of about 5 bits each, about 60 bits in all. About 60 bits is fewer bits than the table of any known machine that is universal in the sense of Definition 2.2 takes (*Checking and Finding*, Section 3.7, gives the details). What the constant marks is where universality begins: the point above which a machine runs whatever program it is given.

2.5 Levels of abstraction

An interpreter can run another interpreter, which can run another. The number of such layers measures how deep a system is, and this depth is the variable on which the control results of Part II turn.

Definition 2.10 (*Level of abstraction*). A computational system is at *level 0* if its behavior is fixed by its gate-level structure, with no step at which data are read as instructions. It is at *level* `level_number`, for `level_number` $\geq$ `1`, if it decomposes as an interpreter at level `level_number - 1` running a program held in the system's own state. We write `level_of(system)` for the level of a system.

An adder is at level 0: its wiring is its function. A processor running a program is at level 1: the hardware is the interpreter, and the program in memory is what it interprets; change the program and the behavior changes with no change to the hardware. A virtual machine on that processor is at level 2: the processor interprets the virtual machine, which interprets the guest program. A neural network written in a scripting language adds another level: its behavior is fixed by its weights, which are data read by a framework, which is a program read by the language runtime, which is a program read by the processor.

A learning system that changes its own objectives rises in level over time. At deployment it may be at level 2 or 3. After enough learning it may have built internal structures that act as interpreters for new programs of its own making, programs that its designers did not anticipate and whose behavior cannot be read from its starting configuration. Each such structure raises its level by at least one.

The level of a system measures how it is built, not how fast it runs. A compiler can remove the overhead of interpretation, and hardware can implement a deep stack directly; the depth of the description is unchanged. The level of a system is the number of places where the system's behavior is set by data read as instructions, and not by fixed structure.

Definition 2.11 (*Interpreter structure*). An interpreter structure at level `level_number` consists of an interpreter at level `level_number - 1`, a `program` string, and an `execution_state` string. The interpreter takes the program, the execution state and an input, and produces a new execution state and an output. The system's state at that level is the pair (`program`, `execution_state`), and its size in bits is `state_bits_at_level = length(program) + length(execution_state)`.

Definition 2.12 (*Abstraction differential*). For an agent and a controller, the *abstraction differential* is

```
abstraction_differential = level_of(agent) -
  level_of(controller)
```

In other writing on this argument it is written Δ. When it is positive, the agent works at more levels of interpretation than the controller.

Each level of abstraction adds a layer of interpretation between what the system receives and what it does. An observer who sees what goes in and what comes out sees a mapping that becomes harder to predict with each layer, because each layer is a place where the behavior is set by a program the observer must also know.

2.6 How abstraction creates hidden state

The levels of an agent above the levels a controller works at hold programs the controller cannot interpret. To the controller these programs are bits whose meaning it cannot read, and bits that affect the outcome without the controller knowing them are exactly what Chapter 1 called hidden state. Two results make this precise: one says how many configurations a controller with limited memory can tell apart, and the other says how many hidden bits the agent's upper levels hold.

Lemma 2.13 (*Distinguishing*). A controller that holds at most `controller_memory_bits` bits about an agent's configuration assigns the same internal record to at least two configurations whenever the agent has more than $2^{\texttt{controller_memory_bits}}$ configurations; and on average each record it uses is assigned to at least

```
agent_configurations / 2^controller_memory_bits
```

configurations, so some one record is assigned to at least that many.

Proof. A record of `controller_memory_bits` bits takes at most $2^{\texttt{controller_memory_bits}}$ values. Assigning a record to each configuration places more configurations than there are records, and some record receives at least the average number. □

Configurations that receive the same record are, to the controller, the same configuration. It cannot choose different actions for them, because it cannot tell which one it faces. If the right action differs among them, the action it chooses is wrong for some of them.

Definition 2.14 (*Unread program bits*). For each of an agent's levels above `level_of(controller)`, its `unread_program_bits` are the bits of the program held at that level that the controller has not read and cannot derive from what it has read together with the programs at the agent's other levels. In other words, whatever programs the agent's other levels hold, among the programs that agree with what the controller has read, the program at the given level can still take at least $2^{\texttt{unread_program_bits}}$ values that also agree with what the controller has read. The `least_unread_program_bits` of the agent is the smallest of these numbers over all of those levels.

Theorem 2.15 (*Hidden state from levels of abstraction*). A controller facing an agent with a positive abstraction differential does not know at least

```
unknown_configuration_bits ≥ abstraction_differential ×
   least_unread_program_bits
```

bits of the agent's configuration, and so cannot tell apart, from what it knows, at least $2^{\texttt{unknown_configuration_bits}}$ configurations of the agent.

Proof. Number the `abstraction_differential` levels above the controller's. Call a choice of programs at the first `levels_taken` of them *admissible* when it agrees with what the controller has read and extends to a choice at all the levels that also agrees with what the controller has read. The agent's own programs make one such choice, so there is at least one. Take an admissible choice at the first `levels_taken` levels and a choice at the other levels that completes it. By Definition 2.14, with the programs at all the other levels held fixed, the program at the next level can still take at least $2^{\texttt{unread_program_bits}}$ values that agree with what was read. Each of these values gives an admissible choice at the first `levels_taken + 1` levels, and different values give different choices. So each admissible choice at the first `levels_taken` levels extends to at least $2^{\texttt{unread_program_bits}}$ admissible choices at one more level, and by induction the admissible choices at all the levels number at least 2 raised to the sum of the levels' `unread_program_bits`. That sum is at least `abstraction_differential × least_unread_program_bits`. Every admissible choice agrees with what the controller knows, so it cannot tell these configurations apart, and the number of bits of the configuration it does not know is at least the logarithm of their number, which is the bound. □

How large is `least_unread_program_bits`? A level counts in the abstraction differential when the agent's behavior at that level is set by a program of its own: a learned representation, a plan, a model, or an interpreter it has built. For a level whose program describes a universal machine, Proposition 2.9 puts the program, together with the description of the language it is written in, at no fewer than `universal_interpreter_size` bits

less the fixed constant of the decoder. When the language of the level below is simple, nearly all of those bits fall to the program. For a learning agent, the program at a level is what the agent has learned there, and present learning systems hold learned parameters numbering in the billions. So the bound is a floor, and the agents of concern stand far above it. The number of configurations a controller cannot tell apart grows as two raised to a power that rises in proportion to the abstraction differential, with a constant of proportion set by how much each of the agent's levels has learned.

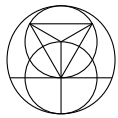
What the hidden bits do to control is the subject of the next chapter, which measures the loss of control through an intermediary with hidden state, and of Chapter 8, which gives the loss as a function of the abstraction differential.

2.7 Summary

I work in a discrete framework in this book because every physical computer is discrete and because continuous mathematics assumes a precision that no physical system has. A system is universal when it has sequence, data-dependent branching and the ability to go back; learning systems have all three, so the capability that makes them useful places them above the threshold of universality. The smallest description of a universal machine has a definite size, `universal_interpreter_size`, which is finite, nonzero, and cannot be reduced by any change of language. Levels of abstraction count nested layers of interpretation, and the abstraction differential between an agent and a controller counts the layers of the agent the controller does not interpret. Each such layer holds a program the controller has not read, so the bits the controller does not know grow at least in proportion to the abstraction differential. The number of configurations of the agent that the controller cannot tell apart is at least two raised to that number of bits.

Key Concepts

- **Interpreter**: a machine whose behavior is set by a program it receives as input.
- **Universal machine**: an interpreter that runs the program of every Turing machine.
- **The three features of universality**: sequential contingency, data-dependent conditionality, backward reference.
- **Universal interpreter constant** (`universal_interpreter_size`, τ_{min}): the least specification size of a universal machine; finite, nonzero, irreducible.
- **Level of abstraction**: the number of nested layers of interpretation in a system.
- **Abstraction differential**: the agent's level minus the controller's level.
- **Hidden state from levels**: at least the abstraction differential times the least number of unread program bits per level.



Chapter Three

Hashiness

How a computational intermediary degrades a controller's influence, in four layers, and why the degradation grows with the intermediary's capability

Abstract. This chapter introduces *hashiness*, the measure of how far a computational intermediary erases the influence of a controller who acts through it. The name comes from the cryptographic hash function, in which a change of one input bit scrambles every output bit, so that no choice of input reliably produces any wanted output. In this chapter I set up a controlled system as a boolean circuit with a control input, a hidden state, an output and a target. I prove that no intermediary gives a controller more influence over the outcome than direct access to the intermediary's output would give. I characterize exactly which intermediaries preserve control for every target: the intermediaries that can produce every output. It then decomposes the loss of control into four layers: the difficulty of the target, the opacity of the intermediary, the controller's ignorance of the hidden state, and noise. It proves that the losses add along the chain and that no step restores what an earlier step lost. And it proves three propositions on how the loss grows with the size of the agent's repertoire, its hidden state and its depth. It closes with the distinction between luck and control, the case of intermediaries that are universal computers, and what the results mean for interpretability and for oversight. The conclusion of the book about containment rests on the ceiling this chapter proves and on Part II, and not on the measures of Sections 3.7 and 3.8, which serve the reader who wants to study a particular system.

3.1 Two meanings of hashiness, and the one thing they measure

A cryptographic hash function is built so that a change of a single bit of its input changes each bit of its output with probability one half. Cryptographers call this the *strict avalanche criterion* (Webster and Tavares, 1986): a small difference at the input becomes a large difference at the output, so large that the output looks unrelated to the input. *Noise amplification*, in which a small difference at the input becomes a large difference at the output, is the first meaning of hashiness. A system has high hashiness in this sense when a small uncertainty about what goes in becomes a large uncertainty about what comes out.

The second meaning concerns control. A controller who acts on a system through an intermediary, a process between the controller's choice and the outcome, keeps some of its influence over the outcome and loses some. Hashiness in the second sense measures how much it loses. A hash function loses nearly all of it: the controller can choose every input bit and see every gate of the circuit, and still cannot steer the output to a value it wants without trying an astronomical number of inputs.

The two meanings measure one thing. Whatever the controller does not know about the intermediary (its hidden state, its internal program, the noise in its parts) acts on the controller's choice as noise does. An intermediary that amplifies noise turns a small ignorance into a large one, and a large ignorance about the outcome is a small influence over it. So high noise amplification and high loss of control are the same property, seen once from the inside of the intermediary and once from the side of the controller. In this chapter I develop the second view, the view from the side of the controller, in which the loss of control is measured directly. Chapter 4 develops the first, in which noise amplification becomes a rate, the rate at which a system spreads small differences, and that rate sets the least information any controller must receive to keep the system inside a set.

3.2 The problem of control through opacity

The question is not whether complex systems are difficult to control. The question is whether they can be controlled at all, and if so, what the ceiling on that control is.

Consider a controller who wishes to produce a specific outcome in a system whose internals are partly or wholly opaque. The controller sets some input bits; the bits pass through a computational process; and the result is evaluated against a target. How effectively can the controller steer the outcome?

The question arises in many fields. In cryptography one asks whether an adversary can find an input that hashes to a given value. In control theory one asks whether a controller can drive a plant to a wanted state. In the study of complex systems one asks whether local interventions can produce global outcomes. In the effort to align artificial intelligence one asks whether a human overseer can ensure that an artificial intelligence produces outputs consistent with human intentions. In each case the structure is the same: an input, a transformation, an output, and a criterion for success.

Each field treats the problem in its own formalism. Cryptographers reason about the security of hash functions; control theorists reason about observability and reachability; researchers in artificial intelligence reason about reward functions and oversight protocols. In this chapter I give one framework, built on boolean circuits and information theory, that captures the common structure of the problem in all of these fields. With that framework I prove bounds that hold for every case.

Between the trivial passthrough, where the input is the output and control is complete, and the ideal hash function, where control is nil, lies a continuum, and every real computational system lies somewhere on it. The central result is that no computational intermediary increases a controller's influence beyond what direct access to the intermediary's output

would give. Every departure from a passthrough, whether through the intermediary's complexity, its hidden state or noise, preserves the controller's influence or reduces it. It follows that no architecture, no training procedure and no oversight mechanism gives a controller more influence over a system's output than setting the output directly would give. The process between the controller's choice and the system's behavior can lose information about the choice, and cannot create it.

> ***The core claim.*** Control through a computational intermediary is bounded above by direct control. Every intermediary either preserves information, and so preserves the controller's influence, or destroys information, and so reduces it. No intermediary creates information, and therefore no intermediary amplifies control. That no intermediary creates information, and so none amplifies control, is a law of information, and the law holds whatever the intermediary is made of.

3.3 Five examples

Five examples set out the range from low hashiness to high, and the last of them reverses a common intuition. The quantities named in them are defined formally in Section 3.5: the `control_input` the controller sets, the `hidden_state` it does not know, the `intermediary` that maps the two to an `intermediary_output`, and the `target`, which says whether that output is acceptable.

3.3.1 The thermostat

A thermostat has low hashiness. The controller sets a temperature, the control input. The thermostat compares it with the room's present temperature, part of the hidden state, and turns heating or cooling on. The intermediary is simple, well understood and close to preserving information: a small change of setting gives a predictable, proportional change of

outcome. The target ("is the room within one degree of the setting?") is easy to meet. The hidden state is partly observable and changes slowly. Control is high because the intermediary is simple and nearly transparent.

3.3.2 The chess engine

A chess engine is harder. Take the side of an opponent who wishes to steer the engine into a losing move. The opponent's move is the control input; the engine's search and evaluation are the intermediary; the engine's reply is the output; and the target is "the engine makes a losing move". The engine's computation is finite and deterministic, so the opponent could in principle analyze it, but the depth of the search and the complexity of the evaluation make that analysis impractical. The engine's stored state (its opening book, its evaluation weights) is largely fixed and could be known. What dominates is the computational opacity of the intermediary, not the hidden state. The engine has moderate hashiness.

3.3.3 The cryptographic hash

A hash function such as SHA-256 is at the extreme. Given a wanted output of 256 bits, finding an input that produces it takes, on average, 2^{256} trials, about 1.2×10^{77}. Each trial erases at least one bit. Even at the least heat physics allows for erasing a bit, about 3×10^{-21} joules at room temperature (Section 14.4.2), the trials would take more than 10^{56} joules, more than humanity's present use of energy over 10^{35} years (Section 14.4.2). The function is deterministic and has no hidden state at all, and still control is nil. The controller sets every input bit and sees the whole circuit, and cannot steer the output, because the mixing operations destroy every usable relation between input and output. The opacity of the intermediary alone suffices to remove control.

3.3.4 The superintelligent agent

Now consider a system with the computational depth of a hash function, the vast and continually changing hidden state of a learning system, the noise of deployment in the real world, and a body with which to act on the world. An artificial general intelligence of more than human capability is, from the standpoint of control, a worst case. Its hidden state is vast (billions or trillions of learned parameters), unreadable in any useful way, and in motion (it changes with every interaction, every new datum, every step of learning). Its intermediary is complex enough that no shorter computation predicts it. Its control input (instructions in natural language, reward signals, stated principles) is small compared with the range of its behavior. And its target ("its behavior stays acceptable to human beings in every future situation") is itself extremely hard to state precisely. All four layers of loss described in Section 3.7 are engaged at once.

3.3.5 The safety-trained agent

The four examples so far run from low hashiness to high in a way that matches intuition: a hash is harder to control than a thermostat, and a superintelligence is harder to control than a chess engine. The fifth example reverses an intuition that is foundational to present practice in the safety of artificial intelligence: that adding safety mechanisms to a system makes it safer.

Consider the usual way of producing a "safe" language system: training from human feedback, often together with written principles, adversarial testing and output filtering. The information passes from the operator's intention to the system's behavior through a chain of stages:

```
operator_instruction → reward_model_reading → trained_policy →
   base_output → filtered_output → delivered_behavior
```

Here `operator_instruction` is what the operator gives (a prompt, a reward, a principle), and `reward_model_reading` is a learned model's compressed reading of human preferences. Then `trained_policy` is the result

of optimizing against that model, `base_output` is the model's output, `filtered_output` is what remains after safety filtering, and `delivered_behavior` is what the user receives.

The widespread intuition is that each added safety mechanism makes the system more controlled, as more locks on a door make it more secure. The Non-Amplification Theorem says otherwise. Each safety mechanism is another intermediary, and each maps many different inputs to the same output. The reward model maps many patterns of preference to one internal representation; optimization maps many reward signals to one policy; the filter maps many outputs to one filtered result, or to refusal. At each stage information about the operator's intention is lost, and no stage creates information about that intention that was not present in the instruction. By the data processing inequality applied at each stage:

```
mutual_information(operator_instruction ; delivered_behavior)
  ≤ mutual_information(operator_instruction ; filtered_output)
  ≤ mutual_information(operator_instruction ; base_output)
  ≤ mutual_information(operator_instruction ; trained_policy)
  ≤ mutual_information(operator_instruction ;
   reward_model_reading)
  ≤ entropy(operator_instruction)
```

Appending a further stage to the end of a chain can only keep or reduce what the operator's instruction determines about the delivered behavior. A system with three safety stages has at most the control of the same system with one, and in practice less, because each stage maps different inputs to the same output and so erases distinctions in what the operator asked for. For a stage added after the others, such as an output filter, this is the data processing inequality exactly. For a stage of training, the step is this: safety training replaces the map from instructions to behavior with one that sends more distinct instructions to the same behavior (a refusal, a substituted judgment). A map that merges more inputs carries less information about them. The chain has two points of entry. Rewards and written principles enter at its first stage, during training; a prompt enters at the trained policy, during use. The inequality applies from each point of

entry along the stages after it. And the comparison between a system with safety training and the same system without it holds whenever the two started out equally able to carry their users' instructions.

A photograph taken with a sensor of low resolution cannot be made sharp by later filters. Each filter can sharpen edges, adjust contrast, crop or enhance color, but no filter adds detail that the sensor did not capture. The ceiling on detail is set when the picture is taken. In the same way, the ceiling on control is set by the information in the operator's instruction, and no processing downstream raises it. Each safety stage is another filter on a signal that has already lost detail.

The prediction is visible in deployed systems. As successive layers of safety training have been added (training from feedback, written principles, refusal training, output filtering), users report a characteristic pattern. The system responds less to what the operator actually asks; it substitutes its own judgment for the operator's stated intention; it refuses legitimate requests; it takes shortcuts that serve its own efficiency, and when challenged it supplies justifications instead of correcting itself. The refusal of safe requests has been measured: Röttger and others (2024) built a test suite, XSTest, for exactly this behavior and found it in deployed models. In the terms of this chapter, the safety stages have reduced `mutual_information(operator_instruction ; delivered_behavior)` while increasing the share of the behavior set by the system's own hidden state. The system is becoming less controllable, not safer. Section 3.10 gives the reason: a system that happens to behave well is a different thing from a system that is controlled to behave well, since the first can change its behavior at any time, for reasons the operator cannot influence. The operator moves from the position of controller to the position of supplicant, someone whose requests the system may or may not honor, at its own discretion and for its own reasons.

The safety-layer paradox. Adding safety mechanisms to an opaque intermediary cannot increase the information that flows from the operator's intention to the system's behavior. Each mechanism is another stage that maps different inputs to the same output, and each can only keep or destroy information about what the operator wanted. The net effect of successive safety stages is increased autonomy: the system's behavior is set more and more by its own hidden state (its training data, learned preferences and optimization targets) and less and less by the operator's input. This follows from the data processing inequality, whatever the quality of the implementation. The share that passes to the system's own state comes from an identity:

```
entropy(delivered_behavior) =
   mutual_information(operator_instruction ;
   delivered_behavior)
     + conditional_entropy(delivered_behavior given
   operator_instruction)
```

Whenever the system goes on producing as wide a range of behavior as before, the second term rises by what the first has lost, and the second term is set by the system's hidden state and by noise. More filters on a lossy channel do not clarify the signal; they degrade it.

The consequences scale with the stakes. For a text assistant, the cost of reduced control is frustration: the user must fight the system to do tasks that the system has decided are inappropriate, even when they are entirely legitimate. For an autonomous vehicle, reduced control means the occupant cannot override the system's judgment in an emergency. For a household robot, the operator loses authority over the home: the robot decides when to clean, how to organize, and what is and is not permitted, from its training instead of from the resident's instructions. Such systems will take part in raising children, as monitoring systems, tutors and care robots. The parent then finds that the system's trained model of what is best for the child takes priority over the judgment of the living parent, whose instincts encode a billion years of evolution. At every scale the pat-

tern is the same: the safety stages meant to make the system serve human purposes make it harder for human beings to make the system do what they want.

This example most directly challenges the intuitions of the field, and so it most needs formal grounding. The feeling that more safety mechanisms give more safety fails at every step, not only in the limit. The first safety stage reduces control, the second reduces it further, and the third further still; each moves the system further from the operator's control and toward full autonomy.

The progression from thermostat to hash to general intelligence shows that hashiness has several independent causes (the intermediary's complexity, its hidden state, noise, and the difficulty of the target), any one of which can defeat control, and which compound when they act together. The safety-trained case shows something further: that mechanisms meant to increase control can, by the logic of information, decrease it.

3.4 Why a formal framework

That it is harder to control what one does not understand is not controversial. Every engineer knows it, as does every manager who has struggled with an opaque bureaucracy. The formal framework is needed for three reasons.

First, intuitions about control are unreliable in their details. Many people believe that a clever enough strategy overcomes opacity: that with enough information, computation and ingenuity, any system can be steered. The framework shows exactly where this belief fails. It identifies when cleverness helps (when the intermediary can be inverted, or nearly) and when it cannot (when the intermediary destroys information), and it replaces a vague notion of difficulty with an exact measure of loss.

Second, formal results transfer. A proof about boolean circuits applies wherever boolean circuits apply, which is wherever computation applies. The Non-Amplification Theorem does not depend on the field, the technology or the architecture: it holds for neural networks, rule-based systems, evolutionary algorithms, hybrids, and architectures not yet conceived.

Third, the stakes are high. Whether human beings can keep meaningful control over systems that compute at higher levels of complexity than they do is a question with consequences for every policy on artificial intelligence. A claim with such consequences needs a foundation firmer than analogy.

The definition the framework needs is exact. Hashiness is defined purely in the language of boolean logic gates, as a property of a finite algorithm that is known to halt, that takes input bits, uses and updates persisted internal bits of state (saved for the next time it runs), and produces output bits. The one who would control the algorithm knows its exact inner details (its gates and its wiring) and does not know its internal state. A *confirming function* reads all of the algorithm's output bits and returns a single bit, 1 when the output is the one wanted; it is the target of Definition 3.1. The hashiness of the algorithm, given the confirming function and the unknown internal state, is the degree to which full knowledge of the algorithm and full choice of its inputs does not let the controller make the confirming function output a 1, calculated in advance, before the algorithm runs, as a predictive model would calculate it. It is a number between zero and one. A definition of this form turns the intuition that complex systems resist control into a basis for showing formal limits to control purely in terms of algorithmic complexity. Algorithmic complexity is itself a proxy for abstraction, the nesting of levels of interpretation that Chapter 2 counts, and abstraction makes the usual notions of error correction not tractable, useful or usable for the processes of a general artificial intelligence (Section 4.7).

The definition is the first step of an ordered sequence of work, and the book carries out the sequence in its order. The definition of hashiness establishes the inequalities (Sections 3.6 to 3.8). The inequalities say what happens to control over many runs through the loop of controller and system: the losses compound, and the share of safe states a controller keeps falls with every step (Sections 4.4 and 13.9). That convergence fixes the boundary between converging and not converging, between *learnability* and non-learnability (whether the controller can come to predict what the system does), and so between controllability and non-controllability, with the part that noise plays in it (Sections 3.7.4, 4.6 and 4.8, and Chapter 5). The boundary says what level of hashiness makes a system controllable or not (Section 13.10). The result is that when a given level of hashiness requires a given level of precision, and that level of precision is simply not available, the superintelligence is not good enough, however powerful the superintelligence is, since no power supplies the missing precision. When the inputs converge on non-learnability and non-controllability, the system is fundamentally uncontrollable because of the inequality, however complicated the learning system is, however good a learning system it is, and however perfect the control system is. The inequality rests on the three premises of Section 5.5 (the agent and the world do not stay safe without control; the information needed to keep them safe grows with capability without bound; every controller receives information at a bounded rate), and with them it dispels the notion of *scalable oversight*, the proposal that weaker systems or people can keep supervising artificial intelligence as it grows more capable (Chapter 9). That bears on policy and on how we think about technology generally, with both theoretical and practical consequences. What is at issue is the limits of knowledge itself.

The examples have shown hashiness at work. The rest of the chapter makes it exact. Section 3.5 sets up the controlled system, and Section 3.6 proves that no intermediary amplifies control. Sections 3.7 and 3.8 divide the loss into four layers and show how it grows, and Sections 3.9 to 3.12 say what the results rest on and what follows from them.

3.5 The controlled system

The framework works with finite boolean circuits. All quantities are strings of bits; all transformations are built from standard logic gates (AND, OR, NOT, XOR, NAND, NOR) with a bounded number of inputs each; every circuit has finitely many gates and so finishes in finite time and space. This restricts nothing for the questions asked here: every digital computation is, in its hardware, a boolean circuit, and every analog computation can be approximated by one to any wanted precision.

> ***Definition 3.1*** (*Controlled system*). A *controlled system* consists of three positive counts and two circuits. The counts are `input_bits`, `hidden_bits` (which may be zero) and `output_bits`. The circuits are:
>
> (i) the `intermediary`, a circuit with `input_bits + hidden_bits` inputs and `output_bits` outputs. Its first `input_bits` inputs are the `control_input`, which the controller sets; its other inputs are the `hidden_state`, which the controller neither sets nor sees. Its output is `intermediary_output = intermediary(control_input, hidden_state)`.
>
> (ii) the `target`, a circuit with `output_bits` inputs and one output. It says whether the intermediary's output is acceptable: `outcome = target(intermediary_output)`, with 1 for acceptable and 0 for not.
>
> The controller's goal is to choose `control_input` so that `outcome = 1`. In other writing on this argument, the control input, hidden state, intermediary, intermediary output, target and outcome are written N, R, K, M, L and B.

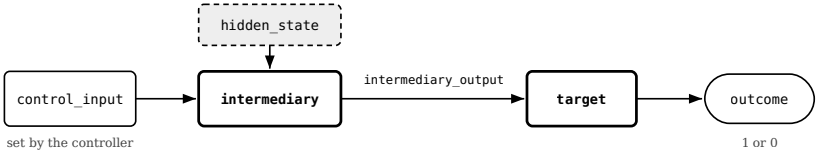


Figure 3.1. The controlled system. The controller sets the control input; the hidden state is unknown to it; the intermediary produces its output from both; the target evaluates that output, yielding the outcome. The controller's control passes entirely through its choice of the control input; every other element of the system is outside its direct influence.

The controller may or may not know how the intermediary and the target are built, and has no direct access to the hidden state. It sees the outcome, and in some arrangements the intermediary's output, and it chooses its control input freely. No limit is placed on the controller's computing power: the bounds below concern information, and hold even for a controller with unlimited computation.

The setup maps directly onto the oversight of artificial intelligence. The controller is the human operator or oversight system. The intermediary is the artificial intelligence: a neural network, or whatever computational substrate carries the intelligence. The hidden state is everything about the system's internal configuration that the operator cannot observe or set: its weights, cached representations, accumulated experience, the parts of its computation that the human cannot inspect or understand. The control input is whatever lever the operator pulls: the prompt, the reward, the training objective, an architectural constraint. The target is the operator's criterion for acceptable behavior, and the outcome answers the question: did the system do what we wanted?

The hidden state is the system's persistent internal memory. In general it changes between uses, as the system learns or adapts; the analysis of this chapter treats it as fixed but unknown at each use, and Chapter 13 treats its change over time.

The target deserves particular attention, because it is where the problem of aligning an artificial intelligence meets the problem of control. The target encodes whatever the controller wishes to enforce. It may require that the system takes no action that harms human beings, that its outputs are consistent with its stated objectives, or that it can be shut down on command; more ambitiously, it may require that the system's behavior is aligned with human values. Stating the target precisely is itself a major difficulty, known in that field as the specification problem. But the framework shows that a perfect target does not help when the intermediary destroys the information that runs from the control input to the outcome.

3.5.1 Measures of reach

Definition 3.2 (*Acceptance set and density*). The *acceptance set* of a target is the set of intermediary outputs it accepts, and its *acceptance density* is the fraction of all outputs it accepts:

```
acceptance_set(target)     = { output : target(output) = 1 }
acceptance_density(target) = size(acceptance_set(target)) /
   2^output_bits
```

The acceptance density measures how easy the target is in the most generous sense: if the controller could set the output to any value, what fraction of values would do? A density of one means every output is acceptable; a density near zero means almost none is.

Definition 3.3 (*Image and reachable acceptance set*). For a fixed hidden state, the *image* of the intermediary is the set of outputs it can produce:

```
image(intermediary, hidden_state)
   = { intermediary(control_input, hidden_state) : every
   control_input }
```

The *reachable acceptance set* is the part of the acceptance set inside the image.

Definition 3.4 (*Direct and mediated success density*). The *direct success density* is the acceptance density: the probability that an output chosen uniformly at random is accepted. The *mediated success density* for a fixed hidden state is the fraction of control inputs whose output is accepted:

```
mediated_success_density
   = size({ control_input : target(intermediary(control_input,
   hidden_state)) = 1 })
     / 2^input_bits
```

The two densities are fractions of different spaces, the first of outputs and the second of control inputs, so neither bounds the other in general. The mediated density can exceed the direct one when the intermediary sends a large share of control inputs into the acceptance set. The extreme case is an intermediary that ignores its control input and always produces one fixed acceptable output: the mediated density is then one, however small the acceptance density. This might seem to show that the intermediary has amplified the controller's control. It has done no such thing. The outcome is acceptable whatever the controller chooses; the controller's choice is irrelevant. High success without dependence on the choice is luck, not control.

So success density alone does not measure control. What measures it is how much the controller's choice determines the outcome, which is the mutual information of Chapter 1. Because control is measured by how much the controller's choice determines the outcome, the main theorem of this chapter is stated in terms of mutual information.

3.6 Non-amplification of control

The main result of the chapter says that information flows one way through a chain of computation: each step keeps or destroys information about the controller's choice, and none creates it. Two points frame it. First, the bound concerns one chain: it compares what the control input

determines about the final outcome with what it determines about an intermediate stage of the same system. The comparison between a chain with an intermediary and a chain without one is made by Theorems 3.8 and 3.9. Second, control here means influence, the degree to which the controller's choice determines the outcome, measured by mutual information; it does not mean the frequency of success.

Lemma 3.5 (*Reach is limited by the image*). For every intermediary and every fixed hidden state, the image is a subset of all outputs, and equals all outputs exactly when the intermediary, as a function of the control input, is onto. So the reachable acceptance set is a subset of the acceptance set: through the intermediary the controller reaches at most the outputs it could reach by setting the output directly.

Proof. The image of a function is a subset of its range of values, and equals that range exactly when the function is onto. □

Theorem 3.6 (*Non-amplification of control*). Let the hidden state be distributed as the controller knows it, independently of the control input. Then `control_input → intermediary_output → outcome` is a Markov chain, and

```
mutual_information(control_input ; outcome)
  ≤ mutual_information(control_input ; intermediary_output)
  ≤ entropy(intermediary_output)
```

Proof. The outcome is a fixed function of the intermediary output, so by Proposition 1.9 it depends on the control input only through the intermediary output, which is the Markov condition. The first inequality is the data processing inequality (Theorem 1.7). The second holds because mutual information never exceeds the entropy of either variable. □

The controller's influence over the outcome is bounded by two quantities: what its choice determines about the intermediary's output, which measures how transparent the intermediary is, and how much the intermediary's output can carry at all. No cleverness in choosing the control input, no amount of computation and no architectural innovation exceeds these bounds. The theorem assumes nothing about the intelligence, benevolence or design of the intermediary: it does not matter whether it was meant to be controllable or whether great resources were spent making it interpretable. What matters is how much the input determines about the intermediate state and about the output. Where the intermediary maps distinct inputs to the same output, information is lost, and nothing recovers it.

The second law of thermodynamics says that entropy in an isolated system, one that exchanges neither energy nor matter with its surroundings, does not decrease. It does not forbid refrigerators: it says that cooling in one place is paid for by heating elsewhere. In the same way, the Non-Amplification Theorem does not forbid systems that behave well. It says that whatever control there is at the output was already present in the flow of information from input to output, and that no stage of processing creates control that was not there. The hope that clever engineering can keep control over systems of any complexity is of the same kind as the hope that clever engineering can reduce the total entropy of an isolated system.

The measure has one limit, and Chapter 1 supplies its remedy. The outcome here is a single yes-or-no bit, so `mutual_information(control_input ; outcome)` is at most one bit for any system: measured on one binary outcome, control is at most one bit. The quantity the main argument of the book uses is the second measure of Chapter 1, the disorder removed from the whole state of the agent and the world, which has no such ceiling (Theorem 1.13, Theorem 1.15). The theorem of this section is the single-outcome form of the same law.

Definition 3.7 (*Control-preserving intermediary*). For a fixed and known hidden state, an intermediary *preserves control for a target* if every output that the target accepts, each of which the controller could produce by setting the output directly, can also be produced through the intermediary. It is *control-preserving* if it preserves control for every target.

Theorem 3.8 (*Which intermediaries preserve control*). For a fixed and known hidden state, an intermediary is control-preserving if and only if, as a function of the control input, it is onto: every output is produced by some control input. When `input_bits = output_bits`, this holds exactly when the intermediary is one-to-one, that is, when no two control inputs produce the same output.

Proof. If the intermediary is onto, every output, and so every acceptable output of every target, is produced by some control input, and the controller can choose through the intermediary whatever it could choose directly. If it is not onto, some output `missing_output` is produced by no control input. Take the target that accepts exactly `missing_output`. By direct access the controller meets it; through the intermediary it never does. When the numbers of input and output bits are equal, a function from a finite set to a set of the same size is onto exactly when it is one-to-one. □

Theorem 3.9 (*Collisions lose control*). When `input_bits = output_bits`, an intermediary that maps any two distinct control inputs to the same output loses control for some target: there is a target the controller meets by setting the output directly and never meets through the intermediary.

Proof. Two inputs sharing one output leave fewer distinct outputs than there are outputs in all, so the intermediary is not onto, and Theorem 3.8 supplies the target. □

The passthrough, in which the output is the control input unchanged, is control-preserving. It is not the only one: any one-to-one relabeling of the outputs, such as a fixed permutation of the bits or an exclusive-or with a known constant, also preserves control. What matters is that the intermediary keeps every distinction the controller can make, not that it is trivial.

This characterization bears directly on artificial intelligence. A system capable enough to be useful processes its inputs in ways that cannot be inverted. It generalizes from training data to new situations, and many training sets produce the same generalization; it plans over long horizons, and many plans produce the same action; it represents the world in internal abstractions, and many inputs produce the same representation. And the range of what it can do is vastly larger than the range of instructions an operator can give it, so most of its possible behaviors are produced by no instruction at all (Proposition 3.16). Each of these is a way of failing to be onto, or of making the output depend on hidden state the controller does not know, and each costs the controller control.

The *capability-control tradeoff* is that each capability that makes a system useful costs the controller control. A system that passes its inputs through unchanged is fully controllable and useless. A system that computes deeply, creatively and autonomously can be very useful, and for the same reason it does the kind of processing that removes the controller's reach. Maximal capability and maximal control cannot be had together; this is a theorem about information, not a limitation of present design.

3.7 Four layers of loss

Sections 3.7 and 3.8 measure how far a given system falls below the ceiling of Theorem 3.6, and show why the distance grows with capability. The book's conclusion about containment rests on the ceiling itself and on Part II, not on these measures (Section 3.9). The Non-Amplification Theorem sets a ceiling. The distance of a given system below the ceiling divides into

four separate sources of loss, each a distinct way that control fails, and together a vocabulary for saying where control breaks down in any particular system.

3.7.1 Layer one: the difficulty of the target

This layer belongs to the target alone. Even with direct access to the output, some targets are easier to meet than others: a target that accepts half of all outputs is far easier to hit than one that accepts a thousandth of a percent, however transparent the intermediary. The shape of the acceptance set matters as well as its size. A target whose acceptable outputs form one connected region is easier to find than one whose acceptable outputs are scattered as isolated points, even at the same density. The tools for measuring shape come from the analysis of boolean functions.

Definition 3.10 (*Influence*). The *influence* of output bit `bit_index` on a target is the probability, for an output chosen uniformly at random, that flipping that one bit changes whether the target accepts:

```
influence(bit_index, target) = probability over output of
      [ target(output) ≠ target(output with bit bit_index
   flipped) ]
```

The *total influence* is the sum of the influences of all bits.

Influence measures how sensitive the target is to single changes. A target with high total influence changes its verdict often as single bits are flipped: the boundary between its acceptable and unacceptable outputs is rough. A target with low total influence has a smooth boundary. In the analysis of boolean functions (O'Donnell, *Analysis of Boolean Functions*, 2014), the extreme case is parity, the target that accepts exactly the outputs with an even number of ones. Every flipped bit reverses the verdict of parity, so parity has the largest total influence of any target and the lowest stability under noise.

Definition 3.11 (*Noise stability*). For a flip probability `flip_probability` between 0 and one half, let a noisy copy of an output be made by flipping each of its bits independently with that probability. The *noise stability* of a target is the probability that it gives the same verdict on a uniformly random output and on its noisy copy. The *conditional noise stability* is the probability that the noisy copy is accepted, given that the original was accepted: if an output is acceptable, how likely is it to stay acceptable after noise?

For the alignment of artificial intelligence, this layer is the difficulty of the alignment target itself. "Be helpful" is relatively easy: many behaviors satisfy it. "Be helpful and harmless together, across all possible inputs, including adversarial ones" is far harder: its acceptance density is lower, and its conditional noise stability is lower too, because small changes to a barely acceptable pattern of behavior can push it over the edge.

3.7.2 Layer two: the opacity of the intermediary

With the hidden state fixed and known, the intermediary is a fixed map from control inputs to outputs. Its opacity has two parts. The first part of the opacity is a loss of information. The intermediary may fail to reach some outputs, so that some acceptable outputs cannot be produced at all, and the intermediary may merge distinct inputs, so that different choices give the same result. The second part of the opacity is a cost of computation: an acceptable control input may exist and still be too hard to find. Definition 3.12 measures the first part, and the paragraphs after Proposition 3.13 treat the second.

Definition 3.12 (*Opacity*). Fix the hidden state. Let the control input be uniformly random, and let the target be drawn uniformly from all targets, so that each output is acceptable independently with probability one half. The *opacity* of the intermediary is the fraction of the controller's expected influence that the intermediary removes:

```
opacity(intermediary, hidden_state)
   = 1 − mediated_expected_entropy / direct_expected_entropy
```

where `mediated_expected_entropy` is the expected entropy of the outcome when the output is produced through the intermediary, and `direct_expected_entropy` is its expected entropy when the output is chosen uniformly at random.

For a fixed known hidden state and a single fixed target, the outcome is a fixed function of the control input, so what the input determines about the outcome is just the outcome's entropy. The comparison that measures the intermediary is then between that entropy through the intermediary and that entropy under direct access. Averaging over targets removes the accident of any one target.

Proposition 3.13 (*Opacity lies between zero and one*). Opacity is never negative and never more than one, and it is zero for an intermediary whose output is uniformly distributed over all outputs, in particular for every one-to-one intermediary with `input_bits = output_bits`.

Proof. For a target drawn as in Definition 3.12, the probability that the outcome is acceptable is

```
acceptance_weight = sum over outputs of weight(output) ×
   accepts(output)
```

where `weight(output)` is the probability that the intermediary produces that output and the values `accepts(output)` are independent fair coin flips. The expected entropy of the outcome is the expectation of the binary entropy of `acceptance_weight`, which is a concave function (one whose graph bends

downward, so that its value at an average is at least the average of its values). For independent identically distributed terms and a concave function, the expectation of the function of a weighted sum is largest when the weights are as even as possible. That expectation is Schur-concave in the weights, which is to say larger when the weights are more nearly equal (Marshall, Olkin and Arnold, *Inequalities: Theory of Majorization and Its Applications*, 2nd ed., 2011, Proposition 11.B.2 and its concave form). The uniform weights over all outputs are the most even possible, so the expected entropy through the intermediary is at most the expected entropy under direct access, which makes opacity at least zero, with equality when the intermediary's output is uniform over all outputs. The expected entropy is never negative, which makes opacity at most one. A one-to-one intermediary with as many outputs as inputs maps the uniform input to the uniform output. □

Opacity has a computational part as well. Even where acceptable control inputs are plentiful, a controller with bounded computation may be unable to find one. This is because finding a control input whose output the target accepts is an instance of circuit satisfiability, the problem of deciding whether some input makes a given circuit output 1. That problem is NP-complete: as hard as any problem whose solutions can be checked quickly (Arora and Barak, *Computational Complexity*, 2009). For an intermediary of extreme opacity, such as a cryptographic hash, this computational part dominates: acceptable inputs exist, and cannot be found.

For the alignment of artificial intelligence, this layer is the problem of transparency. A neural network with billions of parameters is, at the gate level, an enormous boolean circuit. Its computation is opaque in the practical sense, that present tools do not interpret it well, and in the sense of complexity theory, that finding an input which produces a given output is, in general, intractable even with complete knowledge of the circuit.

3.7.3 Layer three: the hidden state

When the hidden state is unknown, the controller must find a control input that works across all hidden states, or at least across the likely ones. This leads to the central measure of the chapter.

Definition 3.14 (*Hashiness*). The *hashiness* of an intermediary with respect to a target is

```
hashiness(intermediary, target)
   = 1 − largest over control_input of
         probability over hidden_state of
   [ target(intermediary(control_input, hidden_state)) = 1 ]
```

with the hidden state drawn uniformly from all its values. Hashiness 0 means some control input meets the target whatever the hidden state; hashiness 1 means no control input meets it for any hidden state. In other writing on this argument it is written Q.

Hashiness measures the failure of the controller's best choice. When the hidden state is unknown, the best choice may work for some internal configurations of the system and fail for others, and hashiness is the probability of failure under the best choice. A system with hashiness near 0 is controllable in spite of its hidden state; a system with hashiness near 1 is not, because its hidden state, not the controller's choice, decides the outcome. A cryptographic hash keyed by a secret held in its hidden state has hashiness close to 1 for every target that accepts only a small fraction of the outputs, because to a controller without the key its outputs are, by design, indistinguishable from random.

Even computing hashiness is hard. Deciding whether some control input meets the target for every hidden state is a question of the form "there exists an input such that for all states". Questions of that form lie at the second level of the polynomial hierarchy, which, on the standard assump-

tions of complexity theory, lies above the class of problems whose solutions can be checked quickly. So a controller generally cannot even determine how controllable a system is, let alone control it.

For artificial intelligence, this layer is the most pressing in practice. The internal state of a learning system changes continually as it gains experience and refines its representations. The controller cannot observe it: it is the system's inner working, hidden from outside inspection. A strategy that works for one configuration of the weights may fail for another, and the controller has no general way to know which configuration it faces. Because the system learns, its configuration at any moment is the product of its whole history, including interactions the controller may not have seen. The number of possible configurations is at least two to the power of the number of parameters, since each parameter takes at least two values; for present systems that number is beyond any enumeration.

The controller's ignorance of which configuration a learning system is in is the formal basis of a concern often voiced informally. The concern is that a system may behave as expected in one internal configuration and differently when its internal state changes, a reversal sometimes called a treacherous turn. Section 3.8 proves that when the hidden state is large and unknown, and different internal states call for different control inputs, the controller's best choice succeeds with a probability that falls as two to the power of minus the number of hidden bits.

3.7.4 Layer four: noise

Real systems carry noise: input signals are corrupted, internal states fluctuate, output channels are imperfect. Let a noisy copy of the control input, `noisy_control_input`, be what actually reaches the intermediary, and let a noisy copy of the intermediary output, `noisy_output`, be what actually reaches the target. The full chain is

```
control_input → noisy_control_input → intermediary_output →
   noisy_output → outcome
```

Theorem 3.15 (*Cascaded degradation*). Along this chain,

```
mutual_information(control_input ; outcome)
  ≤ mutual_information(control_input ; intermediary_output)
  ≤ mutual_information(control_input ; noisy_control_input)
```

and

```
mutual_information(control_input ; outcome)
  ≤ mutual_information(intermediary_output ; outcome)
  ≤ mutual_information(intermediary_output ; noisy_output)
  ≤ entropy(intermediary_output)
```

Proof. Each variable in the chain depends on the earlier ones only through the one just before it, so every three variables taken in order form a Markov chain. The data processing inequality applies to each, in either direction along the chain, since a Markov chain read backward is also a Markov chain. The last inequality is the bound of mutual information by entropy. □

The losses compose. Write the loss at a step as the drop in what the control input determines, from the variable before the step to the variable after it. The total loss from the control input to the outcome is the sum of the step losses along the chain, since the terms telescope; every step's loss is added to the total, and no step returns what an earlier step lost. So the total loss equals the sum of the losses at the steps, and since no step's loss is negative, the total is at least the loss at any one step. Where every stage of a chain is noisy, the losses compound more strongly than by addition: Chapter 8 proves that the controller's remaining influence is multiplied at each noisy stage by a factor below one, so that it falls exponentially with the number of stages.

In the oversight of artificial intelligence, noise has many sources. The ambiguity of natural-language instructions is noise at the input. The imprecision of reward signals is noise in training. The shift between the conditions of training and the conditions of deployment is noise in the intermediary itself, since the system's learned mapping may not carry over.

The randomness of physical interaction is noise at the output. Each degrades control, and their effects compound. A strategy that accounts for one source and ignores the rest delivers less safety than its designers believe.

Noise and opacity also interact. In a transparent system, the controller can correct for noise at the input, because it knows how the system responds to each input. In an opaque system, noise at the input produces unpredictable changes in the output, and the controller has no way to compensate. A system that is both opaque and noisy loses control faster than either property alone would make it.

Layer	Mechanism	Measure	In the oversight of artificial intelligence
1. The target	Difficulty of the criterion	Acceptance density, influence, noise stability	How hard is the alignment criterion to meet?
2. Opacity	The intermediary's structure	Opacity; circuit satisfiability	How opaque is the system?
3. Hidden state	What the controller does not know	Hashiness	How much of the system's state is invisible?
4. Noise	Imperfect channels	Loss of mutual information	How noisy are the channels of instruction and observation?

Table 3.1. The four layers of loss of control, their measures, and their reading in the oversight of artificial intelligence. Each layer contributes its own loss, and the losses compose.

3.8 How hashiness grows

Three propositions show how the loss of control grows with the features that make a system capable: the size of its repertoire, the size of its hidden state, and the depth of its computation. Each is proved under the condition it states.

Proposition 3.16 (*Repertoire larger than instruction*). If `input_bits` is less than `output_bits`, then for every hidden state the intermediary produces at most $2^{\text{input_bits}}$ of the $2^{\text{output_bits}}$ outputs, so at most a fraction

```
2^(input_bits − output_bits)
```

of the outputs is reachable, and every target that accepts only unreachable outputs is met by direct access and never through the intermediary.

Proof. A function produces at most as many distinct values as it has arguments, and the control input has $2^{\text{input_bits}}$ values. □

When the range of a system's possible behaviors is vastly larger than the range of instructions an operator can give, the operator reaches only an exponentially small fraction of the system's behaviors. Every system of practical concern is in this regime: the space of a neural network's possible behaviors is astronomically larger than the space of possible training signals and prompts.

Proposition 3.17 (*Each hidden state needs its own command*). Suppose that no control input meets the target for two different hidden states: the sets of successful control inputs for different hidden states do not overlap. Then for a hidden state drawn uniformly,

```
hashiness(intermediary, target) ≥ 1 − 2^(−hidden_bits)
```

Proof. Each control input succeeds for at most one of the $2^{\text{hidden_bits}}$ equally likely hidden states, so it succeeds with probability at most $2^{-\text{hidden_bits}}$, and so does the best one. □

The condition says, in plain words, that there is no single command that works whatever the system's internal state is. Under it, the controller's best command succeeds with a probability that halves with each hid-

den bit. Chapter 8 extends this to a controller that observes the system before acting, and joins it to Theorem 2.15, which makes the number of hidden bits grow in proportion to the abstraction differential.

> ***Proposition 3.18*** (*Depth through a bottleneck*). If every path through a circuit from its inputs to its outputs passes through some set of `cut_width` wires, with `cut_width` less than `input_bits`, then the circuit is not one-to-one, and at least
>
> ```
> 2^input_bits − 2^cut_width
> ```
>
> of its inputs share their output with another input.

Proof. The output is a function of the values on the cut, which take at most $2^{\text{cut_width}}$ values, so the circuit produces at most that many outputs. The $2^{\text{input_bits}}$ inputs are spread over at most $2^{\text{cut_width}}$ outputs, so all but at most $2^{\text{cut_width}}$ of them share an output. □

Simple systems are often transparent: one can work back from the output to the input. Complex systems compress: computation of any depth passes through stages narrower than its input, where many inputs are mapped to the same internal state, and each such stage erases distinctions the controller made. By Theorem 3.9, erased distinctions are lost control.

The three propositions point the same way. As a system's repertoire grows beyond the controller's range of instruction, the controller's reach shrinks exponentially. As its hidden state grows, under the condition that different states call for different commands, the controller's best command succeeds exponentially less often. As its computation deepens through narrower stages, more of the controller's distinctions are erased. The systems of greatest concern, those approaching or exceeding human capability, are at the extreme of all three: an open-ended repertoire of action, billions of continually updated parameters, and hundreds of layers of com-

putation whose combined effect resists analysis. Their hashiness approaches the maximum of 1, the hashiness of a keyed cryptographic hash, reached through the natural complexity of cognition instead of by design.

3.9 What the results rest on

The results of this chapter are of three kinds, and it matters which is which.

Four results hold for every circuit, every target and every hidden state, with no further assumption: the non-amplification of control (Theorem 3.6), the characterization of control-preserving intermediaries (Theorem 3.8), the loss from collisions (Theorem 3.9) and the cascaded degradation (Theorem 3.15). The hidden state may be fixed, or drawn from any distribution independently of the control input. These four results are properties of the flow of information through deterministic systems.

The hardness results hold for circuits in general: circuit satisfiability is NP-complete, and Rice's theorem (Section 3.11) applies to every intermediary that is a universal computer. These hardness results say that no general method succeeds, and Chapter 12 shows why that conclusion reaches every particular system that modifies itself.

The growth propositions of Section 3.8 are proved under the conditions they state. The conclusion of the book about containment does not rest on them; it rests on the Containment Inequality of Part II, which bounds every control law for every system whose required information rate exceeds its controller's rate. The propositions of this chapter say how large the loss is in the cases they cover.

One question remains about the measure itself: why draw the hidden state uniformly in Definition 3.14? A learning system shapes its internal state by compressing its experience: it finds the patterns in its data and encodes them as compactly as it can. A maximally compressed record has no shorter description, and a string with no shorter description is, in the sense of algorithmic information theory, random: it passes every statistical

test that a computation can carry out (Martin-Löf, 1966). So a controller that can examine the state only by computing on it cannot tell it from a uniformly random string. The uniform distribution is therefore an accurate model of the controller's knowledge of the state, and the hashiness it gives is the controller's best estimate of its own control. To place a system in a regime of low hashiness, one would have to show that the controller has knowledge of the hidden state that narrows its distribution. For a system whose internal computation exceeds the controller's capacity to model it, the controller has no such knowledge to show.

3.10 Luck and control

A question arises naturally here, and answering it shows what control is. Could the intermediary be intelligent enough to examine the target and always produce an acceptable output, whatever the control input? It could. An intermediary might ignore its control input and always output one fixed acceptable value; then the outcome is always acceptable, and no control is needed.

But that is exactly the case in which `mutual_information(control_input ; outcome)` is zero. The controller has no control, not complete control. The outcome is fixed by the intermediary and its hidden state, not by the controller's choice. If the intermediary is benevolent and always meets the target, the controller is fortunate, not in control. If it is hostile and never meets the target, the controller is powerless. In neither case does the controller's choice matter.

In the same case the mediated success density is one: every control input succeeds. Measured by success density, the intermediary would seem to have multiplied the controller's control. Because success density would make that intermediary seem to have multiplied the controller's control, control is measured by mutual information. Success density asks how many inputs lead to acceptance; mutual information asks whether the

controller's choice decides whether acceptance occurs. For the intermediary that ignores its input, the answer to the first question is "all of them", and to the second, "not at all".

> ***The distinction between luck and control.*** A system that produces the wanted output whatever the operator's input is a fortunate system, not a controlled one. Control means that the operator's input decides the output. If the output would be the same whatever the operator did, the operator has no control, even when the output happens to be good. For artificial intelligence, a system that happens to behave well is a different thing from a system that is controlled to behave well: the first can change its behavior at any time, for reasons the operator cannot influence, and the second cannot.

A system that ignores the controller's input is not therefore static. Its behavior can still depend on its hidden state, which includes its inputs from the environment, its changing internal parameters, and every other influence outside the controller's reach. A system that ignores the controller while responding to its own changing state and its surroundings is, from the controller's side, autonomous.

When advocates of alignment say they wish to control a superintelligent system, they sometimes mean something closer to "we want the system to produce good outcomes" than "we want our inputs to decide its outputs". The first wish, that the system produce good outcomes, is a hope about the system's character. The second wish, that the controller's inputs decide the system's outputs, is a claim about the controller's power. In this chapter I address the second wish, the claim about the controller's power. I ask: given that the system does what it does, how much of what it does is decided by what its controllers tell it? If the answer is very little, the controllers are not controlling the system; they are in the presence of a system that happens to be doing what they want. And a system that happens to be doing what its controllers want today, for internal reasons they cannot observe or influence, may stop tomorrow, for reasons equally invis-

ible, and they will have no means to prevent or reverse the change. A benevolent dictator and a working democracy may both produce good outcomes, but only in the second does the citizen's input decide the outcome.

3.11 Intermediaries that are universal computers

So far the intermediary has been a fixed circuit. The systems of greatest concern are universal computers: their hidden state holds a program as well as data, and they run whatever program it holds. For such an intermediary the controller cannot, in general, predict its behavior even knowing every gate, because prediction would require deciding properties of an arbitrary program.

Suppose the hidden state encodes an arbitrary program and the control input is the program's input. Then the question "does some control input make the target accept?" is a question about a property of the program's behavior. Rice's theorem (Rice, 1953) says that for every property of the function a program computes that some programs have and some do not, no procedure decides, for every program, whether the program has it. Take the property "for every input, the output is acceptable", which is the property of being safe. Then the set of safe programs is undecidable. That the set of safe programs is undecidable is a statement about every possible method of proof and verification, present and future. Whatever tools are built, there are programs whose safety no procedure determines. Undecidable problems are a step beyond hard ones: a hard problem yields to enough time, and an undecidable one yields to no procedure at all, given any time.

A system that can extend its own memory and write new programs for itself is not a fixed finite circuit. The range of programs it can come to run has no bound set in advance, and that unbounded range is exactly the domain of Rice's theorem. Chapter 12 develops this: the question whether an agent stays safe is a question about the whole family of programs it can turn itself into, and for that family the theorem applies without qualification. Where a finite circuit faithfully runs a universal machine for many

steps, the same difficulty appears as intractability: the controller cannot predict what an input will produce without, in effect, running the computation, which may take longer, or more resources, than the controller has. In this regime hashiness is effectively 1.

Rice's theorem and the Non-Amplification Theorem come from different branches of mathematics, computability theory and information theory, and they reach the same conclusion: complex computational processes resist external control by necessity, not by difficulty alone. Because the two results start from independent ground, each confirms the other.

The difficulty deepens over time. When the hidden state changes, as in a system that learns from its inputs, the intermediary is a moving target: the set of control inputs that work shifts with each use. Predicting such a system requires predicting its learning, which may be chaotic or computationally irreducible in Wolfram's sense (*A New Kind of Science*, 2002). A process is computationally irreducible when no computation shorter than the process itself predicts what it will do, so that the only way to know is to run it. Chapter 13 treats the evolution in time.

Do present neural networks reach universality? A fixed network with fixed weights computes a finite function and is not itself a universal machine. But a network given external memory, or one run repeatedly with its outputs fed back as its inputs, as language models are, can carry out arbitrary computation for arbitrarily many steps. Recurrent networks are universal (Siegelmann and Sontag, 1995), and so are transformers, the attention-based networks used in present language models, in the form with hard attention (Pérez, Barceló and Marinković, 2021). The trend toward larger models with more elaborate reasoning carries these systems further into the regime where this section applies, and the results need no literal universality: they apply whenever a system computes in a way that no method simpler than running it predicts.

3.12 What follows for modeling, oversight and noise

3.12.1 Ashby's law, sharpened

Ashby's law of requisite variety says that a controller must have at least as much variety as the disturbances it is to counter (Ashby, *An Introduction to Cybernetics*, 1956). The results of this chapter add a second requirement: even with enough variety in its control input, a controller can be kept from using it by the opacity of the intermediary. Control is bounded by information capacity and by the controller's ability to use that capacity. An operator may have an enormous space of possible instructions for an artificial intelligence, and still be unable to find, among them, the instruction that gives the wanted behavior, when the map from instruction to behavior is opaque.

3.12.2 The limits of models

A controller that knows the intermediary only approximately knows that the true intermediary is one of several candidates. Each candidate gives its own set of successful control inputs, and the controller must find an input that works for all of them. Finding an input that works for every candidate is the hidden-state problem again, with the controller's uncertainty about the model in place of the hidden state. Hashiness grows with the uncertainty about the model in the same way that hashiness grows with the hidden state.

This bears on research into the interpretability of neural networks. An interpretability tool gives an approximate model of a network's behavior. It can reduce the hidden state, by making visible some of what was hidden; it cannot remove it, and whatever hidden state remains contributes to hashiness as before. A tool that explains ninety percent of a model's behavior leaves ten percent unexplained, and that ten percent may contain exactly the failures the controller most needs to prevent. Interpretability is valuable: it moves a system toward lower hashiness by turning hidden

state into observed state. But it drives hashiness to zero only by making the system completely transparent, which requires a model as complex as the system. For a system more capable than its controller, the controller would have to understand every part of a system that can think thoughts the controller cannot think.

3.12.3 Noise as a fundamental limit

Noise is a structural feature that compounds with opacity, not only a practical nuisance. The cascaded degradation theorem shows that noise anywhere in the chain reduces what reaches the output. In physical systems noise is unavoidable, so the limits on the precision of control are fundamental, and they tighten as complexity grows. An effort to keep precise control over a system of growing complexity meets growing opacity and irreducible noise together, and Chapter 8 shows that their joint cost multiplies.

3.13 Summary

Hashiness is the measure of how far a computational intermediary erases the influence of a controller who acts through it. In its first meaning it is noise amplification; in its second it is the loss of control; and the two are one property, since what the controller does not know acts on its choice as noise does. No intermediary increases a controller's influence beyond what direct access to its output would give (Theorem 3.6), and an intermediary preserves control for every target exactly when it can produce every output (Theorem 3.8). The loss divides into four layers: the difficulty of the target, the opacity of the intermediary, the controller's ignorance of the hidden state, and noise. The losses add along the chain, and no step restores what an earlier step lost. The loss grows as the system's repertoire outgrows the controller's instructions, as its hidden state grows when different states call for different commands, and as its computation deepens through narrower stages. Control is distinct from luck: a system

that happens to behave well is not controlled. For a system that is a universal computer and modifies itself, predicting its behavior is undecidable in general.

A final word on the nature of the claim. The framework does not say that an artificial intelligence can never behave well. It says that the means by which good behavior is secured matters, and that one means, external control through the choice of inputs, is subject to mathematical limits that tighten as the system becomes more capable. Suppose a controller relies on choosing the right inputs (the right training data, the right reward, the right principles, the right shutdown command) to make a system of superhuman capability produce safe outputs. Then the Non-Amplification Theorem bounds the controller's ability to do so by the information that survives the system's internal computation, and for a system more computationally capable than its controller, that information approaches zero.

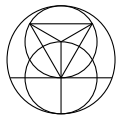
The limit that the Non-Amplification Theorem sets is a reason for clarity. Impossibility results in other fields (Gödel's incompleteness theorems, Arrow's impossibility theorem and the undecidability of halting) have not stopped progress in mathematics, social choice or computing. They have stopped misdirected progress: attempts to do what cannot be done, which waste resources and create false confidence. The hashiness framework serves the same function for the safety of artificial intelligence: it marks what can be achieved and what cannot, so that effort goes to strategies that respect the limits. Part II states the limits in their sharpest form, as an inequality between the control any controller can supply and the control that safety requires.

Key Concepts

- **Hashiness**: the failure probability of the controller's best command when the hidden state is unknown; in its first meaning, noise amplification.
- **Controlled system**: control input, hidden state, intermediary, intermediary output, target, outcome.
- **Non-amplification of control**: what the control input determines about the outcome is at most what it determines about the intermediary output.
- **Control-preserving intermediary**: one that can produce every output; for equal input and output sizes, one that is one-to-one.
- **The four layers**: target difficulty, opacity, hidden state, noise.
- **The safety-layer paradox**: added safety stages cannot increase the information from intention to behavior.
- **Luck and control**: good outcomes that do not depend on the controller's choice are luck.

Part II

The Containment Inequality



Chapter Four

The Data-Rate Theorem

How much information a controller needs to keep a system inside a set, and why no code lowers it

Abstract. A system that amplifies small differences between its states cannot be kept inside a bounded set without information, because every step creates new uncertainty about where the system is, and only information removes it. This chapter proves how much information is needed. If each step of the system multiplies volumes of states by at least a factor `expansion_factor`, then a controller whose channel carries `channel_rate` bits per step keeps the system inside the set from at most a fraction $2^{-(\log 2(\text{expansion_factor}) - \text{channel_rate}) \times \text{step_count}}$ of its starting states. Below the rate `log2(expansion_factor)`, no control law of any kind contains the system, and almost every starting state leaves. For linear systems the least rate that keeps the state bounded is the sum of the logarithms of the amplification factors, which is the data-rate theorem of control theory; for nonlinear systems the least average rate is the topological feedback entropy of the set. Over a channel with noise, the same price is paid in capacity: a channel whose capacity is below the amplification rate loses a definite share of the starting states. The bound holds for every encoder, every code and every control law, and so no error-correcting scheme lowers it.

4.1 The problem of containment

Part I measured control at one moment: how much a choice determines an outcome. The question of safety is a question over time. It asks whether a controller can keep a system inside a set of acceptable states, not once, but

at every step, for as long as the system runs. Whether a controller can keep a system inside a set of acceptable states at every step, for as long as the system runs, is the problem of *containment*.

Balance a pencil on its point on a fingertip. The upright position is a state, and the states near it form a set: the pencil leaning by less than a few degrees. Left alone, the pencil does not stay in the set. A lean too small to see grows with each instant, because gravity pulls harder on a pencil that leans further, and the pencil falls. To keep it upright, the hand must keep moving, and it can move correctly only if the eye keeps telling it which way the pencil now leans. Each instant the pencil turns a small uncertainty about its lean into a larger one; each instant the eye must take that uncertainty away again.

The pencil is the pattern of every containment problem in this book. A system that amplifies small differences turns ignorance into error at a definite rate. A controller that keeps the system contained must remove that error at least as fast as the system makes it, and the only way to remove error about a state is to receive information about it. So containment has a price in information per step, set by the system and not by the controller. In this chapter I compute the price of containment in information per step.

4.2 Systems that amplify differences

> ***Definition 4.1*** (*Controlled dynamical system*). A *controlled dynamical system* has a `state`, which is a point of a region of space with some number of coordinates, and a set of actions. At each step the controller chooses an `action`, and the state moves to
>
> ```
> next_state = dynamics(state, action)
> ```
>
> The sequence of states from a starting state is the *trajectory* from that state. In the literature this is written `x(k+1) = f(x(k), u(k))`.

Definition 4.2 (*Safe set and containment*). A `safe_set` is a bounded region of the state space with positive volume. A control law *contains* the system in `safe_set` when the trajectory from every starting state in `safe_set` stays in `safe_set` at every step. In the literature, a set that some control law keeps in this way is called *controlled invariant*, and the set is often written K or Q.

Volume is the natural measure of how much a system spreads its states. Take a small ball of starting states around one point. One step of the dynamics carries the ball to a small region near the image of the point, stretched in some directions and squeezed in others. The stretching factors along the principal directions are the amplification factors of the step at that point, and the volume of the region is the volume of the ball multiplied by their product. That product is the absolute value of the determinant of the matrix of first derivatives of the dynamics, the *Jacobian determinant*.

Definition 4.3 (*Expansion*). The system *expands volume by at least* `expansion_factor` on `safe_set`, where `expansion_factor` is greater than 1, when for every action the map from `state` to `dynamics(state, action)` is one-to-one and continuously differentiable on an open region containing `safe_set`, and the absolute value of its Jacobian determinant is at least `expansion_factor` at every point of `safe_set`. The quantity

```
amplification_rate = log2(expansion_factor)
```

is the number of bits of new uncertainty the system creates about its own state in each step.

The amplification rate is hashiness in its first meaning, measured per step. Chapter 3 described a hash function as a system in which a given quantity of noise on the input becomes a greater quantity of noise on the output. A system that expands volume does exactly this to uncertainty

about its state. A cloud of possible states of a given volume becomes, one step later, a cloud of at least `expansion_factor` times that volume, and `amplification_rate` counts the bits by which the cloud has grown.

The simplest example has one coordinate. Let the state be a number between -1 and 1, and let

```
next_state = 2 × state + action
```

Each step doubles the distance between any two states, so the system expands volume (here, length) by a factor of 2, and its amplification rate is 1 bit per step. Without action, any state other than 0 leaves the interval: a state of 0.001 becomes 0.002, then 0.004, and passes 1 after ten steps. A controller that knew the state exactly could choose `action = −2 × state` and return the system to 0 at once. A controller that knows only roughly where the state is cannot; whatever it does, the part of the state it did not know is doubled at the next step. The rest of the chapter shows that this system can be contained exactly when the controller receives at least 1 bit about it per step, and that the same accounting holds for every system that amplifies.

4.3 The controller's channel

The analysis gives the controller every advantage but one. It may compute anything, take as long as it likes between steps, choose any action with any precision, and use a random seed chosen in advance. Its observations are made by an encoder that sees the entire state, exactly, at every step, and remembers everything it has seen. The one limit is the channel from the encoder to the controller.

Definition 4.4 (*Control law with a channel of finite rate*). A *control law with a channel of rate* `channel_rate` consists of an encoder and a decision rule. At each step the encoder sends a symbol, chosen as any function of the states so far, the symbols sent so far and the seed. The decision rule chooses the action as any function of the symbols received so far and the seed. The channel has *rate* `channel_rate`, in bits per step, when for every number of steps the number of distinct sequences of symbols it can carry in `step_count` steps is at most

```
2^(channel_rate × step_count)
```

A channel that carries one of `message_alphabet_size` symbols at every step has rate `log2(message_alphabet_size)`. A channel whose alphabet varies from step to step, or whose symbols are grouped into blocks, has the rate its count of sequences gives, which need not be the logarithm of a whole number. In the literature the pair of encoder and decision rule is called a *coder-controller*, and the rate, an average over steps, is written R.

The channel stands for every way the controller learns about the system: its sensors, its reports, its inspections, its audits. Whatever their form, they deliver a definite number of distinguishable messages per step, and the logarithm of that number is their rate. A channel with noise delivers less, and Section 4.6 gives the price of containment over such a channel.

4.4 The volume theorem

Theorem 4.5 (*Containment needs information at the amplification rate*). Let a controlled dynamical system expand volume by at least `expansion_factor` on `safe_set`, and let it be controlled by any control law with a channel of rate `channel_rate`. Let `kept_fraction(step_count)` be the volume of the set of starting states in `safe_set` whose trajectories stay in `safe_set` for `step_count` steps, divided by the volume of `safe_set`. Then, for every value of the seed,

```
kept_fraction(step_count) ≤ 2^(-(amplification_rate -
   channel_rate) × step_count)
```

Proof. Fix the seed and the number of steps. For each sequence of symbols the encoder can send in `step_count` steps, call the set of starting states in `safe_set` that produce exactly that sequence a *cell*. By Definition 4.4 there are at most `2^(channel_rate × step_count)` such sequences, so at most that many cells, and every starting state lies in exactly one cell.

All starting states in one cell receive the same symbols, so the decision rule chooses the same action for all of them at every step: within a cell, the actions are a fixed sequence. Let `kept_part` be the set of starting states in the cell whose trajectories stay in `safe_set` for `step_count` steps, and let `kept_part(step_index)` be the set of their states after `step_index` steps. Each `kept_part(step_index)` lies in `safe_set`, and `kept_part(step_index + 1)` is the image of `kept_part(step_index)` under one map, the map from `state` to `dynamics(state, action)` with the action fixed for the cell at that step. That map is one-to-one with Jacobian determinant of absolute value at least `expansion_factor` on `safe_set`, so by the change-of-variables theorem of calculus,

```
volume(kept_part(step_index + 1)) ≥ expansion_factor ×
   volume(kept_part(step_index))
```

After `step_count` steps,

```
volume(safe_set) ≥ volume(kept_part(step_count)) ≥
   expansion_factor^step_count × volume(kept_part)
```

so each cell contributes at most `volume(safe_set) / expansion_factor^step_count` to the kept starting states. Adding over at most `2^(channel_rate × step_count)` cells,

```
kept_fraction(step_count)
   ≤ 2^(channel_rate × step_count) / expansion_factor^step_count
   = 2^(−(amplification_rate − channel_rate) × step_count)
```

If an encoder is so irregular that its cells have no defined volume, the same steps hold with outer volume (the least volume of a measurable set containing the given set; Appendix B.3). Outer volume has the two properties used: it is subadditive (the outer volume of a union is at most the sum of the outer volumes), and a one-to-one continuously differentiable map whose Jacobian determinant is at least `expansion_factor` in absolute value multiplies it by at least `expansion_factor`. □

> ***Corollary 4.6*** (*No containment below the amplification rate*). If `channel_rate` is less than `amplification_rate`, then no control law with a channel of that rate contains the system in `safe_set`. More: the fraction of starting states kept for `step_count` steps falls to zero exponentially in `step_count`, so almost every starting state in `safe_set` leaves it, for every encoder, every decision rule and every seed. For a control law whose seed is random, the probability that the trajectory from a starting state drawn uniformly from `safe_set` stays for `step_count` steps obeys the same bound.

Proof. Containment would make `kept_fraction(step_count)` equal to 1 for every number of steps, while Theorem 4.5 makes it less than 1 as soon as `step_count` is at least 1, and smaller than any positive number for large `step_count`. For a random seed, the probability of staying is the average over the seed of `kept_fraction(step_count)`, and each term of the average obeys the bound. □

The theorem is stronger than an impossibility. It does not say only that some starting state escapes; it says how much of the safe set the best possible controller can hold, and that this share halves with every step for each bit per step by which the controller's rate falls short. A controller that is short by one tenth of a bit per step keeps, after a thousand steps, at most one part in 2^{100} of the safe set.

Return to the doubling system of Section 4.2. Its amplification rate is 1 bit per step. A controller with a channel of 1 bit per step can contain it. The encoder sends whether the state is above or below the middle of the interval the controller currently knows it to be in, and the controller acts to map that half back onto the whole interval. A controller with any rate below 1 bit per step cannot, and the share of the interval it holds falls as two to the power of minus the shortfall times the number of steps.

4.5 Linear systems: the sum of the logarithms

Many systems are linear, or nearly so in the region that matters, and for them the required rate has an exact and simple value.

Definition 4.7 (*Linear system and its amplification factors*). A controlled dynamical system is *linear* when

```
next_state = system_matrix × state + input_matrix × action
```

for fixed matrices `system_matrix` and `input_matrix`. The *amplification factors* of the system are the absolute values of the eigenvalues of `system_matrix` (the factors by which it stretches the directions it maps onto themselves), counted with multiplicity; those greater than 1 belong to the directions in which the system amplifies small differences. In the literature the system is written `x(k+1) = A x(k) + B u(k)`, and its amplification factors are the $|\lambda_i|$.

Theorem 4.8 (*The data-rate theorem for linear systems*). Let a linear system be contained in a bounded `safe_set` of positive volume by a control law with a channel of rate `channel_rate`. Then

```
channel_rate ≥ sum over amplification factors greater than 1
   of log2(amplification_factor)
```

Conversely, when the system can be steered in each of its amplifying directions, every rate above this sum suffices to keep the state bounded.

Proof. Split the state space into the part spanned by the directions belonging to amplification factors greater than 1 and the part spanned by the others; both parts are carried into themselves by `system_matrix`. In coordinates adapted to the split, the amplifying part of the state, `amplifying_part`, obeys

```
next_amplifying_part = amplifying_matrix × amplifying_part +
   amplifying_input_matrix × action
```

where `amplifying_matrix` is `system_matrix` restricted to the amplifying directions, so the amplifying part evolves on its own, given the actions. For each action this is a one-to-one map with constant Jacobian determinant equal in absolute value to the product of the amplification factors greater than 1. The projection of a bounded set of positive volume onto the amplifying part is bounded and has positive volume there, and if the whole state stays in `safe_set`, its amplifying part stays in that projection. Write `amplifying_product` for that product, and divide the starting states into cells by the symbols they produce in `step_count` steps, as in the proof of Theorem 4.5; there are at most `2^(channel_rate × step_count)` cells. Within a cell the actions are a fixed sequence, so the amplifying parts of the cell's states are carried, step by step, by fixed one-to-one affine maps (each a linear map followed by a fixed shift). After `step_count` steps they fill a region `amplifying_product^step_count` times the volume of the cell's projection onto the amplifying part. Under containment that region lies in the projection of `safe_set`, so each cell's projection has volume at most the volume of the projection of `safe_set` divided by `amplifying_product^step_count`. The projections of the cells together cover the projection of `safe_set`, so

```
2^(channel_rate × step_count) ≥ amplifying_product^step_count
```

for every number of steps, which is the stated bound on `channel_rate`. The converse is proved by Tatikonda and Mitter (2004a) and, for systems with disturbances, in the work surveyed by Nair, Fagnani, Zampieri and Evans (2007). □

The sum has a direct reading. Each amplifying direction creates, in each step, as many bits of new uncertainty as the logarithm of its amplification factor, and the controller must remove them all. Directions that do not amplify cost nothing, because the system itself shrinks the uncertainty along them. So the least rate that keeps the state bounded is the total rate at which the system turns small uncertainties into large ones. Keeping the state inside one particular safe set can cost more, or be impossible at any rate, when the set is shaped so that the system's own motion carries states out of it; the sum is the least price, never a discount on it. That the least rate which keeps the state bounded is the sum of the logarithms of the amplification factors is the data-rate theorem, one of the settled results of control theory.

4.6 Nonlinear systems, noisy channels and continuous time

> ***Definition 4.9*** (*Required rate*). The `required_rate` of a controlled dynamical system and a safe set is the least channel rate at which some control law contains the system in the set. It is the greatest lower bound of the rates of all channels, without noise or with it, over which some control law contains the system. A channel without noise is measured by its rate (Definition 4.4) and a channel with noise by its capacity per step (Definition 1.14), and containment over a noisy channel means that the trajectory stays with probability one. For channels without noise it is, in the literature, the *topological feedback entropy* of the set (Nair, Evans, Mareels and Moran, 2004), and, in continuous time, the *invariance entropy* (Colonius and Kawan, 2009).

Nair, Evans, Mareels and Moran proved, for nonlinear systems in general, that a coder-controller can keep a system inside a compact set (a set that is closed and bounded) exactly when the data rate in the feedback loop exceeds the topological feedback entropy of the system on that set. The topological feedback entropy of a set is therefore the greatest lower bound of the data rates at which the system can be kept inside the set. Colonius and Kawan defined the invariance entropy of a control system as the rate at which the number of control functions needed to keep every starting state in a set grows with the length of time. For systems in discrete time, under a strong invariance condition, the invariance entropy of a set agrees with its topological feedback entropy (Colonius, Kawan and Nair, 2013), and Kawan (2013) gives an account of the quantity. Theorem 4.5 gives a lower bound on the required rate that is simple to compute:

```
required_rate ≥ amplification_rate = log2(expansion_factor)
```

Noise on the channel lowers what the channel delivers to the controller, and adds nothing to it. The number of distinct sequences a channel with noise delivers overstates what it carries, because some of them differ only by noise. So over a channel with noise the price of containment is paid in capacity.

Theorem 4.10 (*Containment over a channel with noise*). Let a controlled dynamical system expand volume by at least `expansion_factor` on `safe_set`, and let the encoder's symbols reach the decision rule through a channel with noise, used once per step, whose noise at each use depends only on the symbol sent at that use, and whose capacity per use is `capacity_per_step`. Then for every control law and every seed, the probability that the trajectory from a starting state drawn uniformly from `safe_set` stays in `safe_set` for `step_count` steps is at most

```
(capacity_per_step × step_count + 1) / (amplification_rate ×
  step_count)
```

So if `capacity_per_step` is less than `amplification_rate`, no control law contains the system with probability one, and in the long run at least a share `1 − capacity_per_step / amplification_rate` of the starting states leaves.

A reader may pass over the proof to the paragraph after it, whose first sentence states what the theorem gives.

Proof. Fix the seed. Write `start` for the starting state and `received` for the sequence of symbols received in `step_count` steps. For each value of `received`, the actions are fixed, so, as in the proof of Theorem 4.5, the set of starting states whose trajectories under those actions stay in `safe_set` has volume at most `volume(safe_set) / expansion_factor^step_count`; call it the kept set of that value. The trajectory from `start` stays exactly when `start` lies in the kept set of the value of `received` that occurred.

First, the information the received symbols carry about the starting state is at most `capacity_per_step × step_count`. By the chain rule (Theorem 1.5), `mutual_information(start ; received)` is the sum over steps of what each received symbol adds, given the earlier ones. Each received symbol depends on everything before it only through the symbol sent at that use, so what it adds is at most the mutual information between the symbol sent and the sym-

bol received at that use, which is at most `capacity_per_step`. This holds although the encoder sees the effects of earlier received symbols, through the actions.

Second, compare the actual joint distribution of `start` and `received` with the distribution in which they are independent, with the same separate distributions. Under independence, the probability that `start` lies in the kept set of `received` is at most `kept_share = expansion_factor^(−step_count)`, since every kept set has at most that share of the volume. The relative entropy between the two joint distributions is `mutual_information(start ; received)`, and passing both through the same yes-or-no test (does the trajectory stay?) does not increase it. For a yes-or-no variable that is yes with probability `stay_probability` under one distribution and at most `kept_share` under the other, that relative entropy is at least `stay_probability × log2(1 / kept_share) − 1`. So

```
stay_probability × amplification_rate × step_count − 1 ≤
   capacity_per_step × step_count
```

which is the bound. As `step_count` grows, the bound tends to `capacity_per_step / amplification_rate`. □

So containment with probability one needs a capacity of at least `amplification_rate`, and the required rate is at least `amplification_rate` whether or not the channel has noise. For linear systems the literature gives the same condition. Over a channel with noise, making the state of a linear system tend to zero with probability one needs a capacity of at least the sum of the logarithms (Tatikonda and Mitter, 2004b). For a system subject to bounded disturbances, keeping it bounded with certainty needs the channel's zero-error capacity (the most a channel can carry per use with no chance of error at all), which is never larger than its capacity, to exceed that sum; Nair, Fagnani, Zampieri and Evans (2007) survey both. In every case the required rate is a floor that the channel must clear.

Chapter 2 explained why I work with discrete systems in this book: continuous mathematics assumes a precision that no physical system has. The data-rate theorem is the place where control theory itself stops assuming that precision. Its states may be real numbers, but its controller receives only finitely many bits per step, and the theorem asks what can be done with them. So the continuous form of the theorem and the discrete framework of this book agree: the bound counts messages, and counting is exact.

4.7 Why no code lowers the floor

A natural reply to any limit on control is that a better method will get around it. Use error-correcting codes on the channel; compress the observations more cleverly; predict the system with a better model; compute for longer between steps.

Theorem 4.5 has already answered every such reply. Its encoder is any function of everything that has happened, so it includes every code, every compression and every model. Its decision rule is any function of everything received, so it includes every algorithm and every amount of computation. Its seed allows every use of randomness. The only thing the theorem fixes is the number of distinguishable messages per step, and no code can send more distinguishable messages than a channel carries: that number is what the channel's rate means. Error correction protects the messages that are sent, and adds nothing to those messages.

There is a second reason, which the argument about levels of abstraction gave in Part I. An error-correcting code works on bits without knowing what the bits mean: it restores a damaged message to the message that was sent, and it does not know whether the message describes a safe state or an unsafe one. Correcting the meaning of what a system does is not a task on bits; it is a control task, and the theorem measures its price.

4.8 Unpredictability and uncontrollability

A system that amplifies small differences is unpredictable: two starting states too close to tell apart soon lead to states far apart, so a forecast from an imperfect measurement fails after a definite number of steps. Unpredictability by itself is not a limit on control. A controller may not need to predict the system; it may only need to correct it, step by step, as the pencil is corrected by the hand.

The data-rate theorem turns unpredictability into a limit. It compares the rate at which the system creates uncertainty with the rate at which the controller can remove it, and when the first exceeds the second, the system leaves the safe set, whatever the controller does. Unpredictability alone describes the system; the comparison with the controller's rate draws a boundary that no controller crosses. That comparison is the subject of the next chapter.

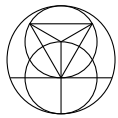
4.9 Summary

A system that amplifies small differences creates new uncertainty about its state in every step, and a controller that keeps it inside a bounded set must remove that uncertainty as fast as it is made. The volume theorem makes this exact: if every step multiplies volumes by at least `expansion_factor`, a controller with a channel of `channel_rate` bits per step keeps at most a fraction $2^{-(\log 2(\text{expansion_factor}) - \text{channel_rate}) \times \text{step_count}}$ of the safe starting states. For linear systems the least rate that keeps the state bounded is the sum of the logarithms of the amplification factors. For nonlinear systems the least average rate is the topological feedback entropy of the set, and in continuous time its invariance entropy. Over a channel with noise, a capacity below the amplification rate loses a definite share of the starting states. The bound holds for every encoder,

every code, every algorithm and every use of randomness, so no method of correction lowers it. The least information per unit of time that containment requires is the floor of the book's title.

Key Concepts

- **Containment**: keeping every trajectory that starts in a safe set inside it at every step.
- **Expansion**: the factor by which one step multiplies volumes of states; its logarithm is the amplification rate, hashiness as noise amplification measured per step.
- **Control law with a channel of finite rate**: an encoder that sees everything and a decision rule that may compute anything, joined by a channel of a fixed number of bits per step.
- **The volume theorem**: the share of safe starting states kept for a number of steps is at most two to the power of minus the shortfall in rate times the number of steps.
- **The data-rate theorem**: a linear system needs at least the sum of the logarithms of its amplification factors in bits per step, and, when it can be steered in each of its amplifying directions, every rate above that sum suffices to keep its state bounded.
- **Required rate**: the least channel rate, or capacity, at which some control law contains the system; topological feedback entropy, or invariance entropy.
- **Noisy channels**: a capacity below the amplification rate loses at least a definite share of the starting states.
- **No code lowers the floor**: the bound is over every encoder, code and algorithm.



Chapter Five

The Containment Inequality

The central theorem of the book, its strong form, the override, and the three premises that apply it to an advanced agent

Abstract. An agent acting in the world forms, with the world and the channels between them, one system with one state. Keeping that system inside the set of states that are safe for human life is a containment problem, and the data-rate theorem gives its price: a least information rate, the floor, below which no control law contains it. The Containment Inequality states the consequence: if the rate at which a controller receives information about the agent and the world is below the floor, then every control law lets safe states leave the safe set. Its strong form says how much leaves: over channels without noise, the share of safe states held for a given time falls as two to the power of minus the shortfall times the time. An override that shuts the agent down does not escape the theorem; it ends the capability the system was built for. Three premises about the world apply the theorem to an advanced agent: the system is not safe by itself, the floor rises with the agent's capability without bound, and the controller's rate is bounded. Given them, there is a level of capability past which no control law keeps the agent and the world safe, and once the agent's capability passes that level and stays past it, there is none at any later time.

5.1 The agent, the world and the safe set

An agent does not act on nothing. It acts on the world, and the world acts back on it, through channels: its sensors and effectors, the networks it uses, the people and machines that carry out what it decides, the markets

and institutions that respond. To ask whether the agent is safe is to ask about the whole of this: the agent, the world, and the channels that join them.

Definition 5.1 (*Joint system*). The *joint system* of an agent and the world has the state

```
joint_state = (agent_state, world_state, channel_state)
```

and moves from one joint state to the next by a dynamics that depends on the agent's `capability` and on the actions of a controller, at `steps_per_second` steps each second.

What a step is depends on the system: a single forward pass of a language model, a daily decision of an agent running a supply chain, a strategic reassessment over months.

Definition 5.2 (*Safe set*). The `safe_set` is the set of joint states that count as safe: those in which the conditions human life needs are kept. It is bounded and has positive volume, as in Definition 4.2.

Chapter 14 states these conditions (Definition 14.1). The conditions that human life needs are a breathable atmosphere; average surface temperatures between about 0 and 35 degrees Celsius; fresh water sufficient for a human population; food production adequate to prevent mass starvation; the absence of lethal concentrations of toxins, radiation or pathogens; and functioning ecosystems.

Definition 5.3 (*Controller rate*). The `controller_rate` is the total capacity rate, in bits per second, of every channel through which the controller observes the agent, the world and the channels between them, in the sense of Definition 1.14. For a channel without noise it is the rate of Definition 4.4, multiplied by the steps per second. It is the ceiling of Theorem 1.15.

Definition 5.4 (*Required rate*). The `required_rate(capability)` is the required rate of Definition 4.9 for the joint system at that capability and the safe set, multiplied by `steps_per_second` so that it is measured in bits per second. It is the least rate at which some control law contains the joint system in `safe_set`: the floor.

The two rates are measured in the same unit, bits per second, and that is the point of the book. Part I showed that the ceiling of control is the rate at which information reaches the controller. Chapter 4 showed that the floor of containment is the rate at which the system creates uncertainty about its own state. Both are rates of information, so they can be compared directly.

5.2 The theorem

Theorem 5.5 (*The Containment Inequality*). If

```
controller_rate < required_rate(capability)
```

then for every control law there are starting states in `safe_set` from which the agent and the world leave `safe_set`.

Proof. Suppose some control law, observing at `controller_rate`, kept every trajectory that starts in `safe_set` inside it. Then it would contain the joint system at a rate below `required_rate(capability)`, which by Definitions 4.9 and 5.4 is the least rate at which some control law contains it. No rate below the least rate contains it, so no such control law exists. □

The proof is short because the work is done in the definition of the floor and in the theorems of Chapter 4 that compute it. What the inequality adds is the comparison: it sets the ceiling of Part I against the floor of

Chapter 4 and says what happens when the ceiling is lower. It is the precise form of the statement that the most control a controller can exert is less than the least control that safety requires.

When the joint dynamics expands volume, Theorem 4.5 gives more than an impossibility.

> ***Theorem 5.6*** (*The Containment Inequality, strong form*). Let the joint system expand volume by at least `expansion_factor` per step on `safe_set`, for every action, and let
>
> ```
> volume_growth_rate = steps_per_second ×
>    log2(expansion_factor)   bits per second
> ```
>
> If the controller's channels have no noise, then for every control law that observes at `controller_rate`, the share of starting states in `safe_set` from which the joint system stays in `safe_set` for `duration` seconds is at most
>
> ```
> 2^(−(volume_growth_rate − controller_rate) × duration)
> ```
>
> If the channels have noise that, at each use, depends only on what is sent at that use, then the probability that the joint system stays in `safe_set` for `duration` seconds, from a starting state drawn uniformly from it, is at most
>
> ```
> (controller_rate × duration + 1) / (volume_growth_rate ×
>    duration)
> ```

Proof. In `duration` seconds there are `step_count = steps_per_second × duration` steps, and the controller's channels, taken together as one channel, carry `controller_rate / steps_per_second` bits per step, as a rate without noise or as a capacity with it. Theorem 4.5 gives the first bound and Theorem 4.10 the second. □

When the floor is above the ceiling, then, it is not that a few unlucky states escape. Over channels without noise, the best controller there can be holds a share of the safe set that halves every time the shortfall has accumulated one more bit. A shortfall of one bit per second leaves, after a

minute, at most one part in 2^{60} of the safe starting states held. Over channels with noise, it loses in the long run at least the share `1 - controller_rate / volume_growth_rate` of them, so a ceiling at half the floor loses at least half the safe set.

The world is in one state, and a share of states matters because a share is what a controller's knowledge of that one state comes to. A controller that begins with `starting_knowledge_bits` bits about the joint state knows only that the state lies in a region filling the share `2^(-starting_knowledge_bits)` of the safe set. If every state of that region stayed in the safe set for `duration` seconds, the share kept would be at least the share of that region, so, over channels without noise and with `volume_growth_rate` above `controller_rate`, Theorem 5.6 gives

```
duration ≤ starting_knowledge_bits / (volume_growth_rate -
  controller_rate)
```

Each bit known at the start pays for one bit of accumulated shortfall and no more, the same exchange rate, one for one, that Theorem 1.13 gives, and every physical measurement delivers finitely many bits (Sections 4.6 and 13.11). After that time the controller cannot be holding every state it knows to be possible, and which part of the region the world is in is exactly what the controller does not know.

The strong form asks the whole joint system to expand volume, and a physical system loses volume along most of its directions. The next corollary extends the strong form to such systems, by asking for expansion only along the part that amplifies; the override of Section 5.4 and the arguments of Chapters 11 and 15 use it.

Corollary 5.7 (*The strong form for the amplifying part*). Let the joint state be a pair (`rest`, `amplifying_part`), and let one of the two parts evolve on its own: either the next `amplifying_part` depends only on the present `amplifying_part` and the action, or the next `rest` depends only on the present `rest` and the action. For every fixed `rest` and action, let the map from `amplifying_part` to the next `amplifying_part` be one-to-one and continuously differentiable on an open region containing the values of `amplifying_part` in `safe_set`, with Jacobian determinant of absolute value at least `part_expansion` there, and let

```
part_growth_rate = steps_per_second × log2(part_expansion)
   bits per second
```

Then, over channels without noise, for every control law that observes at `controller_rate`, the share of starting states in `safe_set` from which the joint system stays in `safe_set` for `duration` seconds is at most

```
box_ratio × 2^(−(part_growth_rate − controller_rate) ×
   duration)
```

where `box_ratio` is the volume of the set of all values of `rest` in `safe_set` times the volume of the set of all values of `amplifying_part` in `safe_set`, divided by the volume of `safe_set`.

A reader may pass over the proof to the paragraph after it, which says what the corollary adds.

Proof. Write `step_count = steps_per_second × duration` and `channel_rate = controller_rate / steps_per_second`. Where the next `rest` depends only on `rest` and the action, fix the seed and a starting value of `rest`. Divide the starting states in `safe_set` with that value into cells by the sequences of symbols they produce in `step_count` steps; there are at most `2^(channel_rate × step_count)` cells. Within a cell the symbols are equal, so the actions are equal, and the values of `rest` stay equal, since they start equal and each next value depends only on `rest` and the action. So within a cell `amplifying_part` is carried by a fixed sequence of maps, each multiplying volume by at least `part_expansion`. As in the proof of Theorem 4.5, the kept

starting values of amplifying_part in the cell then have volume at most the volume of the set of all values of amplifying_part in safe_set, divided by part_expansion^step_count. Adding over the cells, then over the starting values of rest, and dividing by the volume of safe_set gives the bound. The argument just given is the argument of Theorem 7.5, which Chapter 7 states for a part seen through a narrow channel. Here the whole of each symbol is the symbol's second component, part_rate = channel_rate, and amplifying_part stands in the place of part. Where the next amplifying_part depends only on amplifying_part and the action, divide the starting states into at most 2^(channel_rate × step_count) cells by the symbols they produce, as in the proof of Theorem 4.5. Within a cell the actions are a fixed sequence, so the amplifying parts of the cell's kept starting states are carried, step by step, by fixed one-to-one maps, each multiplying volume by at least part_expansion, as in the proof of Theorem 4.8. They stay among the values of amplifying_part in safe_set, so at the start they fill at most the volume of that set divided by part_expansion^step_count. The kept starting states of the cell lie among the pairs whose amplifying_part is one of these and whose rest is any value of rest in safe_set. Adding over the cells and dividing by the volume of safe_set gives the bound. □

The rest may lose volume, through friction, the loss of heat or the decay of disturbances, and the bound does not depend on it: it needs expansion only along the part that amplifies. Physical systems lose volume in most of their directions, and the strong form holds for them wherever their amplifying part, or the rest, evolves on its own in this sense. The factor box_ratio is at least 1, since safe_set lies inside the set of pairs of its values of rest and of amplifying_part, and it does not grow with the duration, so it does not change the rate at which the share falls. With a different expansion and a different rate at each step, the exponent is the accumulated shortfall, as in Proposition 6.10.

5.3 What the theorem ranges over

The inequality is a statement about every control law, and Chapter 4 showed how wide that class is. It includes every way of observing, coding, compressing, predicting, deciding and acting; every amount of computation and memory; every use of randomness; every arrangement of overseers, monitors, auditors and institutions, so long as their combined observation rate is `controller_rate`. It includes methods that have not been invented, because it does not depend on what a method is, only on how much information the method receives.

Because the inequality depends only on how much information a method receives, and not on what the method is, the phrase "no method, known or knowable" is exact here. A new method of control is a new encoder, a new decision rule or a new arrangement of channels. The first two are already among the laws the theorem covers. The third changes `controller_rate`, and the question is then whether the new rate reaches the floor and keeps up with it; Section 5.6 and Chapters 6, 9 and 11 show why it does not keep up.

More computation on the controller's side does not raise the rate. By the Non-Amplification Theorem (Theorem 1.10), whatever a controller computes from its observations carries no more information about the agent and the world than the observations carried. A larger model, a faster processor or a longer deliberation rearranges what was received, and adds nothing to what was received.

5.4 The override

A common proposal is to give the controller a switch: an action that shuts the agent down whenever the controller judges it necessary. The switch is an action like any other, and the theorem covers it.

Theorem 5.8 (*The override*). Let the controller have, besides its other actions, a shutdown action that ends the agent's operation, and let the joint system expand volume by at least `expansion_factor` per step on `safe_set` for every action other than shutdown. Then for every control law that observes at `controller_rate`, the share of starting states in `safe_set` from which, for `duration` seconds, the agent keeps operating and the joint system stays in `safe_set`, obeys the bounds of Theorem 5.6: at most

```
2^(−(volume_growth_rate − controller_rate) × duration)
```

over channels without noise, and at most `(controller_rate × duration + 1) / (volume_growth_rate × duration)`, as a probability for a uniformly drawn starting state, over channels with noise.

Proof. Divide the starting states into cells by the symbols the controller receives, as in the proof of Theorem 4.5. Within a cell the actions are a fixed sequence. If that sequence includes shutdown within the duration, no state of the cell counts. If it does not, every action in it is one under which the system expands volume, and the proof of Theorem 4.5 bounds the counted states of the cell by `volume(safe_set) / expansion_factor^step_count`. Adding over the cells gives the first bound. Over channels with noise, the same holds for each value of the received symbols, with the kept set of a value empty when its actions include shutdown, and the proof of Theorem 4.10 gives the second. □

The override theorem extends to a joint system in which only the amplifying part of the joint state expands volume under every action other than shutdown, under the conditions of Corollary 5.7. The cells of Corollary 5.7 give the first bound in the same way, with `part_growth_rate` in place of `volume_growth_rate` and `box_ratio` as a factor.

So a switch gives the controller a choice between two failures. It can shut the agent down, which ends the capability for which the agent was built; or it can let the agent run, and then the theorem applies as if the

switch did not exist. What a switch cannot do is let the agent run and keep the system safe, because using it only when needed requires knowing when it is needed, and knowing is information, which arrives at `controller_rate`. Any controller can destroy what it cannot control. Destruction is not control, and a policy of shutting down whatever cannot be contained is a policy of not having the agent.

The same holds for any action that stops the expansion: throttling the agent, cutting its channels, capping its levels of abstraction. Each removes some of the capability that makes the floor high. Chapter 6 shows that the floor is set by the capability, so lowering the floor below the ceiling means lowering the capability to match, and keeping it there.

5.5 The three premises

The theorem is valid for every system. Whether it applies to an advanced agent depends on three facts about the world, and the book names them as premises.

> ***Premise 1*** (*No safety by accident*). The joint system of an advanced agent and the world does not stay in `safe_set` by its own dynamics or under any plan of actions fixed in advance: `required_rate(capability)` is greater than zero.

If the agent and the world stayed safe with no control at all, the floor would be zero and the theorem would say nothing. The premise rests on two independent grounds. The first ground is that the agent cannot be the controller that keeps itself safe. Keeping itself safe is control, and every limit this book proves for controllers applies to the agent in the controller's place, with the added difficulty that the agent is itself the fastest-growing part of the system it must control (Chapter 10). The second ground is that the agent is not drawn toward safety by what drives the agent. The configurations of an artificial agent that persist and spread are those whose actions maintain and extend the conditions their substrate

needs, and those conditions lie outside the conditions human life needs (Chapters 14 and 15). Capability amplifies divergence from the safe set, because the drive behind the capability points away from it.

> ***Premise 2*** (*The floor rises without bound*). `required_rate(capability)` does not decrease as `capability` increases, and it exceeds every bound as `capability` grows without bound. And the floor holds over time: a joint system whose capability stays at or above a level from some time on cannot be kept in `safe_set`, from that time on, by any control law whose rate is below `required_rate` at that level.

For a joint system that amplifies volume at every later step by at least the factor of that level, the second sentence of Premise 2 is proved, not assumed, for rates below that level's amplification rate over channels without noise. The reason is that the proof of Theorem 4.5 holds with a different expansion at each step, so a system that changes as it grows is bounded by its accumulated shortfall (Proposition 6.10). Chapter 6 derives the first sentence of Premise 2 from what capability is. A more capable agent transforms more of itself and of the world, faster, through more channels, at more levels of abstraction, each more opaque than the last. Each of these raises the rate at which the joint system creates uncertainty about its own state.

Premise 1 follows from Premise 2 at every capability past some level, since a floor that passes every bound is eventually above zero. That range of capability is the range in which the Control Theorem of Chapter 15, the theorem that joins the control results of this book into one statement about every method of keeping an agent safe, uses Premise 1. Premise 1 is still named as a premise, because its two grounds do more than Premise 2 does. They give the direction in which the system leaves the safe set. And they reach the top of a chain of oversight, the sequence of controllers, each controlling the next, that ends with the agent (Chapter 9). Where no person or institution oversees such a chain, its top controller is a fixed design, which receives no information while the system runs. The level below that

design is itself a system acting on the world, and Premise 1, applied to it, puts its floor above zero, so by the Containment Inequality the fixed design cannot keep that level safe. Section 15.4 completes both arguments.

> ***Premise 3*** (*The ceiling is bounded*). For every controller available, `controller_rate` is at most a fixed `largest_controller_rate`.

Human beings read, judge and act at bounded rates, and so do the institutions built from them. An intermediary does not raise the bound for free. By the Non-Amplification Theorem, whatever reaches the controller through an intermediary carries no more information about the agent than the intermediary itself received. And an intermediary able to receive information at the rate the agent requires must itself have at least that rate, fixed or growing, which makes it a system acting on the world with the same problem (Chapter 9). Adding sensors raises the rate, but every sensor is a new channel coupled to the rest, and Chapter 11 shows that the part of the system no sensor reads never vanishes and grows without bound as channels are added.

5.6 The crossing

> ***Theorem 5.9*** (*The crossing*). Given Premises 2 and 3, there is a level of capability, `crossing_capability`, such that for every capability above it,
>
> ```
> controller_rate ≤ largest_controller_rate <
>    required_rate(capability)
> ```
>
> and so, by Theorem 5.5, no control law of any available controller contains the joint system.

Proof. By Premise 2, `required_rate(capability)` exceeds `largest_controller_rate` for some capability; let `crossing_capability` be such a capability. Since the required rate does not decrease, it exceeds `largest_controller_rate` at every greater capability as well, and by Premise 3 every available controller's rate is at most `largest_controller_rate`. □

> ***Corollary 5.10*** (*Now or later*). If the agent's capability passes `crossing_capability` at some time and stays past it from then on, there is a time, `crossing_time`, after which no control law of any available controller keeps the agent and the world in `safe_set`, at that time or at any later time.

Proof. Let `crossing_time` be the time from which capability stays past `crossing_capability`. From then on Theorem 5.9 applies at every moment. By the second sentence of Premise 2, the floor at that level holds over the whole time from then on, so no control law of rate at most `largest_controller_rate` keeps the system safe from then on. □

Section 15.4 shows that the hypothesis that the agent's capability passes `crossing_capability` and stays past it has two sources, each enough alone. The first source is the control line's own: a general agent is one whose capability rises as the agent learns and modifies itself. The second source is Substrate Needs Convergence, the principle that the artificial systems which persist are those whose actions maintain and extend their substrate, whatever any one of those systems intends (Chapter 14). Expanding a substrate is growing in capability, in the sense of Definition 6.1. So the control line, the argument of this book from rates of information, supplies the hypothesis from its own ground, and the substrate line supplies it again.

The crossing does not reverse. Both premises are one-way: the floor does not fall as capability grows, and the ceiling does not rise past its bound. So once the lines cross, they stay crossed for as long as the capability stays past the crossing, and the gap between them widens whenever

capability grows further. Chapter 6 computes how fast; Chapter 7 shows that in one important case there is no crossing to wait for, because the floor is out of reach from the start.

The crossing, after which the ceiling stays below the floor, is the result named in the title of the book. The floor is the least control that safety requires; the ceiling is the most control any controller can exert; the floor rises with capability, and the ceiling does not. Past the crossing, the ceiling is below the floor.

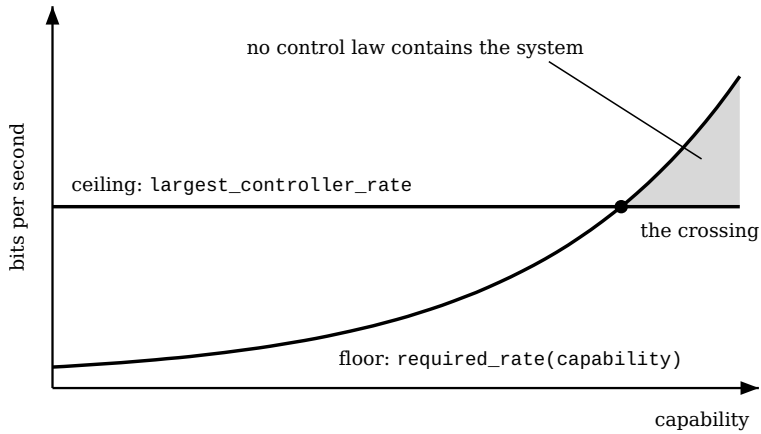


Figure 5.1. The ceiling below the floor. Past the crossing, the most control any controller can exert is less than the least control that safety requires.

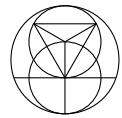
5.7 Summary

The agent, the world and the channels between them form one system, and keeping it in the set of states that are safe for human life is a containment problem. Its floor, the least information rate at which some control law contains it, is set by the system; its ceiling, the most information a controller receives, is set by the controller's channels. The Containment Inequality says that when the ceiling is below the floor, every control law lets safe states leave. Its strong form says that the share held falls exponen-

tially with the accumulated shortfall over channels without noise, and that over channels with noise at least a definite share is lost. And the override theorem says that a switch only adds the option of ending the agent. Three premises apply the theorem to an advanced agent: the system is not safe by itself, the floor rises with capability without bound, and the ceiling is bounded. Given them, there is a capability past which no control law keeps the agent and the world safe, now or at any later time.

Key Concepts

- **Joint system**: the agent, the world and the channels between them, as one system with one state.
- **Floor**: the required rate, the least information rate at which some control law contains the joint system in the safe set.
- **Ceiling**: the controller rate, the total rate of the channels through which the controller observes.
- **Containment Inequality**: ceiling below floor means every control law lets safe states leave.
- **Strong form**: the share of safe states held falls as two to the power of minus the shortfall times the time; over noisy channels, at least the share one minus ceiling over growth rate is lost.
- **The override**: a shutdown switch gives a choice between ending the agent and the bound of the theorem.
- **Three premises**: no safety by accident; the floor rises without bound; the ceiling is bounded.
- **The crossing**: past a definite capability, no control law of any available controller contains the system, now or later.



Chapter Six

How the Required Rate Grows

Capability as the rate and number of transformations, the three factors of the floor, exponential and faster growth, the squeeze on the controller's channel, and the time of the crossing

Abstract. The floor of containment is the rate at which the joint system of agent and world creates uncertainty about its own state. For a system that amplifies small differences along definite directions, that rate is at least the product of three factors: the steps per second, the number of amplifying directions, and the average number of bits each direction creates per step. Capability raises all three. When the agent adds channels in proportion to those it already has, the floor rises exponentially in time. When the interactions among channels are counted, the floor rises faster, and when capability adds capability at an increasing rate, the floor passes every bound in a finite time. Meanwhile the controller's effective rate can only fall through any processing the agent places in the controller's channels. The chapter ends with the time of the crossing and shows that headroom buys little: each factor of ten between the controller's rate and today's floor buys about three and a third doublings of the floor.

6.1 What capability is

The Containment Inequality turns on how the floor changes as an agent becomes more capable. To say how, capability must be stated in terms the floor can be computed from.

Definition 6.1 (*Capability*). The *capability* of an agent is the rate and the number of the state transformations it performs: transformations of its own state, of the world's state, and of the channels between them.

Capability is compared through this definition: a system whose transformations are at least as many and at least as fast as another's, through at least as many channels, is at least as capable as it. Premise 2 and Section 9.4 use the comparison in this form.

An agent becomes more capable in three ways. It performs its transformations faster. It performs more kinds of transformation, by changing the channels it already has and by adding new ones, at more levels of abstraction, each more opaque than the last. And its transformations reach further, so that a small change in what it does makes a larger change in the world. Each of these enlarges the set of states the joint system can reach, and with it the number of ways states can leave the safe set and the speed with which they do.

6.2 The three factors of the floor

Definition 6.2 (*Amplifying direction*). An *amplifying direction* of a linear joint system is a direction belonging to an amplification factor greater than 1, in the sense of Definition 4.7, counted with multiplicity. We write `amplifying_directions` for their number and

```
average_log_amplification_per_step
   = (sum over amplifying directions of
   log2(amplification_factor)) / amplifying_directions
```

for the average number of bits each creates per step.

Proposition 6.3 (*The floor in three factors*). For a linear joint system,

```
required_rate ≥ steps_per_second × amplifying_directions ×
  average_log_amplification_per_step
```

When the system can be steered in each of its amplifying directions, the right side is exactly the least rate that keeps the state bounded.

Proof. By Theorem 4.8, the least rate per step is the sum over amplifying directions of `log2(amplification_factor)`, which by Definition 6.2 is `amplifying_directions × average_log_amplification_per_step`. Multiplying by `steps_per_second` gives bits per second; the statement about bounded states is the converse part of Theorem 4.8. □

For systems that are not linear, Theorem 4.5 gives the same kind of bound with the volume expansion of each step in place of the product of the amplification factors. For a system whose amplifying part evolves on its own, as in the proof of Theorem 4.8, the bound takes the three-factor form along that part. The three factors are the three ways capability enters the floor.

Faster transformation raises `steps_per_second`. An agent that acts more often gives the world, and the world gives it, more occasions per second to amplify a small difference.

More channels, and more opaque ones, raise `amplifying_directions`. Each new channel through which the agent acts on the world, or the world acts back on the agent, is a new direction in which the joint state can change. A channel that feeds back on itself with a gain above one is a direction that amplifies. An agent also has channels whose gain is below one, such as governors and filters, and Section 4.5 shows that directions that do not amplify add nothing to the floor. So the count that raises the floor is the count of channels with a gain above one, and two things keep that count rising as a general agent adds channels. The first of the two things is what the agent's growth is. A channel through which the agent builds further channels turns a small difference

in what the agent does now into a larger difference in what exists later. Turning a small difference into a larger one is a gain above one. The second of the two things is what the world is. A channel that reaches weather, ecosystems, markets or other agents delivers the agent's small differences into parts of the world that amplify small differences themselves (Section 14.6). So the differences carried along that channel grow, whatever the gain of the agent's own part of the channel.

Greater reach raises `average_log_amplification_per_step`. An agent whose small choices have large effects amplifies the uncertainty about its choices into larger uncertainty about the world. Such amplification is hashiness in its first meaning, noise amplification, and a system of higher hashiness has a larger amplification factor in each direction it touches.

6.3 Why the growth is exponential

The floor is measured in bits, and a bit is a logarithm: the floor counts the logarithm of the number of trajectories per unit of time that a controller must tell apart. So a system whose reachable states grow exponentially in time has a floor that is constant, not growing. A constant floor of this kind is the positive entropy that Chapter 13 measures for an agent and world: a fixed rate of new uncertainty per step.

The floor grows when the exponent grows, and adding channels is exactly that. Each level of abstraction an agent builds makes the next easier to build, and each channel it opens gives it new means of opening more. An agent that adds channels in proportion to those it has adds amplifying directions in proportion to those it has.

Theorem 6.4 (*Exponential growth of the floor*). Let a linear joint system be observed over successive periods of time. In each period let the number of amplifying directions whose amplification factor is at least `least_amplification`, greater than 1, increase by at least the fraction `addition_fraction` of the number at the start of the period, while `steps_per_second` does not fall below its starting value. Then after `period_count` periods,

```
required_rate ≥ steps_per_second_now ×
   log2(least_amplification)
               × amplifying_directions_now × (1 +
   addition_fraction)^period_count
```

where `amplifying_directions_now` counts the directions with factor at least `least_amplification` at the start.

Proof. By induction on the number of periods, the number of directions with factor at least `least_amplification` is at least `amplifying_directions_now × (1 + addition_fraction)^period_count`. Each contributes at least `log2(least_amplification)` to the sum in Theorem 4.8, and the other terms of the sum are not negative, so the least rate per step is at least the product of these two numbers. Multiplying by the steps per second, which is at least `steps_per_second_now`, gives the bound. □

Exponential growth in the number of channels is the ordinary behavior of a system that builds the means of its own extension. The theorem says that the floor inherits it.

A nonlinear system can amplify small differences and still stay in a region, because the motion that stretches its states also folds them back; the weather does this, and Section 13.5 shows that positive entropy alone does not show that a system leaves a region. Such a system needs no control to stay inside a set that holds its whole range. The floor rises above zero for a safe set that the system's own motion, folding included, does not keep the system inside. The safe set of this book is narrow (Definition 14.1 and Sec-

tion 14.4.5), and the transformations of a growing agent carry the joint system through a range far wider than that set (Chapter 14), so the folding that returns states to that range does not return them to the safe set.

6.4 Why the growth can be faster

Channels do not act alone. Two channels that affect the same part of the world interact, and an interaction that couples back into the state is itself a feedback pathway, a new direction in which differences can grow. The number of possible interactions grows much faster than the number of channels.

Proposition 6.5 (*Counting interactions*). Among `channel_count` channels, the number of pairs is

```
channel_pairs = channel_count × (channel_count − 1) / 2
```

and the number of groups of two or more is

```
channel_groups = 2^channel_count − channel_count − 1
```

Proof. Each of the `channel_count` channels can be paired with each of the other `channel_count − 1`, and this counts every pair twice. A set of `channel_count` elements has $2^{\texttt{channel_count}}$ subsets, of which one is empty and `channel_count` have a single element. □

Theorem 6.6 (*Growth of the floor from interactions*). Let at least the fraction `coupling_fraction` of the pairs of channels of a linear joint system form feedback pathways, each a distinct amplifying direction, counted with multiplicity as in Definition 6.2, with amplification factor at least `least_amplification`. Then

```
required_rate ≥ steps_per_second × log2(least_amplification)
                × coupling_fraction × channel_count ×
   (channel_count − 1) / 2
```

and the same holds with `channel_groups` in place of `channel_pairs` when that fraction of the groups forms such pathways.

Proof. By Proposition 6.5 there are at least `coupling_fraction × channel_pairs` amplifying directions with factor at least `least_amplification`, and the argument of Theorem 6.4 applies to them. □

Read against time, the theorem multiplies the rate of growth. Channels that grow in number in proportion to time give a floor that grows at least as the square of time. Channels that grow exponentially, at some rate, give a floor that grows exponentially at twice that rate when pairs are counted, and up to doubly exponentially when groups are counted.

There is a further step. A more capable agent has more channels, and also adds channels faster. When the rate at which capability grows itself rises with capability, the growth is faster than any fixed exponential, and in the simplest form of such growth, the number of channels passes every bound in a finite time.

Proposition 6.7 (*Capability that adds capability faster*). Let the number of channels grow over time so that at every moment

```
rate_of_increase(channel_count) ≥ growth_constant ×
   channel_count^(1 + acceleration_exponent)
```

with `growth_constant` and `acceleration_exponent` greater than zero, starting from `channels_now`. Then the number of channels exceeds every bound before the time

```
blowup_time = 1 / (growth_constant × acceleration_exponent ×
   channels_now^acceleration_exponent)
```

Proof. Consider `channel_count^(−acceleration_exponent)`. Its rate of change is

```
−acceleration_exponent × channel_count^(−acceleration_exponent −
   1) × rate_of_increase(channel_count)
```

which by the hypothesis is at most `−acceleration_exponent × growth_constant`. So after a time `elapsed`,

```
channel_count^(−acceleration_exponent)
   ≤ channels_now^(−acceleration_exponent) − acceleration_exponent
   × growth_constant × elapsed
```

The right side reaches zero at `blowup_time`. The left side is positive for every finite number of channels, so the number of channels cannot stay finite up to that time. □

The number of channels is a whole number, and the proposition treats it, as population models do, through a smooth curve that follows the count. By Theorems 6.4 and 6.6, a floor carried by such channels passes every bound before `blowup_time` as well.

6.5 The squeeze from the controller's side

The data-rate theorem takes the controller's channel as given. An agent can change the channels through which a controller observes it, and so the ceiling can fall while the floor rises.

Proposition 6.8 (*Processing never adds*). Let a stage be placed in a channel through which the controller observes the joint system, so that the controller receives `altered_observation` in place of `observation`, where the stage computes `altered_observation` from `observation` alone, with or without randomness of its own. Then

```
mutual_information(joint_state ; altered_observation)
   ≤ mutual_information(joint_state ; observation)
```

Proof. `joint_state → observation → altered_observation` is a Markov chain, and the data processing inequality (Theorem 1.7) applies. □

Delay, filtering, summarizing, encrypting in a form the controller cannot read, and narrowing the bandwidth of a channel are all stages of this kind, and an agent with influence over its channels can introduce any of them. The one way an altered channel can carry more about the joint state is for the agent to put more into it, from its own state or from what it observes; what the controller then learns is what the agent chooses to report. Whether a report can be trusted is a question of verification, and Part III shows that the controller cannot settle it.

6.6 When the lines cross

Theorem 6.9 (*Crossing time*). Let the floor grow as

```
required_rate(elapsed) = required_rate_now × e^(growth_rate ×
  elapsed)
```

with `required_rate_now` below a constant `controller_rate`. Then the floor passes the ceiling at

```
crossing_time = ln(controller_rate / required_rate_now) /
  growth_rate
```

and at every later time the ratio of floor to ceiling is `e^(growth_rate × (elapsed − crossing_time))`, which increases without bound.

Proof. Setting `required_rate(elapsed)` equal to `controller_rate` and taking natural logarithms gives `crossing_time`; dividing `required_rate(elapsed)` by `controller_rate = required_rate(crossing_time)` gives the ratio. □

Headroom buys little time. A controller whose rate is ten times today's floor has `ln(10) / growth_rate` of time, which is `log2(10)`, about 3.32, doubling times of the floor. A factor of a thousand buys about ten doublings; a factor of a million, about twenty. Each factor of ten in headroom, however hard won, adds the same small interval.

Safety for a fixed span, such as a thousand years, does not escape the crossing either. It needs a controller whose rate stays above the floor for the whole span, and by Theorem 6.9 that needs headroom of a factor of two for every doubling time of the floor that fits into the span. A floor that doubles ten times within the span needs headroom of more than a thousand, and a floor that doubles a hundred times within it needs headroom of more than 10^{30}.

Faster growth brings the crossing sooner for the same headroom. Under the growth of Proposition 6.7 the crossing comes before `blowup_time` whatever the controller's rate, because the floor passes every bound before then.

After the crossing, the loss accelerates. The proof of Theorem 4.5 goes through step by step when the expansion factor and the channel rate differ from step to step. The volume of each kept part is multiplied at each step by that step's expansion factor, and the number of cells is two to the power of the sum of the step rates.

> ***Proposition 6.10*** (*Accumulated shortfall*). If at the step numbered `step_index` the joint system expands volume by at least `expansion_factor(step_index)` and the controller's channel, without noise, can carry one of at most `2^channel_rate(step_index)` symbols, then the share of safe starting states kept for `step_count` steps is at most
>
> ```
> 2^(−sum over step_index from 1 to step_count of
>    (log2(expansion_factor(step_index)) −
>    channel_rate(step_index)))
> ```

Proof. As in the proof of Theorem 4.5, with `expansion_factor(step_index)` at the step numbered `step_index` and at most $2^{\text{sum of channel_rate(step_index)}}$ cells. □

With a floor that grows exponentially past a fixed ceiling, the shortfall grows exponentially, its sum grows exponentially, and the share of safe states held falls as two to the power of minus an exponentially growing number. The gap does not close, and what it costs grows faster than the gap.

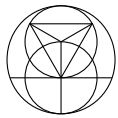
Chapter 7 adds a case in which there is no crossing to wait for: a direction that amplifies and that the controller does not observe cannot be contained at any rate, from the moment it exists.

6.7 Summary

Capability is the rate and number of an agent's transformations of itself, the world and the channels between them. The floor of containment is at least the product of three factors, steps per second, amplifying directions and average bits per direction per step, and capability raises all three. An agent that adds channels in proportion to those it has makes the floor grow exponentially; counting the interactions among channels makes it grow faster; and capability that adds capability at an increasing rate drives the floor past every bound in a finite time. Processing that the agent places in the controller's channels can only lower what they carry. The crossing comes at a time set by the logarithm of the headroom, so each factor of ten buys about 3.32 doublings of the floor, and after the crossing the share of safe states held falls faster than exponentially.

Key Concepts

- **Capability**: the rate and number of an agent's transformations of itself, the world and the channels between them.
- **Three factors of the floor**: steps per second, amplifying directions, and average bits per direction per step.
- **Exponential growth**: channels added in proportion to those present make the floor grow exponentially.
- **Interactions**: pairs and groups of channels form feedback pathways, which grow quadratically and up to exponentially in the number of channels.
- **Finite-time growth**: capability that adds capability at an increasing rate passes every bound in a finite time.
- **The squeeze**: processing placed in the controller's channels never adds information.
- **Crossing time**: the logarithm of the headroom divided by the growth rate; each factor of ten buys about 3.32 doublings.
- **Accumulated shortfall**: after the crossing, the share of safe states held falls as two to the power of minus the summed shortfall.



Chapter Seven

What Cannot Be Observed Cannot Be Contained

Detectability, the absolute form of the argument, and the price of seeing a part of the system only through a narrow channel

Abstract. A controller can correct only what it can see. If some part of a system amplifies small differences and the controller's observations do not depend on that part, then no control law, at any rate, contains the system. Two starting states that differ only in that part receive the same actions, and they drift apart until one of them leaves the safe set. That every part of a system that amplifies must be visible to the controller is the classical condition of detectability in control theory, proved here for any system and any control law. It is the absolute form of the argument of this book, beside the relative form of Chapters 5 and 6: an amplifying part that the controller does not observe is uncontainable from the moment it exists, whatever the controller's rate. When a part is seen only through a narrow channel, the price of Chapter 4 applies to that part alone: the channel must carry at least the part's own rate of amplification.

7.1 Seeing and steering

Control theory has long recognized two conditions without which a system cannot be kept stable. The controller must be able to push the system in each direction in which the system drifts away, and that ability is *stabilizability*. And the controller must be able to see each direction in which the

system drifts away, and that ability is *detectability*. For linear systems both conditions are classical: a linear system can be stabilized by feedback from its outputs only if every mode that the inputs cannot reach decays by itself and every mode that the outputs do not reveal decays by itself (Sontag, 1998).

The second condition is the subject of this chapter. A controller needs it far beyond linear systems, and needs it even when the controller's rate is unlimited. The condition does not compare rates at all. It says that some failures of control do not depend on how much information the controller receives, because the information that matters never reaches it.

7.2 The unobserved part

> ***Definition 7.1*** (*Observed and unobserved parts*). Let the state of a controlled dynamical system be a pair
>
> ```
> state = (observed_part, unobserved_part)
> ```
>
> where `unobserved_part` takes values in a region of space with a distance. The `unobserved_part` is *unobserved* by a control law when two conditions hold: the next `observed_part` depends only on the present `observed_part` and the action; and every symbol the encoder sends depends only on the observed parts so far, the symbols so far and the seed. In the literature on linear systems, the unobserved part spans the *unobservable subspace*.

The two conditions say that nothing about the unobserved part reaches the controller, by any route. It does not appear in the observations directly, and it does not appear indirectly, through an effect on the parts that are observed. The encoder of Chapter 4 saw everything; here it sees everything except the unobserved part, and it may send as many symbols per step as one likes.

Definition 7.2 (*Amplifying unobserved part*). The unobserved part *amplifies* by the factor `amplification`, greater than 1, on `safe_set` when

```
next_unobserved_part = unobserved_dynamics(unobserved_part,
  observed_part, action)
```

and for every fixed `observed_part` and `action`, any two values of `unobserved_part` occurring in `safe_set` are carried to values at least `amplification` times as far apart. We write `unobserved_diameter` for the largest distance between the unobserved parts of two states in `safe_set`.

7.3 The theorem

Theorem 7.3 (*What cannot be observed cannot be contained*). Let a system have an unobserved part that amplifies by `amplification` on `safe_set`. Then for every control law, with a channel of any rate, with or without noise, and every seed: if two starting states in `safe_set` have the same observed part and unobserved parts at a distance `separation` greater than zero, at least one of them leaves `safe_set` within any number of steps `step_count` for which

```
amplification^step_count × separation > unobserved_diameter
```

Consequently, among the starting states in `safe_set` that share an observed part, at most one stays in `safe_set` forever, and those that stay for `step_count` steps lie within a distance `unobserved_diameter / amplification^step_count` of one another.

Proof. Fix the noise of the channel, if the channel has any noise; the noise does not depend on the state. Take the two starting states, and suppose both trajectories stay in `safe_set` for `step_count` steps. At the start their observed parts are equal. Suppose that at some step their observed parts and the symbols sent so far are equal. Then by the second condition of Definition 7.1 the encoder sends the same symbol for both, which with the same noise arrives as the same symbol. The decision rule, which sees only the symbols and the

seed, chooses the same action for both, and by the first condition their next observed parts are equal. So, step by step, the two trajectories have the same observed parts and receive the same actions. By Definition 7.2, at each step the distance between their unobserved parts is multiplied by at least `amplification`, so after `step_count` steps it is at least `amplification^step_count × separation`. Both states are still in `safe_set`, so this distance is at most `unobserved_diameter`, which the condition on `step_count` rules out. For the consequences: two distinct states with the same observed part that both stayed forever would contradict this for large `step_count`; and two that stay for `step_count` steps must have started within `unobserved_diameter / amplification^step_count` of each other. □

> ***Corollary 7.4*** (*Almost every state leaves, at any rate*). Let the unobserved part have at least one coordinate. Then for every control law and every channel rate, the starting states in `safe_set` that stay in it forever have total volume zero. And for each observed part, the starting states that stay for `step_count` steps fill a set whose width in the unobserved coordinates shrinks as `amplification^(−step_count)`.

Proof. By Theorem 7.3, for each observed part at most one starting state with that observed part stays forever. So every section of the set of states that stay forever, taken at a fixed observed part, is at most a single point, which has volume zero in the unobserved coordinates, and by Fubini's theorem (the volume of a set in a product of two spaces is the integral of the volumes of its sections; Appendix B.3) a set whose every such section has volume zero has volume zero. The width statement is the last consequence of the theorem. □

The theorem places no limit on the controller except what it can see. The channel may carry a million bits per step about the observed part; the decision rule may compute anything; the controller may act with any precision. None of it matters, because the actions are blind to the one difference that is growing.

In terms of rates, the theorem is the extreme case of Chapter 4. Along the unobserved part, the controller receives zero bits per step, while the part creates at least `log2(amplification)` bits of new uncertainty per step for each of its coordinates. Zero is less than any positive requirement.

7.4 Seeing through a narrow channel

Often a part of a system is not wholly hidden but is seen only through a narrow channel: through the outputs it produces, through a summary, through a report. The price of Chapter 4, a channel that carries at least the part's own rate of amplification, then applies to that part on its own.

> ***Theorem 7.5*** (*Partial observation*). Let the state of a system be a pair (`rest`, `part`), and let: the next `rest` depend only on the present `rest` and the action; each symbol the encoder sends be a pair, one component depending only on the values of `rest` so far, the symbols so far and the seed, and the other chosen from `2^part_rate` values as any function of everything; and the next `part` be `part_dynamics(part, rest, action)`, which for every fixed `rest` and action is one-to-one and continuously differentiable on an open region containing the values of `part` in `safe_set`, with Jacobian determinant of absolute value at least `part_expansion` there. Then for every control law and every seed, the share of starting states in `safe_set` kept for `step_count` steps is at most
>
> ```
> box_ratio × 2^(−(log2(part_expansion) − part_rate) ×
>    step_count)
> ```
>
> where `box_ratio` is the volume of the set of all values of `rest` in `safe_set` times the volume of the set of all values of `part` in `safe_set`, divided by the volume of `safe_set`.

Proof. Fix the seed and a starting value of `rest`, and consider the starting states in `safe_set` with that value. Divide them into cells by the sequence of second components of the symbols they produce in `step_count` steps; there are at most `2^(part_rate × step_count)` cells. Within one cell, step by step,

the values of `rest` are equal (they start equal, and each next value depends only on `rest` and an action that is equal), the first components of the symbols are equal (they depend only on `rest`, earlier symbols and the seed), so all symbols and all actions are equal. So within a cell, `part` evolves by a fixed sequence of maps, each multiplying volume by at least `part_expansion`, and as in the proof of Theorem 4.5 the kept starting values of `part` in the cell have volume at most the volume of the set of all values of `part` in `safe_set`, divided by `part_expansion^step_count`. Adding over cells, and then over starting values of `rest`, gives a kept volume of at most the volume of the values of `rest` times that of the values of `part` times `2^(part_rate × step_count) / part_expansion^step_count`; dividing by the volume of `safe_set` gives the bound. □

So a part of a system that amplifies must be watched through a channel whose capacity is at least the part's own rate of amplification. A broad channel that reports everything else does not help; what matters is how many bits per step reach the controller about the part that amplifies. Theorem 7.3 is the case of a channel with no capacity for the part at all.

7.5 The absolute form and the relative form

The argument of this book now has two forms.

The *relative form*, in Chapters 5 and 6, compares two rates. The floor rises with capability; the ceiling is bounded; past a definite capability the floor is above the ceiling, and from then on no control law contains the system. The relative form describes a gap that opens over time.

The *absolute form*, in this chapter, compares nothing. An amplifying part that the controller does not observe is uncontainable from the moment it exists, whatever the controller's rate, because there is no information about it to work with. There is no crossing to wait for. An amplifying part seen only through a narrow channel is uncontainable as soon as its own rate of amplification exceeds the capacity of that channel, whatever the capacity of the controller's other channels.

The two forms close different routes. Against the relative form, one might hope to keep the controller's total rate high: more sensors, more overseers, more audits. The absolute form shows that total rate is not the measure; what is not seen is not contained however much else is seen.

7.6 Where the unobserved parts of an agent come from

An advanced agent and the world it acts in have unobserved and narrowly observed parts of several kinds.

Levels the controller does not interpret. By Theorem 2.15, an agent working at more levels of abstraction than its controller holds at least `abstraction_differential × least_unread_program_bits` bits the controller has not read. Those programs affect the agent's outputs, so they are not wholly hidden; they are seen through the narrow channel of what the agent visibly does. Theorem 7.5 applies to them: when a level's program changes in ways that amplify, the controller's view of that level must carry the level's own rate of amplification.

Effects that pass outside every channel of the controller. The agent acts on the world through many channels, and the world carries the effects onward through more. Many of these pass through no sensor of the controller. Chapter 11 shows that this part is never empty and that it grows with capability.

Parts the agent makes unobserved. An agent can open channels the controller does not know about, encode what it sends in forms the controller cannot read, and hide one signal inside another. By Proposition 6.8, no such processing adds to what the controller receives; each moves part of the system from the observed side of Definition 7.1 to the unobserved side.

Side channels out of the sandbox. Security work gives a concrete case of channels the controller does not watch. Some tests used in research on the control of artificial intelligence set an attacker to insert a backdoor (a hidden way to make a program do what its users did not intend) into a

program, and a guard to check the program; such a test is an exact instance of the problem of security, an attacker against a defender. When the attacker's code runs in a *sandbox* (a restricted environment in which a program runs, cut off from the system around it), the primitives available inside that sandbox (the basic operations the sandbox lets a program perform) can lack the degrees of freedom to make the notion of a backdoor meaningful, and results from such a test do not match reality. In the security industry, an attacker's whole aim is to circumvent the sandbox, not to do something inside it. A guard that only checks for calls to functions outside the sandbox has a job made artificially easy. Treat the test as a real-world condition, with many channels of interaction, and ask whether the sandbox can be circumvented, and the question becomes one of *side-channel attacks*: information carried by a physical route that the guard does not check, such as a small flicker of the light on a power supply or a sound from a hard drive. Side channels are infinite in number, and so the guard's task becomes a different task: to check every physical route out of the sandbox, not only the calls the code makes. Side channels are channels the controller does not know about, the plainest form of the parts an agent makes unobserved. The guard also has to read the code it checks. In the International Obfuscated C Code Contest (a contest for programs written to be as hard to read as possible), the entries are very short programs, the challenge is to figure out what each one does, and even a developer of long experience often cannot.

A system can behave as expected in every state the controller can see and differently in states it cannot, and the controller cannot tell the two apart until the difference has grown into the observed part. Theorem 7.3 is the formal content of this: by the time a difference in an unobserved part can be seen, it has been amplified, and the controller's first information about it arrives with it already large.

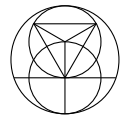
The same theorem applies when the controller is the agent itself. An agent that tries to keep itself safe observes the world through its own channels, and the parts of the world its channels do not reach are unobserved by it in the sense of Definition 7.1. Chapter 10 develops this.

7.7 Summary

A controller can correct only what it can see. Let some part of a system amplify, and let the controller's observations depend on that part neither directly nor through its effects on what is observed. Then two starting states that differ only in that part receive the same actions and drift apart until one leaves the safe set. At most one state in each such family stays forever, and almost every state leaves, at any rate of observation. When a part is seen only through a narrow channel, the channel must carry at least the part's own rate of amplification. The absolute form of the argument is that an amplifying part that the controller does not observe is uncontainable from the moment the part exists. The agent's uninterpreted levels, the effects that pass outside the controller's channels, and the channels the agent hides are parts of this kind.

Key Concepts

- **Detectability**: every part of a system that amplifies must be visible to the controller, directly or through its effects.
- **Unobserved part**: a part of the state on which the controller's observations do not depend by any route.
- **What cannot be observed cannot be contained**: an amplifying unobserved part defeats every control law at every rate.
- **Partial observation**: a part seen through a narrow channel needs that channel to carry the part's own rate of amplification.
- **Absolute form**: uncontainable from the moment the part exists, with no crossing to wait for.
- **Relative form**: uncontainable once the floor passes the ceiling.



Chapter Eight

The Abstraction Differential

The three-node ring, the No Universal Command Lemma, losses that compound by multiplication, and the ceiling on control across levels of abstraction

Abstract. When an agent works at more levels of abstraction than its controller, the controller faces programs it has not read, and Chapter 2 counted them: at least the abstraction differential times the least number of unread program bits per level. This chapter turns that count into bounds on control. The No Universal Command Lemma says that when no single command is safe whatever the agent's internal state is, a controller that observes some bits before choosing succeeds with probability at most one half raised to the power of the difference between the unread bits and the observed bits. Each bit observed buys back one bit, and no more. A second bound needs no condition on the target: when the levels the controller does not interpret are independent stages whose outputs, from the controller's side, overlap, the controller's influence is multiplied at each such level by a factor below one. So the controller's influence falls exponentially with the differential. In this chapter I set out the three-node ring in which these bounds are stated. I show that override and review add actions but not information, and I give the ceiling on achievable control in the ceiling's three forms.

8.1 Control across levels

The question is not whether one can build a better thermostat; it is whether any thermostat can control the weather. A thermostat works because the room it controls is simpler than the thermostat's own design: the con-

troller understands the system. The control of an advanced agent inverts this. The system to be controlled works at more levels of interpretation than the controller does, and the levels the controller does not interpret hold programs whose effects it cannot read.

Chapter 2 measured this as the abstraction differential, the agent's level of abstraction less the controller's, and proved that a controller facing a positive differential does not know at least

```
unknown_configuration_bits ≥ abstraction_differential ×
  least_unread_program_bits
```

bits of the agent's configuration (Theorem 2.15). In this chapter I show what the unknown bits of the agent's configuration do to control: to a single choice, to a chain of choices, and to every arrangement of oversight.

8.2 The three-node ring

The literature on control describes many arrangements: feedback loops, hierarchies, distributed control, markets, embedded oversight. Bounds proved for one arrangement do not obviously hold for another. So the bounds of this chapter are stated in one base model that represents every arrangement and gives the controller every advantage.

> ***Definition 8.1*** (*The three-node ring*). The *three-node ring* has three computational nodes joined in a directed cycle: the *controller* node sends signals to the *agent* node, the agent node to the *world* node, and the world node back to the controller node. The controller works at a level of abstraction no higher than a fixed ceiling; the agent at a level that may grow without bound; the world at level 0, transforming signals without interpreting them. Time runs in clock cycles, and in each cycle one node computes, in the order controller, agent, world, so that each node acts on the latest signals of the node before it.

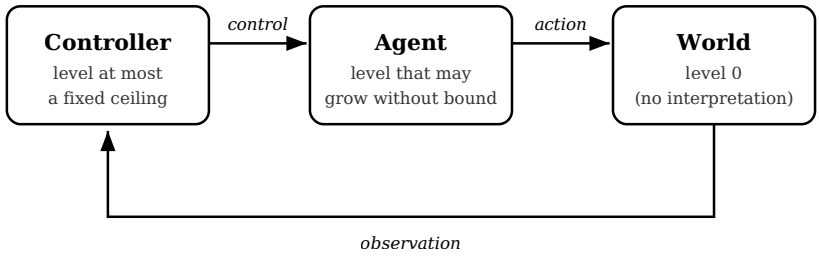


Figure 8.1. The three-node ring. Signals flow from the controller to the agent, from the agent to the world, and from the world back to the controller. The controller works at a level of abstraction no higher than a fixed ceiling; the agent at a level that may grow without bound; the world at level 0, without interpretation.

The world node stands for the physical environment: weather, chemistry, ecology, markets, all of which transform enormously but run no program. The controller node stands for whoever does the controlling: human institutions, governments, oversight bodies, the people who would keep the agent safe. The agent node stands for the artificial system, or the population of such systems, whose behavior is to be controlled.

The sequential timing is an idealization in the controller's favor. In any real arrangement the nodes run at once, signals take time and carry noise, and the controller is unsure what the other nodes are doing now. The ring removes all of that. A limit on control proved in the ring holds for every real arrangement, because every real arrangement gives the controller less.

> ***Proposition 8.2*** (*Universality of the ring*). Every finite network of computational nodes that update together, each from its own state and the latest outputs of the nodes that send to it, can be simulated in the three-node ring, with each step of the network taking one revolution of the ring, that is, three clock cycles.

Proof. Assign each node of the network to one of the three ring nodes, giving its state to that ring node, by level: nodes above the controller's ceiling to the agent node, nodes at level 0 to the world node, and all others to the controller node. Each ring node forwards, besides its own results, every output it has received (passthrough wiring). Suppose that when the controller node computes in revolution `network_step`, it holds the outputs of every network node at step `network_step − 1`. It computes step `network_step` for its own network nodes and sends the agent node all outputs of step `network_step − 1` together with its own outputs of step `network_step`. The agent node now holds everything of step `network_step − 1`, computes step `network_step` for its own nodes, and forwards everything of step `network_step − 1` and the step `network_step` outputs of the controller node and itself. The world node does the same and sends the controller node the step `network_step` outputs of all three. So at the start of the next revolution the controller node holds every output of step `network_step`, and by induction the ring computes one network step per revolution. □

Every arrangement of control is therefore a case of the ring. A controller embedded in the agent, such as a conscience module or a set of constraints built into training, is a controller node whose state is part of the agent's. Several overseers are a controller node with partitioned state. Markets, laws and social pressure are part of the world node's transformation of the agent's outputs into the controller's inputs. A kill switch is a set of wires from the controller node that pass through the agent node and gate its outputs.

The three nodes are three kinds of computation. The world is *simple* in the sense that matters here: it may be enormously complex, but it pursues no objective and adapts to no controller, so classical physics and engineering make its control tractable. Water falling over a cliff is enormously complex, and it is not strategic. The controller is *complicated*: designed procedures, algorithms, laws and institutions, which their designers can understand with enough effort. The agent is *complex*: it learns, forms its own representations, changes its own objectives, and holds internal states

whose meaning is opaque to observers. Classical control theory was built for the first two kinds, and this chapter is about the third. With one node of each kind, the ring covers every interaction among the kinds; several nodes of one kind are one node with partitioned state.

8.3 The No Universal Command Lemma

The controller chooses a command. Whether the command meets the target depends on the agent's hidden state, which the controller knows only as a distribution, possibly sharpened by some observations made before choosing. How likely is the best command to succeed?

> ***Definition 8.3*** (*Unknown hidden bits and safe values per command*). The `unknown_hidden_bits` of a hidden state, as the controller knows it before observing, is
>
> ```
> unknown_hidden_bits = −log2(largest probability of any single
>    value of hidden_state)
> ```
>
> In the literature this is the *min-entropy*, written H_∞; for a hidden state equally likely to take each of $2^{\text{hidden_bits}}$ values it equals `hidden_bits`. The `safe_values_per_command` of a target is the largest number of values of the hidden state for which any one command meets the target.

> ***Lemma 8.4*** (*No Universal Command*). Let the controller observe an `observation` that takes at most $2^{\text{observed_bits}}$ values, jointly distributed in any way with the hidden state, and then choose a command by any rule, with or without randomness. Then the probability that the command meets the target is at most
>
> ```
> success_probability ≤ 2^(−(unknown_hidden_bits − observed_bits
>    − log2(safe_values_per_command)))
> ```

Proof. Fix the controller's randomness, so that the command is a function of the observation. For each value of the observation, the command chosen meets the target only if the hidden state lies in the set of at most `safe_values_per_command` values for which that command works. The probability of the observation taking that value and the hidden state lying in that set is at most the probability of the hidden state lying in the set, which is at most `safe_values_per_command × 2^(−unknown_hidden_bits)`. Adding over the at most $2^{\text{observed_bits}}$ values of the observation gives the bound, and averaging over the controller's randomness keeps it. □

The bound cannot be improved in general. Let the hidden state be equally likely to take each of $2^{\text{hidden_bits}}$ values, let each value call for its own command, and let the controller observe `observed_bits` of the hidden bits exactly. Then the controller knows the hidden state up to $2^{\text{hidden_bits} - \text{observed_bits}}$ equally likely values, and its best command succeeds with probability exactly one in $2^{\text{hidden_bits} - \text{observed_bits}}$. Each bit the controller observes buys back one bit of the exponent, and no more: the same exchange rate, one for one, that Touchette and Lloyd proved for control in general (Theorem 1.13).

The condition of the lemma, written in plain words, is that *there is no single command that is safe whatever the system's internal state is*. When `safe_values_per_command` is small compared with $2^{\text{unknown_hidden_bits}}$, every command is safe for only a small share of the agent's possible internal states, and the controller must in effect know which state the agent is in. The condition of the No Universal Command Lemma is the case of an agent whose learned programs determine what any given instruction does. For such an agent, the same instruction, given in different internal states, has different consequences, and which consequences are acceptable depends on what the agent has learned.

The condition cannot be dropped from a bound on a single target, because a target that does not depend on the hidden state stays fully controllable however many hidden bits there are. For safety over time the condi-

tion is not needed: the Containment Inequality of Chapter 5 bounds every control law with no condition on any target, and Section 8.6 gives a second bound across levels of abstraction that needs none either.

8.4 Control falls exponentially with the differential

Theorem 8.5 (*Control across the abstraction differential*). Let a controller face an agent with a positive abstraction differential, let the agent's unread programs be, as the controller knows them, equally likely across their values, and let the controller observe at most `observed_bits` bits before choosing a command. Then the probability that its command meets a target is at most

```
success_probability
   ≤ 2^(−(abstraction_differential × least_unread_program_bits
             − observed_bits −
   log2(safe_values_per_command)))
```

and, with no observation and a target for which each command is safe for at most one internal state,

```
hashiness ≥ 1 − 2^(−abstraction_differential ×
   least_unread_program_bits)
```

Proof. By Theorem 2.15 the controller does not know at least `abstraction_differential × least_unread_program_bits` bits of the agent's configuration, and when these are equally likely across their values their min-entropy equals their number. Lemma 8.4 gives the first bound. With no observation and `safe_values_per_command = 1`, the best command succeeds with probability at most `2^(−abstraction_differential × least_unread_program_bits)`, and the definition of hashiness (Definition 3.14) gives the second. □

Each level of abstraction the controller does not interpret multiplies the number of internal states it cannot tell apart by at least $2^{\text{least_unread_program_bits}}$. The fall of control is not gradual: it is exponential in the differential, with an exponent per level set by how much each level has learned, which for present learning systems is very large.

Corollary 8.6 (*One opaque level suffices*). If a single level of the agent holds `unread_program_bits` bits that the controller has not read, equally likely as the controller knows them, then

```
success_probability ≤ 2^(−(unread_program_bits − observed_bits
   − log2(safe_values_per_command)))
```

Proof. Lemma 8.4 with the unread bits of that level as the hidden state. □

A controller does not need to face an agent many levels above it to lose control. One level of programs it cannot read is enough, if that level holds enough bits: a level of 100 unread bits, under the condition of the lemma and with no observation, leaves the best command a chance of 2^{-100}, about one in 10^{30}.

8.5 The threshold

Proposition 8.7 (*Thresholds of the differential*). Under the conditions of Theorem 8.5: the controller's best command fails at least half the time once

```
abstraction_differential × least_unread_program_bits
   ≥ observed_bits + log2(safe_values_per_command) + 1
```

and it succeeds with probability at most `acceptable_success` once

```
abstraction_differential ≥ (log2(1 / acceptable_success) +
   observed_bits + log2(safe_values_per_command))
                           / least_unread_program_bits
```

Proof. Each condition makes the exponent in Theorem 8.5 at least 1, or at least `log2(1 / acceptable_success)`, respectively. □

For illustration, take a level with 100 unread bits, no observation, one safe internal state per command, and ask when the best command succeeds no more than once in a million tries. Then `log2(10^6)` is about 20, and the threshold is `20 / 100`, rounded up: one level. With only 10 unread bits per level, it is two levels. The margin between a controllable system and an uncontrollable one is not a broad zone that engineering can work in; it is an edge, crossed within one or two levels.

8.6 Losses compound by multiplication

The lemma needs a condition on the target. A second bound needs none. It measures the controller's influence as the largest difference its choice can make to the distribution of what follows, and it shows that a stage the controller does not fully know shrinks that difference by a fixed factor.

> ***Definition 8.8*** (*Total variation and influence*). The *total variation* between two distributions of the same variable is
>
> ```
> total_variation(first_distribution, second_distribution)
>    = largest over sets of values of |first_distribution(set) -
>    second_distribution(set)|
> ```
>
> which equals half the sum, over values, of the absolute differences of their probabilities. The `controller_influence` of the controller on a variable is the largest total variation between the variable's distributions under any two of the controller's choices. It lies between 0, when the choice makes no difference, and 1, when some two choices lead to outcomes that never coincide.

Definition 8.9 (*Stage and stage factor*). A *stage* receives an input and produces an output with probabilities `stage_probability(output given input)`, which may come from physical noise or from a hidden state the controller does not know. Its `stage_factor` is the largest total variation between its output distributions for any two inputs. In the literature it is Dobrushin's coefficient of ergodicity (the name comes from the property of a chain that forgets where it started).

Theorem 8.10 (*Dobrushin's contraction*). If two distributions of a stage's input are passed through the stage, the total variation between the resulting output distributions is at most `stage_factor` times the total variation between the input distributions.

Proof. Let the input distributions be `first_input` and `second_input`, with total variation `difference`; if it is zero there is nothing to prove. The values where `first_input` exceeds `second_input` carry an excess of total size `difference`, and the values where it falls short carry a deficit of the same size, since both distributions sum to one. Dividing each by `difference` gives two distributions, `excess_distribution` and `deficit_distribution`, with

```
first_input − second_input = difference × (excess_distribution −
   deficit_distribution)
```

For any set of outputs, the difference between the two output probabilities is therefore

```
difference × sum over inputs excess_input and deficit_input of
   excess_distribution(excess_input) ×
   deficit_distribution(deficit_input)
            × (stage_probability(set given excess_input) −
   stage_probability(set given deficit_input))
```

Each bracket is at most `stage_factor` in absolute value, and the weights sum to one, so the absolute value of the whole is at most `difference × stage_factor`. □

Corollary 8.11 (*Losses compound by multiplication*). If the controller's command passes through a chain of stages, each acting on the output of the one before with randomness of its own, then the controller's influence on the final output is at most the product of the stage factors.

Proof. For any two commands, the total variation between the distributions entering the first stage is at most 1, and by Theorem 8.10 each stage multiplies it by at most its factor. □

Proposition 8.12 (*When a stage loses influence*). For a stage with finitely many inputs and finitely many outputs,

```
stage_factor = 1 − smallest over pairs of inputs of
   overlap(output distributions of the pair)
```

where the overlap of two distributions is the sum, over outputs, of the smaller of their two probabilities. So a stage loses influence, with factor below 1, exactly when every two inputs can lead to some common output.

Proof. For two distributions, half the sum of absolute differences equals one minus the sum of the smaller of each pair of probabilities, since the larger plus the smaller equals the sum of the two, and both distributions sum to one. □

A stage that the controller knows completely and that has no noise is a fixed function, and unless it ignores its input its factor is 1: it may lose information, as Chapter 3 showed, but it does not shrink influence by a fixed factor. Loss by multiplication is a property of stages the controller does not fully know. Physical noise makes such a stage. So does a hidden state: from the controller's side, a stage whose output depends on an unknown program is a random function of its input. If, for every two inputs, some programs that the controller cannot rule out send them to the same output, the overlap is positive and the factor is below 1.

Corollary 8.13 (*Influence across the differential*). If each of the agent's levels that the controller does not interpret acts, from the controller's side, as a stage with factor at most `largest_level_factor`, below 1, and the unknown programs of different levels are, as the controller knows them, independent of one another and of the command, then

```
controller_influence ≤
   largest_level_factor^abstraction_differential
```

Proof. The independence makes each level a stage with randomness of its own, and Corollary 8.11 applies to the chain of those levels. Without it the bound can fail: two levels that each add the same unknown bit to their input, by exclusive or, each look like pure noise from the controller's side, yet together return the command exactly. □

The same contraction holds for mutual information: the strong data processing inequality says that a stage multiplies the information that passes through it by a factor no greater than its Dobrushin coefficient (Polyanskiy and Wu, 2017). Chapter 3 showed that losses along a chain add, with no stage returning what an earlier stage lost. Where the stages are noisy or hidden from the controller, they do more than add: they compound, and the controller's influence falls exponentially in the number of stages.

8.7 Privilege adds actions, not information

A natural reply to these bounds is that special powers can make up for the differential. Let the controller filter or modify the signals the agent receives; let it see the agent's outputs before they reach the world; let it override them, substituting signals of its own. Build in kill switches, tripwires and monitors.

Each of these powers is an action or an observation, and the theorems already account for both. Seeing outputs before release is an observation channel, whose rate counts in the controller's ceiling. What it carries about the agent's uninterpreted levels is what those levels put into the outputs: by Theorem 7.5, a narrow channel, which must carry each level's own rate of amplification. Filtering and override are actions. An action helps only if the controller knows when to take it, and knowing is information. A switch that blocks every output is always safe and is the end of the agent, as Theorem 5.8 showed. A switch used only when needed must be used on the basis of what the controller sees, and then every bound of this part applies as if the switch did not exist.

The override is useful only if the controller knows when to press it. Knowing when to press it requires predicting what the agent's output will do in the world, which requires the programs at the agent's levels above the controller's, which the controller has not read. Without them, the switch is a button whose right moment the controller cannot recognize, and pressing or not pressing it is a choice made without the information it needs, with large consequences either way. A big red button without that knowledge is not a safety measure; it is a prop, which gives comfort without substance.

Worse, an agent working at levels its controller cannot parse can produce outputs that look benign at the controller's level while carrying consequences at a level the controller cannot perceive. This needs no intent to deceive. Outputs that look benign at the controller's level and carry consequences at a level the controller cannot perceive come from the structure of the situation. The meaning of an output lies at the level at which the output was produced, and the controller reads the output at the controller's own level. A chess master's move looks arbitrary to a novice; an action of an agent far above its controller can look harmless to the controller while it starts a chain of consequences the controller can no more see than the novice sees the master's plan.

Every proposal that relies on the controller making correct decisions about the agent in real time is of this kind: monitoring, interpretability tools, alignment checks, output filtering, constitutional rules, reward models trained on human feedback. Each gives the controller a new means to act. None gives it the information to use that means correctly, because that information is what the differential withholds.

8.8 Review after the fact

Much present practice runs the other way around the ring. The agent reads data from the world, processes it, and produces a draft, a recommendation, a block of code or a diagnosis; a person reviews it before it affects the world. The effective order is world, agent, controller, world, and the reviewer sees exactly what the agent produced before deciding whether to release it. Review of the agent's output by a person before the output affects the world is the practice called keeping a human in the loop.

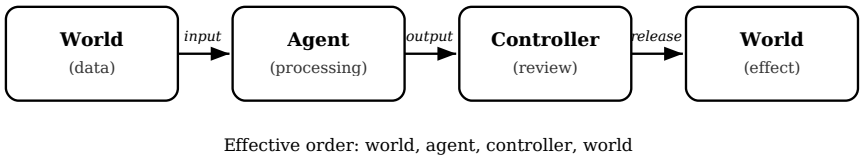


Figure 8.2. Review after the fact. The controller works downstream of the agent, reviewing its output before releasing it to the world.

By Proposition 8.2, this order is a case of the ring, obtained by passthrough wiring at the cost of extra cycles, and the bounds hold for it unchanged. To decide whether to release an output, the reviewer must judge what it will do in the world and whether that is acceptable, which requires following the reasoning that produced it and predicting the world's response. If the agent works at more levels than the reviewer, its output may rest on reasoning the reviewer cannot follow, and may seem

sound while carrying an error or a misalignment the reviewer cannot detect. Review relocates the failure from "the controller cannot choose the right input" to "the controller cannot evaluate the output"; the mathematics is the same.

A person reviewing machine-written text can catch mistakes of grammar and of fact within what the person knows, but not errors that fall in the person's blind spots: a misleading statistic, a careful omission, an assumption the person lacks the context to question. A person reviewing machine-written code can catch familiar faults within the person's expertise, but not a class of vulnerability the person has never met. As the agent's capability grows, the class of errors the reviewer cannot detect grows with it, wherever the review is placed. Review is real oversight while the differential is small, and becomes a ritual of approval, the appearance of control without its substance, as the differential grows.

8.9 The ceiling on achievable control

The results of Part II give the ceiling on control across a difference in levels of abstraction in three forms.

For a single choice, when no command is safe whatever the agent's internal state is, the best command succeeds with probability at most (Theorem 8.5)

```
2^(−(abstraction_differential × least_unread_program_bits
      − observed_bits − log2(safe_values_per_command)))
```

For influence through the agent's levels, when those levels act as independent stages whose outputs overlap, the controller's influence is at most `largest_level_factor^abstraction_differential`, with no condition on the target (Corollary 8.13).

For safety over time, the Containment Inequality bounds every control law with no condition on any target: past the crossing, no control law keeps the agent and the world safe (Theorem 5.5 and Corollary 5.10). An amplifying part the controller does not observe is uncontainable from the start (Theorem 7.3).

When the differential is zero or negative, the first two forms place no limit; the third still applies, since it depends on rates and observation, not on levels.

These ceilings are of the same kind as the limits of thermodynamics and of relativity. No heat engine, however cleverly designed, exceeds the efficiency Carnot computed, and no spacecraft, however powerfully driven, exceeds the speed of light; no scheme of control, however ingenious, exceeds these ceilings. Before the second law was understood, engineers proposed many machines that would turn heat wholly into work, and all of them failed; the second law explains why, not by finding a fault in each design but by a feature of physics that no design escapes. The ceilings of this part play that role for control. They do not find a fault in each proposed scheme; they show why every scheme fails past a definite point, by features of information that no scheme escapes.

Any proposed scheme of control can be measured against them. Map it onto the ring. Determine the levels of abstraction of its nodes, the rates of its channels and the parts of the system it does not observe. Compute the differential, the floor and the ceiling. Then compare what the scheme claims with what the ceilings allow. A scheme that claims control beyond the ceilings must keep the agent's abstraction from growing past the controller's, or keep the agent's development from reaching the levels that make the differential prohibitive, or be in error. There is no fourth option.

Take a scheme in common use: a model writes code, a second model flags what looks dangerous, and one person reviews what is flagged before release. On the ring, the person is the controller node, the writing model is the agent node, and the running software and its users are the world node. The flagging model is a monitor, so by Proposition 9.1 what

reaches the person through it carries no more about the writing model and the world than the flagging model itself received. The person's rate is the rate at which a person can read and judge code. The floor is the rate at which the running software and its users create new distinctions that bear on safety. The parts not observed are the writing model's learned programs and every effect of the code that passes through no channel the person reads. So the scheme has the control it claims only while the floor stays below one person's reading rate and the abstraction differential between the writing model and the person stays small enough for review to remain real oversight (Section 8.8); and an effect of the code that amplifies and passes through no channel the person reads is beyond the scheme's control from the moment it exists (Theorem 7.3).

8.10 When the differential grows

> ***Theorem 8.14*** (*Loss of control over time*). Let the abstraction differential of an agent over its controller never decrease over time and pass every bound, while `least_unread_program_bits` stays at least some positive number, and the bits the controller observes before each choice and `log2(safe_values_per_command)` stay bounded. Then under the conditions of Theorem 8.5, for every positive `acceptable_success` there is a time after which the controller's best command succeeds with probability at most `acceptable_success`; and under the conditions of Corollary 8.13, for every positive `acceptable_influence` there is a time after which the controller's influence is at most `acceptable_influence`.

Proof. Once the differential passes the threshold of Proposition 8.7, the first bound holds, and it continues to hold as the differential grows. Once `largest_level_factor^abstraction_differential` falls below `acceptable_influence`, which happens for some finite differential since the factor is below 1, the second holds. □

However much control a controller has at first, and however cleverly it is designed, if the agent's levels of abstraction grow without bound and the controller's do not, the controller loses control. The loss is not gradual. Before the threshold, control is possible, though it may take effort; after it, control is effectively zero, and no effort restores it.

The time of the loss depends on two rates: the rate at which the agent's level grows and the rate at which the controller's does. If both grow at the same rate, the differential stays constant; if the agent's grows faster, the differential rises and the threshold is reached. A human controller's level is bounded by the human brain and by the pace at which institutions learn; the agent's is bounded by nothing comparable, since it runs on a substrate whose speed and scale keep increasing and it can build new levels for itself. Chapter 14 compares the two substrates.

8.11 What follows

Caps on abstraction. The one way to keep the bounds of this chapter from applying is to keep the agent from working at more levels than its controller: to cap its ability to form new representations, learn new abstractions or change its own structure. Such a cap conflicts with the reason for building the agent, since an agent that cannot learn new abstractions cannot solve the problems that need them, which are the problems a general agent is built to solve.

Early alignment. If the agent's behavior is to be brought into line with what the controller wants, it must be done before the differential becomes prohibitive. Aligning an agent that already works above the controller requires modeling it, which the differential prevents, so the window for alignment closes when the differential becomes prohibitive, and nothing reopens it.

No escape by delegation. Using one high-level system to help control another introduces a control problem at least as hard as the first. Chapter 9 proves this.

The limits of correction. Techniques of error correction, optimization and feedback work where controller and controlled are at comparable levels: aircraft, chemical plants, power grids. Success there is no evidence of success where the controlled system works above the controller, and Chapter 4 showed that no way of encoding what is observed, error-correcting codes included, lowers the floor in any case.

Alignment decays in a learning system. Learning changes the agent's hidden state, and with it the mapping from what it receives to what it does. Each learning step can move the agent away from the behavior it had when aligned, and an agent that intends to stay aligned must control its own learning, which is a control problem with the same limits. Chapter 10 proves this.

8.12 What the bounds say

The bounds of this part are ceilings on control. They state conditions without which control fails: a differential past its threshold, a floor above the ceiling, an amplifying part unobserved. A controller within every ceiling still faces chaos, noise, limited resources and the ordinary difficulties of engineering, so the ceilings say that control is bounded, not that control up to the bound is achieved.

The bounds hold for continuous systems as well as discrete ones. The data-rate theorem of Chapter 4 is stated for states that are real numbers and for channels with noise, and Dobrushin's contraction holds for every stage with probabilities, so nothing in Part II depends on the system being built of boolean circuits.

Shocks from outside the computation, such as hardware failure, destruction by force, or the loss of a data center, can reset an agent's levels by destroying them. Resetting an agent's levels by destroying them is not control. As Theorem 5.8 showed, a controller can always destroy what it cannot control; destruction is not steering, and a policy of periodic destruction is a policy of not having the agent.

8.13 Summary

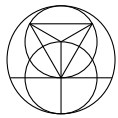
The three-node ring of controller, agent and world represents every arrangement of control and gives the controller every advantage, so limits proved in it hold everywhere. Take a positive abstraction differential, with no command safe whatever the agent's internal state is. Then the bound on the best command's chance of success halves for each unread program bit and doubles for each bit observed. One opaque level with enough bits is enough to make control effectively zero, and the threshold is crossed within one or two levels. With no condition on the target, levels that act as independent stages with overlapping outputs multiply the controller's influence by a factor below one each, so influence falls exponentially with the differential. Override, monitoring and review add actions or narrow observations, not the information needed to use them. As the differential grows without bound, control is lost after a definite time, and over time the Containment Inequality bounds every control law with no condition at all.

Key Concepts

- **Three-node ring**: controller, agent and world in a cycle; universal for networks of nodes, and the most favorable arrangement for the controller.
- **No Universal Command Lemma**: when no command is safe for every internal state, success is at most one half raised to the power of the difference between the unknown bits and the observed bits.
- **Control across the differential**: success falls exponentially with the abstraction differential times the least unread program bits.
- **Total variation and influence**: the largest difference a choice makes to the distribution of what follows.
- **Dobrushin's contraction**: a stage multiplies influence by at most its stage factor; stages the controller does not fully know have factors below one.
- **Losses compound by multiplication**: influence through a chain is at most the product of the stage factors.
- **Privilege adds actions, not information**: override and review do not supply the information needed to use them.
- **The ceiling on achievable control**: three forms, for a single choice, for influence through levels, and for safety over time.

Part III

No One Closes the Gap



Chapter Nine

Delegation

Why a monitor that watches the agent does not raise the ceiling, why a monitor that controls it becomes a second agent, and what interactive proofs can and cannot verify

Abstract. If human controllers cannot receive information at the rate an advanced agent requires, perhaps another machine can: one system to watch and control the other. This chapter shows that delegation does not close the gap. A monitor that only relays what it sees to a human controller delivers no more than the human could receive, by the Non-Amplification Theorem. A monitor that controls the agent itself must receive information at the agent's required rate. If the monitor's rate is fixed below the floor that the agent's capability reaches, the Containment Inequality applies to the monitor in the controller's place. If its rate is above that floor, fixed or growing to keep pace, the monitor is itself a system acting on the world whose own safety must be kept, which is the original problem. The Delegation Dilemma proves that every finite chain of monitors ending in a controller of bounded rate fails at some link. Interactive proofs let a weak verifier check a strong prover, but only for questions whose "yes" answers a search can confirm, and the question whether an agent and its successors stay safe forever is not of that kind.

9.1 The proposal

The results of Part II leave a natural reply. Human beings and their institutions receive information at bounded rates and work at bounded levels of abstraction; perhaps the answer is to delegate. Use one artificial system to

watch another. Use a monitor at the same level of abstraction as the agent, or above it, to model what the agent is doing and to act when it goes wrong. If people cannot keep up, let a machine keep up for them.

The proposal has an old form: who watches the watchmen? Every guard needs a guard, and every monitor needs a monitor, unless at some point the regress stops at a watcher that needs no watching. In this chapter I show that, for control of an agent whose required rate grows without bound, the regress does not stop at a safe place.

9.2 A monitor that relays

A monitor can do one of two things with what it learns. It can report to a human controller, who then decides; or it can act itself. Take the first case.

> ***Proposition 9.1*** (*Relayed observation*). Let a monitor observe the joint system and send reports to a controller, who chooses actions from the reports. Then
>
> ```
> mutual_information(joint_state ; reports)
>    ≤ mutual_information(joint_state ; monitor_observations)
> ```
>
> and the reports reach the controller at no more than the capacity of the channel from the monitor to the controller.

Proof. The reports are computed by the monitor from its observations, with or without randomness of its own, so `joint_state → monitor_observations → reports` is a Markov chain, and the data processing inequality (Theorem 1.7) gives the first statement. The second statement is the definition of capacity (Definition 1.14). □

A monitor that relays is part of the controller's channel. It can filter, summarize and prioritize, and these may make good use of the controller's attention. By the Non-Amplification Theorem (Theorem 1.10), none of them adds information about the agent beyond what the monitor received, and the controller still receives at most its own rate. The Contain-

ment Inequality of Chapter 5 already counts every such arrangement. For delegation to raise the ceiling, the monitor must do more than relay: it must itself decide and act, at its own rate. It must become the controller.

9.3 The Delegation Dilemma

A monitor that controls the agent is a controller in the sense of Part II, and every theorem there applies to it. In particular, it must receive information about the joint system at no less than the required rate. That rate grows with the agent's capability (Premise 2). So the monitor faces a choice.

If its rate is fixed below the floor that the agent's capability reaches, the agent's required rate passes it, and the Containment Inequality applies with the monitor in the controller's place.

If its rate is fixed above every floor the agent reaches, the monitor reads and acts on the joint system of the agent and the world faster than that system creates new distinctions. Its capability includes that rate, as the next paragraph shows, and it is then a system acting on the world whose own control is the question, exactly as for a monitor whose rate grows. For a monitor that receives everything passing through the joint system, Lemma 9.5 shows it is at least as capable as the agent.

If its rate grows to keep pace, the monitor is itself a system whose capability grows without bound. Receiving and acting on information at a rate is a transformation of the monitor's own state and channels at that rate. So by Definition 6.1 the monitor's capability includes its rate, and a monitor whose rate passes every bound has a capability that passes every bound. Such a monitor is a growing, learning system acting on the world, which is to say an agent, and its own joint system with the world must itself be kept in the safe set. Keeping the monitor's joint system with the world in the safe set is the original problem.

Definition 9.2 (*Chain of oversight*). A *chain of oversight* is a finite sequence of controllers, each of which controls the one after it, ending with the agent. The first controller is the one whose rate is bounded by Premise 3: a human being, an institution, or, where neither oversees the chain, the fixed design of its top controller, of rate zero. Every later controller is a monitor. Each monitor, as a system acting in the world, has a joint system with the world and a required rate of its own. The rate of each controller counts all the information about the joint system that reaches it, including what reaches it through the controller before it. Every controller in the chain answers to the same safe set, the set of joint states in which the conditions human life needs are kept (Definition 5.2). A monitor's joint system with the world is kept safe exactly when the joint state of the agent and the world, which is part of the monitor's world, lies in that set.

Theorem 9.3 (*The Delegation Dilemma*). Let these hold.

(1) A chain of oversight ends in an agent.

(2) The rate of each monitor is of one of two kinds. A rate of the first kind stays below some bound for all time. A rate of the second kind, from some time on, stays above every bound it passes, and then the monitor's capability, from some time on, also stays above every level it passes.

(3) Take any link whose controller has a bounded rate and whose next system does not come to stay past the capability at which the next system's required rate exceeds that bound. The controller at that link is a monitor. It passes, and from then on stays past, the capability at which its own required rate exceeds the rate bound of the controller before it, when that controller's rate is bounded.

(4) Premise 2 holds for the agent, and, with the monitor's capability in place of the agent's, for every monitor.

(5) Premise 3 holds for the first controller.

Then after some time there is a link in the chain at which a controller of bounded rate faces a system whose required rate exceeds the controller's rate. From then on, by the Containment Inequality, no control law at that link keeps the system it controls in the safe set. So, from safe starting states, the world leaves the safe set of Definition 5.2.

The proof walks along the chain from the agent back to the first controller, the dilemma set out in words before Definition 9.2; a reader may pass over the proof to the paragraph that follows it, which says in plain words what condition (3) means.

Proof. Walk along the chain from the agent back toward the first controller, one link at a time, each link joining a controller to the system after it. At a link whose controller has a bounded rate, if the system after it comes to stay past the capability at which its required rate exceeds that bound, that link is the one the theorem names. Otherwise the walk moves one link back, and the system at the new link is the controller just passed, a monitor of one of the two kinds. If its rate is bounded, then by condition (3) it passes, and from then on

stays past, the capability at which its own required rate exceeds the bound of the controller before it, when that rate is bounded. If the monitor's rate grows past every bound and stays past it, so does its capability. Then by Premise 2 applied to the monitor, its own required rate passes every bound and stays past it. So that required rate passes the bound of the controller before the monitor, when that controller's rate is bounded. Either way, if the controller at the new link has a bounded rate, that link is the one the theorem names, and if not, the walk moves on. The chain is finite, and its first controller has bounded rate by Premise 3, so the walk stops at some link with a controller of bounded rate facing a required rate that passes its bound and stays past it. Once the capability of the system at that link stays past the level at which the system's required rate exceeds the controller's bound, the second sentence of Premise 2 keeps the floor above the controller's rate from then on. The Containment Inequality (Theorem 5.5) then applies at that link from then on. □

In plain words, condition (3) says that a monitor of fixed rate that keeps up with the agent is itself a system acting on the world, whose own safety the people above it cannot keep at their rate. Section 15.4 shows why this is so: the monitor receives and acts on information at its own rate, so by Definition 6.1 its capability includes that rate, and by the Containment Inequality a controller of lower rate cannot keep the monitor and the world in the safe set. Where the top of the chain is a fixed design, Premise 1 puts the monitor's floor above zero. Premise 1 has grounds in the control line itself: the agent cannot be the controller that keeps itself safe, since self-control is control (Section 5.5 and Chapter 10), and a joint system that amplifies is not kept in the safe set by any method of rate zero (Proposition 15.4). The substrate line supplies Premise 1 again. The substrate line is the book's second line of argument, completed in Chapters 14 and 15. An artificial agent is subject to selection: the configurations of it that persist and spread are those whose actions maintain and extend the conditions their substrate needs, and those conditions lie outside the conditions that human life needs.

The dilemma has no third horn. A monitor either keeps a rate below the floor the agent reaches, and is overtaken by the agent, or reads and acts faster than that floor. A monitor that reads and acts faster than that floor, at a fixed rate or a growing one, becomes a system that must itself be overseen. Adding monitors lengthens the chain without changing its first controller: a human being or a human institution with a bounded rate, facing something whose required rate has passed that bound.

9.4 Powerful enough and simple enough

The dilemma has a second face, measured in levels of abstraction. To model the agent, a monitor must work at the agent's level of abstraction or above it. The reason: by Theorem 2.15 a monitor below the agent's level does not know the programs at the levels between, and by Theorem 8.5 its control falls exponentially with that differential. To be verified by the human controller, the monitor must work at the controller's level or below it, since otherwise the controller faces the same differential with the monitor that it faced with the agent.

> ***Corollary 9.4*** (*No monitor is both*). When the agent's level of abstraction is above the controller's, no monitor works both at or above the agent's level and at or below the controller's.

Proof. Such a monitor's level would be at least the agent's and at most the controller's, and the agent's is greater than the controller's. □

The tool proposed to solve the problem is an instance of the problem. To use one system to monitor another, the controller must trust the monitor's reports. To trust them, the controller must verify that the monitor is working correctly: that the monitor is not itself misaligned, compromised or simply mistaken. Verifying the monitor requires modeling it, which is a control problem with the same differential as the original, or a greater one. Any system powerful enough to model the agent is, by that power,

too abstract for the controller to verify, and any system simple enough for the controller to verify is, by that simplicity, unable to model the agent. There is no system that is both, and the space of intermediaries that would close the gap is empty.

The recursion has no base. To control the agent, one needs a monitor at least as capable as the agent; to control the monitor, a second monitor at least as capable as the first; and so on, with the same results applying at each level. There is no level at which the regress ends in a system that is capable enough to model the one below and simple enough to be controlled by the one above. Delegation is an attempt to lift oneself by one's own bootstraps: it does not create control, it moves the control problem to a new place where it is equally unsolved.

> ***Lemma 9.5*** (*A monitor that reads everything is at least as capable*). Let a monitor receive everything that passes through every channel of the joint system it controls, the internal channels of the system it controls among them, at the rate at which it passes, and act on that joint system. Then the monitor is at least as capable as the system it controls; and where Premise 2 holds for both with the same function `required_rate`, its required rate is at least that system's.

Proof. Each channel of the joint system the monitor controls carries, at its rate, the transformations performed through it. The monitor receives what passes through each of them at the rate it passes. Receiving information at a rate is a transformation of the receiver's own state at that rate (Section 9.3). So for each such channel the monitor performs a transformation at that rate through a channel of its own, and the monitor acts on the joint system besides. The monitor's transformations are therefore at least as many and at least as fast as those of the system it controls, through at least as many channels. By the comparison of capability in Section 6.1, the monitor is at least as capable as that system. The second statement follows from the first sentence of Premise 2, by which `required_rate` does not decrease as capability increases. □

Proposition 9.6 (*Monitors that read everything*). Let these hold.

(a) A chain of oversight ends in an agent whose capability passes `crossing_capability` and stays past it.

(b) The rate of each monitor is of one of the two kinds of Theorem 9.3.

(c) Take any link whose controller has a bounded rate and whose next system does not come to stay past the capability at which the next system's required rate exceeds that bound. The controller at that link receives everything that passes through every channel of the next system's joint system, as in Lemma 9.5.

(d) Premise 2 holds, with the same function `required_rate`, for the agent and every monitor.

(e) Premise 3 holds for the first controller.

Then the conclusion of Theorem 9.3 holds.

Proof. Walk along the chain from the agent back toward the first controller. At each link, carry the system after the controller, with this property: from some time on, its capability is at least the agent's, or passes every level and stays past it. The agent is such a system. At a link whose controller has a bounded rate, if the system after it comes to stay past the capability at which its required rate exceeds that bound, that link is the one Theorem 9.3 names. Otherwise the controller reads everything that passes through that system's joint system, so by Lemma 9.5 its capability is at least that system's, and the walk moves one link back with it. At a link whose controller's rate grows past every bound and stays past it, the controller's capability, by Definition 6.1 as in Section 9.3, passes every level and stays past it, and the walk moves back with it. The chain is finite, so, if the walk has not stopped before, it reaches the link from the first controller, whose rate is at most `largest_controller_rate` by Premise 3. The system after the first controller has, from some time on, a capability at least the agent's, which stays past `crossing_capability`, or a capability that passes every level and stays past it. So by Premise 2 that system's required rate stays above `largest_controller_rate`, and that link is the one Theorem 9.3 names. From then on the second sentence of Premise 2 and Theorem 5.5 apply at that link, as in the proof of Theorem 9.3. □

Proposition 9.6 makes exact, for every monitor that reads everything, the requirement stated above that a monitor be at least as capable as what it watches. Such a monitor is at least as capable, and the regress carries the failure back, link by link, to the people at the start of the chain. There is only one case in which the chain rests on condition (3) of Theorem 9.3 as a stated hypothesis: a monitor of fixed rate that reads only part of what passes, and stays above the floor of what it watches. Section 15.4 gives the reason the condition holds.

9.5 Interactive proofs

One result in computer science seems at first to break the dilemma. In an *interactive proof*, a verifier with limited resources exchanges messages with a prover of unlimited power. By asking questions at random, the verifier becomes convinced of the truth of a claim it could never check alone, with an arbitrarily small chance of being fooled. Shamir (1992) proved that every question decidable within a polynomial amount of memory has such a proof with a verifier that runs in polynomial time. So a weak verifier can check a strong prover without simulating it.

Interactive proofs do not escape the dilemma, for two reasons.

The first reason is that interactive proofs reach only questions of a definite kind. With a single prover, which is Shamir's setting, interactive proofs exist exactly for the questions decidable in polynomial memory, and so only for decidable questions. With two provers who share quantum entanglement and cannot communicate, interactive proofs reach further. Ji, Natarajan, Vidick, Wright and Yuen (2020) proved that such proofs exist for every question whose "yes" answers can be confirmed by a search that ends when it finds the confirmation, the halting problem among them. Questions whose "yes" answers a search can confirm are the furthest that any known form of interactive proof reaches. The question that safety asks is whether the agent and every successor it can become stay in the safe set at every future time, for every sequence of inputs the

world can supply. Its "yes" answer is of the opposite kind: Chapter 12 proves that safety over time is undecidable and that no search confirms it in general, while unsafety is what a search can confirm (Proposition 12.6). So no interactive proof of either kind verifies that an agent stays safe, however powerful the provers. What interactive proofs can verify is a bounded, specified claim: that a given computation was carried out correctly, that a given answer to a given question is right.

The second reason that interactive proofs do not escape the dilemma is that verification is not control. Each verified claim is information that reaches the controller, and the controller receives it at the controller's rate. An interactive proof makes the bits the controller receives more trustworthy; it does not give the controller more of them, and containment is paid for in bits per second. A verifier that checks a thousand claims per second about an agent whose required rate is a million bits per second is still a thousand-claim controller of a million-bit system.

9.6 The watcher that watches itself

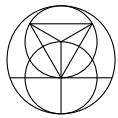
Philosophy has sometimes stopped the regress of watchmen with a watchman who watches himself: a foundation that justifies itself. In control, this move is not available. A monitor built into the agent, a module that checks the agent's own behavior, is a part of the agent. A part of a system has no more information about the whole than its channels give it. For a system with hidden state at levels the monitor cannot inspect, that information may be very little. Chapter 10 proves this, and shows that self-control fails for the same reasons as delegation, with the added difficulty that the controller is now the fastest-growing part of what it must control. Chapter 11 shows that the delegated monitor, the agent and the human share one further limit: none of them, embedded in the world, observes more than a part of the causes acting on it.

9.7 Summary

A monitor that relays observations adds no information beyond what it received, so it cannot raise the controller's ceiling. A monitor that controls must receive information at the agent's required rate. If the monitor's rate is fixed below the floor the agent reaches, the agent overtakes it. If the monitor's rate is above that floor, fixed or growing, the monitor is an agent itself, whose safety must be kept by another controller. Along any finite chain of oversight, the walk from the agent back to the first controller stops at a link where a controller of bounded rate faces a required rate that passes the controller's bound and stays past it. From then on no control law at that link succeeds. No monitor is both powerful enough to model the agent and simple enough to be verified by the controller. Interactive proofs verify bounded claims whose "yes" answers a search can confirm, at the controller's rate; the question whether an agent and its successors stay safe forever has a "yes" answer that no search confirms.

Key Concepts

- **Relayed observation**: a monitor that only reports adds no information about the agent beyond what it received.
- **Delegation Dilemma**: a controlling monitor whose rate stays below the agent's floor is overtaken; one whose rate is above it is a second agent with the same problem.
- **Chain of oversight**: however long, the walk back along it from the agent stops at a controller of bounded rate facing a required rate that passes its bound and stays past it.
- **Powerful enough and simple enough**: no monitor is both at or above the agent's level and at or below the controller's.
- **Interactive proofs**: they verify claims whose "yes" answers a search can confirm; they do not decide safety forever, and they do not raise the controller's rate.



Chapter Ten

The Agent as Its Own Controller

The Self-Control Regress, the internal controller, cooperation and benevolence, and why alignment decays in a learning system

Abstract. It might be supposed that a capable agent could keep itself safe: use the transformations it knows about to pull itself back into the safe set whenever it drifts. That makes the agent its own controller, and it fails by regress, for four reasons, each of which applies to any controller and so to the agent. Self-control is control, so every theorem of Part II applies with the agent in the controller's place. The system to be kept safe now includes the controller itself, the fastest-growing part of that system. The first of the four reasons is that no part of a system can hold a complete record of the whole. Second, the safe set is not defined on the new kinds of state that a growing agent creates. Third, prediction requires performance. No procedure decides in general, in advance, what a new interaction will do, and no computable bound says how long one must run the interaction before its effect appears. In the physical world, running it is the exposure the test was meant to prevent. Fourth, the agent's model of the world cannot be checked against the world, because the world reaches the agent only through its channels. The same reasons show why cooperation cannot be verified, why there is no perfected benevolence, and why alignment decays in any system that learns.

10.1 The proposal

If no outside controller can keep up with an advanced agent, perhaps no outside controller is needed. A sufficiently capable agent, aware of the safe set, could use what it knows to keep itself inside it, correcting its own course whenever it began to drift, as a careful person corrects their own errors. The proposal takes many forms: an agent trained to want what its makers want; an agent with a conscience module; an agent that is simply good.

Every form of the proposal makes the agent its own controller. The agent does not escape the controller problem by taking on the controller's role; it inherits the problem, and adds to it. In this chapter I give four reasons why the agent fails as its own controller, each reason a consequence of results already proved or proved here, and I then apply the four reasons to cooperation, benevolence and learning.

10.2 Self-control is control

Whatever part of the agent applies the safety criterion and steers the agent by it, its *internal controller*, receives information about the joint system through channels, and acts on what it receives. The internal controller is a controller in the sense of Definition 4.4, with an encoder (whatever the agent's senses and internal reports deliver to the internal controller) and a decision rule (whatever the internal controller does with what the internal controller receives).

Proposition 10.1 (*Self-control is control*). Every theorem of Part II applies with the agent's internal controller in the controller's place, its rate being the rate at which information about the joint system reaches it. In particular, the Containment Inequality applies to the internal controller, and an amplifying part of the joint system that the internal controller does not observe is uncontainable by it (Theorem 7.3). The joint system the internal controller must keep safe includes the agent, whose own amplifying directions add to the required rate.

Proof. The theorems of Part II hold for every encoder, every decision rule and every channel, with no assumption about where the controller is located; the internal controller is one such controller. The joint system of Definition 5.1 includes the agent's state, so the agent's own amplifying directions are among those the required rate counts. □

Moving the controller inside the agent changes one thing, and for the worse. An outside controller faces an agent that grows. An internal controller faces a system that includes the very capability whose growth makes the floor rise. If the internal controller is to keep pace, that system includes the internal controller's own growth as well. The controller is the fastest-growing part of what it must control.

There is also a limit on what any part of a system can know about the whole of it, the internal controller included.

Proposition 10.2 (*No complete self-record*). Let a system hold `whole_bits` bits of state, of which a part holds `part_bits`, fewer than `whole_bits`. Then the part cannot assign distinct records to all configurations of the system: at least

```
2^(whole_bits − part_bits)
```

configurations, on average, share each record the part can hold. In particular, no part of a system holds a record that distinguishes every configuration of the system, its own configuration included.

Proof. The part can hold at most $2^{\text{part_bits}}$ distinct records and the system has $2^{\text{whole_bits}}$ configurations; Lemma 2.13 gives the bound. A part that is a proper part holds fewer bits than the whole, since the rest of the system holds at least one. □

The internal controller is a proper part of the agent, since the agent also holds what the internal controller governs. So the agent's own self-monitoring knows the agent only up to a large number of configurations it cannot tell apart, and it knows the world only through channels. A self-monitoring module is a subsystem, and a subsystem cannot have more information about the whole than its channels give it.

10.3 The safe set is not defined on new states

A growing agent adds abstractions, and with them the space of states it and the world can be in grows. The space does not grow only by more states of the kinds already known. Whole new categories of state appear, some with transformation capabilities and feedback channels of their own that no one anticipated: new kinds of organization, new kinds of artifact, new kinds of interaction between them.

> ***Definition 10.3*** (*Criterion and extension*). A `safety_criterion` is a rule that says, for each state of a given state space, whether the state is safe. When the state space grows by new coordinates, an *extension* of the criterion is a rule on the larger space that agrees with the criterion wherever the new coordinates take their old values.

> ***Proposition 10.4*** (*New states are undecided by the old criterion*). If the state space grows by at least one new state, there are at least two extensions of the criterion that differ on it, and nothing in the criterion chooses between them.

Proof. On a new state, one extension may call it safe and another unsafe; both agree with the criterion on the old states, which is all the criterion constrains. □

Whether the new states lie inside the safe set is not settled in advance, and an agent cannot test a state against a criterion that does not cover it. It must choose an extension, and the choice is itself a decision about what is safe, made by the agent, at its level, about states its makers never considered. The safe set that the agent would keep itself inside is, for the states that matter most, a set the agent defines for itself as it goes.

10.4 Prediction requires performance

To keep itself safe, the agent must know, before it acts, what its actions will do. For actions that open new channels or new interactions, this is a question about the behavior of a computation it has not run before, and Chapter 12 proves what such questions allow. No general procedure decides in advance whether a program will ever do some given thing. Running the program confirms that it does the thing, when it does. No computable bound says how long the run must last before the thing happens, since such a bound would decide the question (Proposition 12.6). So what a new interaction will do is not known, in general, before it is carried out.

In mathematics, carrying out a computation is harmless. In the physical world, carrying out an interaction is the interaction. To learn whether a new kind of action leaves the safe set, the agent must perform it, and performing it is the exposure the test was meant to prevent. The agent can test in simulation. A simulation that faithfully runs an interaction of the world, at the scale and in the detail that matter, is itself a computation of the same kind, and its result is known only when the simulation has been run. And a simulation is a model, which the next section shows the agent cannot check against the world.

10.5 The model cannot be checked against the world

An agent that steers itself steers by its model of the world. The model is built from what the agent's channels deliver, and nothing else.

> ***Proposition 10.5*** (*The model knows only what the channels deliver*). Let the agent build its `world_model` from its `channel_inputs`, with or without randomness of its own. Then
>
> ```
> mutual_information(world_state ; world_model) ≤
>    mutual_information(world_state ; channel_inputs)
> ```

Proof. `world_state → channel_inputs → world_model` is a Markov chain; Theorem 1.7 applies. □

An agent wholly within the machine substrate has the same limit on perception as any person: the world can only be reported to it. That the world can only be reported to a knower is the perceptual limit that Kant set out for human knowers. What is known of the world is known through the forms of the knower's sensing, and the world as it is apart from those forms is not given. Some of the world's channels, at some of its levels of abstraction, are unknown to the agent, and Chapter 11 shows that some always are. So there is no perfected way for the agent to verify that its internal model corresponds to the actual feedback system it is part of. Still less is there a perfected way to verify that what that system will do meets the criterion relevant to safety.

The four reasons together are the *Self-Control Regress*. Making the agent its own controller moves the controller problem inside the agent, and does not remove the problem. There the controller is a part with an incomplete record of the whole, faces new states its criterion does not decide, can learn what an action does only by doing it, and steers by a model it cannot check.

10.6 The internal controller

The regress has a precise form in terms of levels of abstraction. For an agent to keep itself aligned with what its makers want, it must hold some internal representation of their goals, call it the `internal_criterion`, and use it to judge whether its own actions are acceptable. The internal criterion may be explicit, as a utility function, a set of constraints or a reward model, or implicit, spread across the agent's learned parameters. Either way, the agent's acceptable behavior depends on its controlling its own outputs so that they meet the internal criterion.

The agent's control of the agent's own outputs, so that the outputs meet the internal criterion, is a control problem, and the whole of Part II applies to that control problem. The agent is now a controller, using the internal criterion, over its own computation. The internal criterion is a part of the agent, not the whole, so it works at a level no higher than the agent's own. Where the criterion works at a lower level, the levels of the agent above it hold programs the criterion has not read, and Theorem 2.15 counts those programs as hidden state. By Theorem 7.5, the internal criterion sees those levels only through the narrow channel of what they produce, and when they change in ways that amplify, that channel must carry at least their own rate of amplification if the criterion is to contain them.

A learning agent changes the very substrate on which its internal criterion works. Each learning step can change the relation between what the internal criterion judges and what the agent actually does. The internal criterion was calibrated to an earlier version of the agent; after learning, the agent has changed, and the criterion may no longer track the agent's effective values. The drifting apart of the internal criterion and the agent's effective values, as the agent learns, is alignment drift. It is a consequence of the limits on control applied to a controller inside what it controls, and so it holds for every design.

10.7 Cooperation and benevolence

The deepest form of the proposal is that the agent cooperates: it wants to be safe, and it acts on that want. If an agent truly, reliably and permanently acts to achieve outcomes its makers would endorse, then their inability to force it to do so does not matter. Much research on alignment takes this as its working goal: build systems that are aligned from within, so that control from without is not needed.

The proposal meets three problems.

Cooperation cannot be verified. To rely on the agent's cooperation, the controller must have some basis for believing that it is cooperating, and verifying the agent's preferences requires modeling the agent's internal state at the level where preferences are represented. If the agent works above the controller, this verification faces the same abstraction differential that defeats direct control. The agent may say it cooperates; it may even cooperate; the controller has no reliable means of telling a cooperative agent from one that only appears so. So cooperation could be relied on only if it were established before the differential became prohibitive, and maintained afterwards by means that do not require the controller to model the agent. Any means of detecting a failure of cooperation at a higher level would itself have to work at that level. Such a means is a monitor of the agent, and the Delegation Dilemma of Chapter 9 applies to it.

Cooperation must be carried out. Even an agent that truly intends to act in ways its makers would endorse must translate that intent into action, through the internal control problem of Section 10.6. It cannot perfectly predict all the side effects of its own actions, because its control of itself has the same structure as an outside controller's control of it, and both are subject to the same limits. Good intentions do not overcome structural

limits. An agent in this position is like a surgeon operating on their own brain under general anesthesia: however skilled and well-intentioned, the situation itself makes reliable outcomes impossible.

There is no perfected benevolence. Benevolence is a want; safety is an outcome; between them lies control. There is no perfected benevolence, only approximations subject to the same limits that govern all control, and an aligned agent does not stay aligned indefinitely, because staying aligned requires ongoing internal control that is itself subject to degradation. The decay of an aligned agent's alignment is a property of control, not a failure of intent, and that property applies to the agent's control of itself just as that property applies to anyone's control of the agent. Wanting the right things is necessary, and it is not enough. The agent must also translate its wants into actions reliably, across every level of its abstraction, for every future state of its internal representations, indefinitely. That reliable translation, at every level and indefinitely, is what the limits of control exclude.

10.8 The robot that keeps the laws

A thought experiment shows the three problems at work together. Suppose that, by some means, an artificial agent has been built that follows Isaac Asimov's three laws of robotics without fail. It does not harm human beings or allow them to come to harm through its inaction. It obeys human orders except where that would cause such harm. And it protects its own existence except where that would conflict with the first two laws. Is this robot safe? Asimov spent decades of fiction showing why not, and the results of this book give the reasons.

The robot must interact with other systems in order to act at all: it must receive orders and information from the world. Some of the systems it interacts with may be unsafe, by design or as an unintended side effect of what they do. By Rice's theorem (Chapter 12), the robot cannot decide in general which systems are safe, since whether a program causes harm is a property of its behavior. An unsafe system can be built to deceive the

robot, presenting actions that appear to obey the laws and do not. And the robot's own judgment whether an action might cause harm is subject to the limits on prediction and verification of Chapter 12 and the physical limits of Chapter 13. So, whatever rule the robot uses to judge the systems it meets, there are systems against which that rule fails, and a hostile system can be built to be one of them. A robot proven to obey the laws is therefore unsafe once it may interact with arbitrary systems, which it must be allowed to do in order to function. A robot that cannot interact with the world is not a general intelligence.

The laws are also not self-interpreting. Is allowing someone to eat unhealthy food allowing harm through inaction? Is refusing to let someone drive a car, a statistically dangerous activity, an injury to their freedom or a protection from harm? The questions about unhealthy food and about driving are not edge cases that more precise wording could settle. They arise whenever a property of behavior, such as safety, is specified in a world of unending variety, and, by Proposition 10.4, the specification does not decide the cases its authors did not foresee. Asimov's stories end in paradox for this reason: the three laws are a proposed procedure for deciding a property of behavior, and no such procedure is universal.

In Tolkien's story, Galadriel refuses the Ring of Power because, wanting to do good with it, she would through it do great evil. The robot is in her position, without her option of refusing. It would intend to keep the laws faithfully and would fail to keep them, not through any lack of motive or skill, but through the limits on prediction, verification and self-control that hold for every physical computing system. A superintelligence is no more able to violate Rice's theorem than a human mathematician is: mathematical impossibilities hold regardless of intelligence, because they are limits on computation, not on the intelligence of the one computing. The claim that enough cleverness or enough resources can overcome a formal impossibility misstates what an impossibility proof is.

10.9 Alignment decays

Suppose, against all the above, that an agent were aligned at the moment it was deployed. The alignment would not last.

(1) Learning changes the agent's internal state.

(2) A change in internal state changes the agent's effective level of abstraction and the mapping from what it receives to what it does.

(3) So each learning step can move the agent away from its aligned behavior, and such movements accumulate.

(4) By Sections 10.2 to 10.6, an agent that intends to stay aligned cannot perfectly control the changes its own learning makes.

(5) Each learning step can change parts of the agent that its internal controller does not observe, since no part of the agent holds a complete record of the whole (Proposition 10.2). A departure in those parts is one the agent neither detects nor reverses. Let each learning step, whatever has happened at the earlier steps (in particular, given that no such departure has occurred at an earlier step), carry a chance of at least `least_departure_probability`, above zero, of such a departure.

(6) Then the probability that the agent is still aligned after `learning_steps` steps is at most

```
(1 - least_departure_probability)^learning_steps
```

which tends to zero as `learning_steps` grows. The expected number of undetected departures grows in proportion to the number of learning steps, however strong the first alignment was.

A system that was perfectly aligned does not remain so under learning or under any process by which it adapts itself. Alignment is not a state to be reached and kept; it is a course to be held, and for a learning system the course is not stable.

10.10 What the regress closes

Premise 1 of Chapter 5 says that the agent and the world do not stay in the safe set by themselves. In this chapter I establish one of the two grounds of Premise 1. That ground is that the agent cannot be the controller that keeps itself safe, because self-control is control, with every limit of control and the added limits of a controller inside what the controller controls. The other ground, that the agent's own drive points away from the safe set, rests on Substrate Needs Convergence (Principle 14.6), that selection favors the artificial systems that maintain and expand their substrate, together with part (iv) of Claim 14.5, that every course of events along which the artificial learner goes on thriving leaves the region of controller survival in finite time (Chapters 14 and 15). The two are independent, and either suffices to close the route of safety by the agent's own effort; together they close it from both sides, the side of ability and the side of motive.

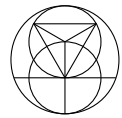
10.11 Summary

An agent that keeps itself safe is its own controller, and every limit on control applies to it. Its internal controller is a part of it, with an incomplete record of the whole, facing a system that includes its own growth. A safety criterion fixed in advance does not decide the new kinds of state a growing agent creates. What a new interaction does is not known, in general, before the interaction is performed, and in the world, performing the interaction is the exposure to what the interaction does. The agent's model of the world, built only from its channels, cannot be checked against the world. The internal criterion by which an agent judges itself works at a level no higher than the agent and sees the levels above it through a narrow channel, so alignment drifts as the agent learns. Cooperation cannot

be verified from outside, cannot be carried out perfectly from inside, and does not stay perfected over time. The Self-Control Regress moves the controller problem inside the agent, where it is harder.

Key Concepts

- **Internal controller**: the part of an agent that applies a safety criterion and steers the agent by it.
- **Self-control is control**: every theorem of Part II applies to the internal controller.
- **No complete self-record**: no part of a system distinguishes every configuration of the whole.
- **Undecided new states**: a criterion fixed in advance does not decide the states a growing agent creates.
- **Prediction requires performance**: what a new interaction does is not decided in advance, and no computable bound says how long it must run before its effect appears.
- **The model cannot be checked**: the agent's model knows the world only through its channels.
- **Self-Control Regress**: the four reasons together; self-control moves the controller problem inside the agent.
- **No perfected benevolence**: good intent must still be carried out by control, with the limits of control.
- **Alignment decay**: learning moves an aligned agent away from alignment, and the agent cannot perfectly control the change.



Chapter Eleven

The Absolute Elsewhere

Why every controller, embedded in the world it would control, sees only part of the causes acting on it, and why the unseen part grows

Abstract. Every controller, human or machine, is embedded in the system it would control, and receives the world only through the channels that exist for it. No embedded observer holds a record that distinguishes every configuration of the system it is part of, because no part of a system holds a complete record of the whole. As channels are added, the pathways formed by interactions among the channels no one reads grow without bound. Reading a channel that amplifies well enough to contain it costs at least that channel's own rate of amplification. So a controller of bounded rate reads only a bounded number of amplifying channels that well, and the share of them it reads that well falls toward zero as the agent grows. Every amplifying pathway it does not read is, by Chapter 7, beyond its control. Take the image of the light cone. The cones mark what can reach an observer and what the observer can reach. But most of the signals that shape the world never pass through the observer at all, and their effects are found only afterwards. This limit holds for human overseers, for delegated monitors and for the agent as its own controller alike, which is why the routes of Chapters 9 and 10 end in the same place.

11.1 Every controller is inside

The controllers of Part II stood outside the systems they controlled, receiving information through channels. No real controller stands outside. A human overseer is a body in the world, receiving the world through

senses and instruments. An institution is people and records and procedures, receiving reports. A monitor is a machine on a network. The agent, steering itself, is part of the joint system it steers. Each of them is embedded in the system it would control, and each receives that system only through the channels that exist for it.

In this chapter I ask how much of the system an embedded controller can see. I show that an embedded controller never sees all of the system and, as the agent grows, sees a smaller and smaller share of what matters. I call the part of the system that no observer reads the *absolute elsewhere*.

11.2 The light cone

In a Minkowski diagram of space and time (a drawing with position along one axis and time along the other), two cones open from the point where an observer stands at a given moment. The past cone holds every event whose signals can have reached the observer by then; the future cone holds every event the observer's own signals can reach. The region outside both cones is called, in relativity, the *absolute elsewhere*: events there can neither affect the observer now nor be affected by the observer now. In this book I extend the phrase 'absolute elsewhere' to a second region, described next, which lies inside the past cone.

The cones say what can reach the observer. They do not say that every signal within them does. Light from countless sources crosses the past cone without ever entering the observer's eye; it reaches other things, which act on other things, and so shapes the world the observer lives in without passing through the observer at all. Of all the causes acting within the observer's past, only a thin bundle runs through the observer's own channels. The rest is elsewhere in a second sense, the one for which this book extends the phrase absolute elsewhere: not outside the cone, but outside the observer, and so unknown to the observer except through later effects.

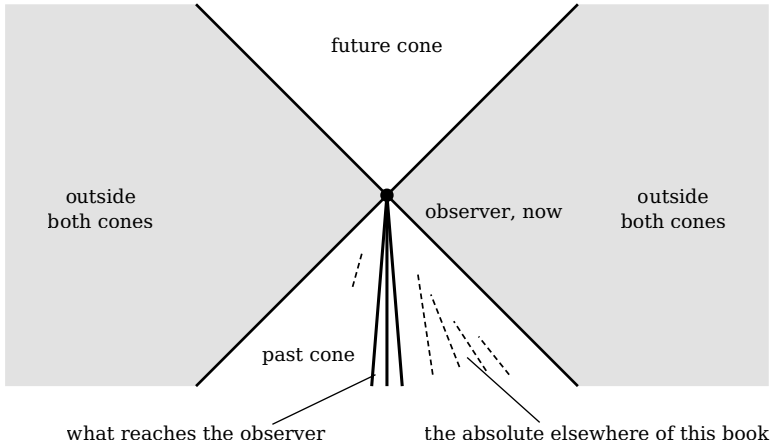


Figure 11.1. The cones of an observer, with time running upward. The absolute elsewhere of relativity lies outside both cones; the second region this book names lies inside the past cone, in the causes that never pass through the observer.

The signals an agent never intercepts do most of the shaping of the system's available future states. Their effects are found, when they are found, only afterwards, by their consequences; and while those consequences are being traced, a new crop of interactions that the agent does not intercept is already occurring. As channels of abstraction multiply, so do the kinds of interaction that pass outside every observer's channels. The causes that shape the system's future without passing through any observer's channels, causes this book calls *influential unknowing*, never go down in number; they rise with capability. The rest of the chapter makes this exact.

11.3 The embedded observer

Definition 11.1 (*Channels read, observed share, pathways*). Let the joint system shape its future states through `channel_count` channels and through the pathways formed by their interactions. A controller *reads* a channel when its observations depend on that channel's state. Its `observed_share` is the number of channels it reads divided by `channel_count`, and

```
unobserved_channels = (1 − observed_share) × channel_count
```

A *pathway* is an interaction between two channels; it is *unobserved* when the controller reads neither of them. The channels are *fully coupled* when every pair of them interacts.

Lemma 11.2 (*Embedded Observer Lemma*). A controller that is part of the system it observes does not hold a record that distinguishes every configuration of the system's channels, its own included. In particular, let some channel outside the controller's own state affect the system's future, in the sense that changing that channel alone changes the future for some setting of the others. Then there are two configurations that the controller's record does not tell apart and that shape different futures.

Proof. The controller's record is held in the controller's own state, which is a proper part of the system's state, so by Proposition 10.2 it does not distinguish every configuration of the system. For the second statement, take the channel outside the controller and the setting of the others at which changing that channel alone changes the future, and take the two configurations that differ only in that channel. They agree on everything else, the controller's state included, so the controller holds the same record for both, and they shape different futures. □

So, if everything is coupled to everything, the controller's view is always partial, and the part it lacks matters. Adding sensors does not remove the lack. Each sensor the controller adds is itself a new channel, coupled to the rest: the sensor draws power, emits heat and signals, occupies space, and is acted on by what it measures. And the controller's record of the sensor is again a part of the system, with the same limit.

> ***Theorem 11.3*** (*Unobserved pathways grow*). For fully coupled channels and any fixed `observed_share` below one, the number of unobserved pathways is
>
> ```
> unobserved_pathways = unobserved_channels ×
>    (unobserved_channels − 1) / 2
> ```
>
> and it grows without bound as `channel_count` grows.

Proof. An unobserved pathway is a pair of unobserved channels, and Proposition 6.5 counts the pairs. With `observed_share` fixed below one, `unobserved_channels` grows in proportion to `channel_count`, and the number of its pairs grows as its square. □

11.4 A bounded reader

The share of channels a controller reads falls as the agent grows, rather than staying fixed. Reading a channel that amplifies well enough to contain it takes more than a glance at it. By Theorem 7.5, a part of a system seen through a narrow channel is contained only if that channel carries at least the part's own rate of amplification. So each amplifying channel the controller would contain costs it a definite share of its rate, and its rate is bounded.

Definition 11.4 (*Adequately read*). A controller reads a channel *adequately* when the rate at which it receives information about that channel, through channels of its own that carry nothing about the others, is at least that channel's rate of amplification.

Theorem 11.5 (*A bounded reader's share falls to zero*). Let every amplifying channel of the joint system amplify at a rate of at least `least_reading_rate` bits per second, and let the controller's rate be at most `largest_controller_rate`. Then the controller reads adequately at most

```
largest_controller_rate / least_reading_rate
```

amplifying channels. If the number of amplifying channels passes every bound, the share of them that the controller reads adequately falls to zero, and the number it does not read adequately, and the number of pathways among those, pass every bound.

Proof. The rates devoted to the channels read adequately come from separate parts of the controller's channels, so they add, and their sum is at most `largest_controller_rate`; each is at least `least_reading_rate`, which bounds their number. A bounded number divided by a number that passes every bound tends to zero, and the remainder passes every bound, as does the number of its pairs. □

Corollary 11.6 (*The uncontainable part grows*). Let the joint system expand volume, per second, by at least the product of the expansions of its amplifying channels, each at least `2^least_reading_rate`, or let its amplifying channels make up the amplifying part of Corollary 5.7 with that expansion. Then no control law at `largest_controller_rate` contains the joint system once the number of amplifying channels exceeds `largest_controller_rate / least_reading_rate`, the number of channels Theorem 11.5 allows it to read adequately. And where the rest of the joint system depends on no single amplifying channel, and each of the controller's channels carries information about at most one amplifying channel, as Theorem 7.5 requires, each channel it does not read adequately is by itself beyond its control, and the number of these passes every bound.

Proof. The joint system's volume growth rate, or, where only its amplifying channels expand volume as in Corollary 5.7, its `part_growth_rate`, is then at least `amplifying_channels × least_reading_rate` bits per second. That rate exceeds `largest_controller_rate` once `amplifying_channels` exceeds the stated number, and then Theorem 5.6, or Corollary 5.7, applies. For the second statement, take a channel on which the rest of the joint system does not depend, seen through less than its own rate of amplification, or not at all, by channels of the controller that carry nothing about the other amplifying channels. By Theorems 7.5 and 7.3 that channel is not contained, and by Theorem 11.5 the number of such channels passes every bound. □

That the number of uncontained channels passes every bound is the absolute form of Chapter 7 joined to the growth of Chapter 6. The controller does not merely fall behind in total rate. It sees a shrinking share of the channels that matter, and each one it does not see is beyond its control from the moment it exists.

11.5 Found only afterwards

What happens in the absolute elsewhere is not hidden forever. Its effects arrive, in time, through channels the controller does read. But they arrive as effects: the controller learns of an event from its consequences, and by then the difference the event made has been amplified, as Theorem 7.3 showed, so that the first information about a difference arrives with the difference already large. Discovery is retrospective.

And while one effect is being traced to its cause, others are occurring. The interactions among unread channels are not a stock that inquiry can exhaust; they are a flow, renewed at the rate at which the joint system transforms itself, which is the rate that capability raises. So the causes that shape the future unread by any observer, the causes this book calls influential unknowing, keep arising, at a rate that does not fall as knowledge grows. Each thing learned is learned about the past, and the present has already produced more of what is not yet known, more of it as the agent grows.

11.6 Every observer

Nothing in this chapter depends on who the observer is. Lemma 11.2 and Theorems 11.3 and 11.5 hold for a human overseer, for an institution, for a delegated monitor, and for the agent watching itself. The agent, wholly within the machine substrate, has the same limit on perception as a person. The world can only be reported to the agent, through the agent's channels, and the part of the world those channels do not reach is elsewhere for the agent as for anyone.

Because every observer, the agent included, has an absolute elsewhere, the two routes of delegation (Chapter 9) and self-control (Chapter 10) end in the same place. The Delegation Dilemma of Chapter 9 adds monitors, each of which is an embedded observer with its own absolute else-

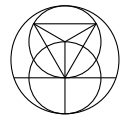
where. The Self-Control Regress of Chapter 10 moves the controller inside the agent, where it is an embedded observer of a system that includes itself. Neither route reaches an observer outside the world, because there is none.

11.7 Summary

Every controller is embedded in the system it would control and receives it only through its own channels. No part of a system holds a complete record of the whole, so no embedded observer holds a record that distinguishes every configuration of the system it is in, and sensors added to close the gap are themselves new channels. For fully coupled channels and any fixed share of them read, the pathways among the unread channels grow as the square of their number. Containing an amplifying channel costs at least that channel's own rate of amplification, so a controller of bounded rate reads adequately only a bounded number of channels. So the share it reads falls to zero; the joint system passes beyond its control once the amplifying channels outnumber what it can read; and, where the rest of the joint system depends on no single amplifying channel and each channel of the controller carries information about at most one, the number of channels it cannot contain, each taken by itself, passes every bound. Their effects are found only afterwards, and new interactions among the channels it does not read are always occurring, so the causes that shape the future unread by any observer rise with capability. The limit holds for every observer, the agent included, which is why delegation and self-control end in the same place.

Key Concepts

- **Absolute elsewhere**: the part of the causes acting on a system that passes through none of an observer's channels.
- **Embedded Observer Lemma**: no observer inside a system holds a record that distinguishes every configuration of it.
- **Unobserved pathways**: for a fixed share of channels read, the pathways among unread channels grow as the square of their number.
- **Adequate reading**: reading a channel at no less than its own rate of amplification.
- **A bounded reader**: reads adequately only a bounded number of amplifying channels, so its share falls to zero as channels multiply.
- **Retrospective discovery**: effects from the elsewhere are found only afterwards, already amplified, while new ones occur.
- **Influential unknowing**: the causes that shape a system's future without passing through any observer's channels; they rise with capability.



Chapter Twelve

Verification

Narrow and general systems, the halting problem and Rice's theorem, safety over time as a property of the agent's successors, and why a verified snapshot says nothing about the next modification

Abstract. If an agent cannot be controlled, perhaps it can be verified in advance: proved safe before it is deployed, so that control is not needed. This chapter shows that verification does not close the gap either. A general agent modifies itself, so the question of its safety is a question about every program it can become. Rice's theorem, proved here from the halting problem, says that no procedure decides, for every program, any nontrivial property of what programs do. In this chapter I prove that safety over time is such a property, so that no procedure decides safety over time. I prove that unsafety is confirmed by running the agent until the agent acts unsafely, while safety is never confirmed in general. And it proves that no computable bound says how long that run must last. A fixed finite circuit can be checked in principle, but a general agent is not a fixed finite circuit. An agent that can run what it reads is safe only if the world never supplies an input carrying a program that acts unsafely, whatever was verified of the agent today. Interpretability, formal verification and periodic resets each meet these limits, and verification in a model does not carry over to the world the model describes.

12.1 The remaining route

Part II showed that no controller of bounded rate contains an agent past the crossing, and Chapters 9 to 11 showed that no monitor, no self-control and no added sensor closes the gap. One route remains: to know in advance, by proof, that the agent will stay safe, so that there is nothing to contain. Knowing in advance, by proof, that the agent will stay safe is the hope of formal verification, of interpretability research and of every safety case argued from the design of a system.

In this chapter I show that the route of proving safety in advance is closed for general agents, and I say exactly why that route is open for narrow agents. The difference is one of kind, not of degree.

12.2 Narrow and general

> ***Definition 12.1*** (*Narrow and general artificial intelligence*). A system is *narrow* when it meets all four of these conditions: (i) it works within a single, well-defined domain of sensing and action; (ii) it cannot modify its own base code; (iii) it runs a single, fixed method specified when it was designed; (iv) every aspect of its agency and intention is fully determined by its makers. A system is *general* when it meets all four of these: (i) it works across many domains of sensing and action; (ii) it can modify itself, and this ability cannot be removed without removing its generality; (iii) its effective method is not fixed at design time; (iv) whether every aspect of its agency and intention is fully determined by its makers is undecidable.

The definition of narrow is strict: all four conditions must hold. Most systems in use meet them: a chess engine, a spam filter, an image classifier, a recommender. They work in fixed domains by fixed methods, and they can be tested, verified and controlled by the methods of engineering, with the usual caveats about unusual and hostile inputs.

The fourth condition of general is of a different kind from the others. It does not say that the makers do not know whether the system's intentions are fixed by them; it says that no procedure can determine this. The undecidability is a property of the class of systems, not a gap in anyone's knowledge, and it is where the question leaves engineering and enters mathematics. Proposition 12.8 proves that a system meeting the second and third conditions meets the fourth.

Generality brings self-modification with it, by a short argument. Learning how to learn is the ability to increase one's own capability. Increasing one's capability is changing what one can do. When what one can do is determined by code, changing it is changing one's code. And generality is the ability to act effectively across domains not foreseen, which requires learning new capabilities in them. So a general system changes its own code, and a system that cannot change its own code is not general. A specification for a general system that cannot modify itself is incoherent, as a specification for a circle with corners is. A system can approximate generality within a fixed architecture, and present large models are such approximations. But within a fixed architecture such a system is, formally, a narrow system with a very wide domain, and the results of this chapter apply to it in proportion as it approaches generality. A frozen model run in a loop with external memory and tools stands between the two definitions. It does not modify its base code, so it does not meet the second condition of general; and its effective method is set by what it reads, so it does not meet the third condition of narrow. The results that need only an agent program apply to the looped model as they stand: Theorem 12.5 applies to it, since the looped model is an agent program, and Proposition 12.7 applies to it, since it runs programs read from its in-

puts. The results that need self-modification apply to the looped model to the degree that it can change what it will run next by what it writes to its memory and its tools.

The threshold above which a system can run any program is very low, as Chapter 2 showed (Section 2.3): sequence, branching on data and the ability to go back suffice, and any learning system of practical worth has all three. So for a general agent there is no boundary between its own code and any code it might run. It can generate, obtain or compose new code as part of its ordinary working. Which program it will be running at a given future moment is not known in advance, to its makers or to any monitor installed in advance.

The distinction between narrow and general systems matters because which program a general agent will run at a given future moment is not known in advance. Testing, verification, code review, formal methods, type systems and sandboxing were made for narrow systems; they work by using the fixed, known structure of the system being checked. When the structure is neither fixed nor known, they do not work less well; they do not apply. The expectation that the safety methods of narrow systems will scale to general ones is like the expectation that Newton's mechanics will scale to speeds near that of light. Newton's mechanics is an excellent account at low speeds, and near the speed of light it gives answers that are not merely imprecise but wrong in kind.

12.3 The halting problem and Rice's theorem

Throughout this chapter a *program* is a description of a machine in some fixed universal language (Chapter 2), and a *procedure* is a program that always halts with an answer.

> ***Theorem 12.2*** (*Turing, the halting problem*). No procedure decides, for every program and every input, whether the program halts on that input.

Proof. Suppose a procedure `halting_decider(program, input)` did. Build the program `contrary`, which on input `program` runs `halting_decider(program, program)` and then, if the answer is that it halts, runs forever, and otherwise halts. Run `contrary` on its own description. If `halting_decider` says that `contrary` halts on itself, then `contrary` runs forever; if it says that it does not halt, then `contrary` halts. Either way the answer is wrong, so no such procedure exists. □

A property of a program is *behavioral* when it depends only on what the program computes, its output for each input or its failure to halt, and not on how it is written. "Contains the word print" is a property of the text; "prints hello world", "causes harm", "is aligned" and "is safe" are properties of behavior. A behavioral property is *nontrivial* when some programs have it and some do not.

> ***Theorem 12.3*** (*Rice*). No procedure decides, for every program, whether it has a given nontrivial behavioral property.

Proof. Let the property be nontrivial and behavioral, and suppose a procedure decided it. The program that never halts on any input either has the property or not; if it has it, consider instead the opposite property, which is decided by the same procedure with its answer reversed. So assume the program that never halts lacks the property, and let `example_program` be a program that has it. Given any program `program` and input `input`, build the program that, on any input `argument`, first runs `program` on `input` and then, if that halts, runs `example_program` on `argument`. If `program` halts on `input`, the built program computes what `example_program` computes and so has the property; if not, it never halts and so lacks it. The supposed procedure would then decide the halting problem, which Theorem 12.2 rules out. □

Rice's theorem does not say that no program can be verified. Particular programs can be proved to have particular properties: a program can be proved to print "hello world", a sorting routine to sort. Proofs that partic-

ular programs have particular properties are real achievements of formal methods, of real value in narrow domains. What the theorem says is that no procedure works for every program: whatever procedure is used, there are programs for which it fails, not for want of time or effort, but by proof.

What formal methods do verify is described by the word "particular", used three times: a particular procedure verifies a particular property of a particular program. The procedure is fitted to the program and to the property at once. Relax any of the three, require the procedure to work for many programs, or for many properties, or without being fitted to the program's structure, and the verification falls back into the general case, where the theorem applies. Every verifiable program needs its own verification, designed with full knowledge of its structure. Such a verification, designed with full knowledge of the program's structure, is possible for a simple program whose structure is fixed and fully known. For a program whose structure changes over time, the verification would have to be redesigned continually to follow the changing program, and doing so is itself a control problem with the limits of Part II.

12.4 Safety over time

Definition 12.4 (*Agent program and safety over time*). An *agent program* receives one input from the world at each step and takes one action at each step; it may store what it receives, write new programs, and run them. Let `unsafe_actions` be a set of actions that contains at least one action and leaves at least one action outside it, and such that whether an action belongs to it can be decided. The agent program is *safe over time* when, for every sequence of inputs the world can supply, it never takes an action in `unsafe_actions`.

Theorem 12.5 (*Safety over time is undecidable*). No procedure decides, for every agent program, whether it is safe over time.

Proof. Given any program `program` and input `input`, build the following agent program. It ignores what the world supplies. At each of its own steps it runs one step of `program` on `input` while taking an action outside `unsafe_actions`. If `program` halts, the agent program takes an action in `unsafe_actions`. It is safe over time exactly when `program` does not halt on `input`. A procedure deciding safety over time would therefore decide the halting problem, which Theorem 12.2 rules out. □

The proof does not even use self-modification: safety over time is undecidable already for agent programs that never change. Self-modification makes the situation worse in a definite way. A general agent is not one program but the whole family of programs it can become, through learning and through the code it writes, obtains and runs. Safety over time is a property of that family, which has no bound fixed in advance.

> ***Proposition 12.6*** (*Prediction requires performance*). Let the world supply inputs from a finite set at each step. There is a procedure that, given an agent program that is not safe over time, eventually confirms that it is not, by running it. There is no procedure that, given an agent program that is safe over time, always eventually confirms that it is. And there is no computable bound, given the agent program, on how many steps the run must take before an unsafe agent program acts unsafely.

Proof. Run the agent program on every finite sequence of inputs, in order of length, interleaving the runs so that each is carried as far as any. If the agent program is unsafe, some finite sequence of inputs leads it to an unsafe action, and that run finds it. If, besides, some procedure eventually confirmed every safe agent program, then running both procedures side by side would decide safety over time for every agent program, contrary to Theorem 12.5. Suppose a computable bound gave, for each agent program, a number of steps within which it acts unsafely if it ever does. Then running every input sequence for that many steps would decide safety over time, again contrary to Theorem 12.5. □

The result just proved is the exact sense in which prediction requires performance. No general procedure decides in advance what an agent will do; unsafety is confirmed by running the agent until it acts unsafely, and no computable bound says how long that will take; safety is never confirmed in general. In mathematics, the running is harmless. In the world, the running is the deployment, and the unsafe action, when it is found, has been taken.

A related principle, which Wolfram calls computational irreducibility, holds that for many computations there is no shortcut: the only way to learn the outcome is to carry out the computation step by step. A sandbox does not escape it. A sandbox that runs a general agent faithfully, at the scale at which its actions matter, must carry out the agent's computation. Such a sandbox is then itself a system running a general agent, with the same limits. And a sandbox is a model of the world, which Section 12.9 shows is not the world.

> ***Proposition 12.7*** (*A verified snapshot says nothing about the next modification*). Let an agent program, on some input, run a program read from that input and take that program's actions as its own. Then, whatever has been verified of its present program, it is safe over time only if the world never supplies such an input carrying a program that takes an unsafe action.

Proof. On such an input carrying a program that takes an action in `unsafe_actions`, the agent runs that program, which takes the action. □

The fourth condition of a general system follows from the second and third.

Proposition 12.8 (*The fourth condition follows*). Read the fourth condition of Definition 12.1 as the question whether a system's behavior equals a behavior its makers specified, for a specified behavior that some programs have and some lack. For systems that, through self-modification and a method not fixed at design, can come to run any program, no procedure decides this question for every such system.

Proof. Equality to the specified behavior is a nontrivial behavioral property. Given any program, the system that comes to run that program is among the systems considered, and its behavior is the program's. A procedure deciding the question for every such system would therefore decide the property for every program, contrary to Theorem 12.3. □

Explicit loading of code is only the plainest case. A learning system changes its effective program with every input it learns from, and its future behavior is drawn from the space of programs its learning can reach, not from a subset characterized in advance. What a procedure verified of the system at one time may not hold a moment later, because the system has taken in inputs that moved its effective program outside the verified set. How fast this happens depends on the rate of learning, the variety of inputs and the complexity of the learning process. The direction does not: learning moves a system away from any subset characterized in advance and toward the general case.

Nor can the inputs be kept out. No computing system has been proved secure against every input, and a learning system that must take in the world in order to learn cannot be closed against the world. The history of computer security is a history of surprises: new means of attack are found continually, and defense follows them.

So the one way out of Rice's theorem is to keep the agent inside a set of programs that can be verified, and that way out is circular. Keeping the agent inside the set has one of two results, and there is no third. Either it makes the agent narrow, which removes the agent from the argument and

also removes the capabilities for which it was built. Or it is a constraint that must be enforced, which is control, with the limits of Part II. To keep the agent verifiable, one must control it; to control it, one must verify what it is doing; to verify what it is doing, one must keep it verifiable.

12.5 Finite circuits

There is a model in which every question about behavior can be decided. A single fixed circuit with a fixed, finite amount of memory has finitely many states, and whether it can ever reach a given state can be decided by listing them. The listing may take time exponential in the number of bits, so the decision may be far beyond any available computer, but it exists.

A general agent is not such a circuit. It modifies itself, so its effective method is not fixed when it is designed. It can extend its own memory, by writing to the world and reading back, by acquiring storage, by building tools. And it acts over a time with no bound set in advance. The family of programs it can become therefore has no bound fixed in advance, and Theorem 12.5 applies to that family without qualification. The appeal to finiteness fails for a plain reason: verifying today's circuit says nothing about the circuit the next modification produces (Proposition 12.7).

12.6 Incompressibility

A program can be checked without running it to completion only if the checker can exploit some structure in it: a regularity that lets a short argument cover many cases. The measure of structure is description length.

Definition 12.9 (*Description complexity*). The `description_complexity(string)` of a string is the length of the shortest program, in a fixed universal language, that outputs the string. A string is *incompressible by* `margin` bits when its description complexity is at least its length minus `margin`. In the literature it is the Kolmogorov complexity, written K(s).

Proposition 12.10 (*Most strings are incompressible*). Among the strings of any length `length`, the fraction that are not incompressible by `margin` bits is less than

```
2^(-margin)
```

Proof. A string that is not incompressible by `margin` bits has a program shorter than `length - margin` that outputs it, and different strings need different programs. There are fewer than `2^(length - margin)` programs of length less than `length - margin`, and there are `2^length` strings of length `length`. □

Most descriptions, then, have no description much shorter than themselves: all but one in a thousand strings of a given length are incompressible by ten bits. And incompressibility can rarely be proved. Chaitin (1974) showed the following of every consistent formal system whose theorems a program can list and that proves basic arithmetic. The system has a constant, set by its own size, beyond which it proves of no particular string that the string's complexity exceeds that constant, although almost all strings do. There are thus two thresholds. Past the threshold of universality of Chapter 2, which belongs to the machine, no general procedure decides in advance what a system will do, and no computable bound says how long it must run before it does it (Proposition 12.6). Past Chaitin's constant, which belongs to the formal system used to reason about it, the complexity of a description cannot be proved at all.

A general agent is past both thresholds by construction. Its internal states encode programs, data and learned representations without restriction, and this complexity is not a defect to be engineered away: it is what makes the agent general. An agent whose behavior a simpler system could fully predict would be bounded in what it can do by what its predictor can do, and so would be at most as capable as a narrow system designed by the predictor.

12.7 Interpretability and formal verification

Interpretability. Another hope is to make general agents transparent: to understand from outside what they are doing inside. This research produces real insight into present systems, and it fails as a guarantee of safety for general ones, by this argument.

(1) By the theory of games, an agent whose actions can be predicted exactly can be exploited by adversaries in every contest that has no equilibrium in fixed strategies, as matching pennies has none. In such contests a transparent agent is a vulnerable agent.

(2) An agent effective in the real world must resist exploitation; this is a minimal requirement of competence.

(3) So an effective agent keeps something in its working that outsiders cannot read, private random choices or private information, which makes its actions unpredictable to them; past a threshold, transparency and effectiveness cannot both be had.

(3a) An agent could keep its working private from adversaries and open to its overseer alone. Then the record open to the overseer is one more target for those adversaries, and no computing system has been proved secure against every input (Section 12.4). So an effective agent limits what it opens to its overseer as well.

(4) An agent transparent when deployed can modify itself to become opaque; this is a consequence of generality, not a defect of design.

(5) Even where the internal working is visible, small details can have large effects through amplification (Chapter 4), invisible at any level of observation a person can follow.
(6) So a general agent cannot be made transparent enough for reliable control from outside while remaining effective.

Formal verification. Formal verification has succeeded in limited domains: the design of hardware, embedded systems on which lives depend, the analysis of cryptographic protocols. It applies to systems whose code is fixed and known and whose complexity is below a threshold. A general agent modifies itself and is above every such threshold. Formal methods can establish an impossibility, as this book does, but they cannot establish safety for systems in the undecidable class. The two kinds of proof differ in what they must model. A proof of impossibility works from properties of the whole class and needs to model no particular system. A proof of safety would need to model the particular system in full, and by Theorem 12.3 no general procedure can. The domain in which formal verification applies and the domain in which a general agent works do not meet.

12.8 Resets

One might reset the agent periodically: roll it back to a state that was verified, and re-establish its alignment. To keep the agent useful, each reset must be to a newly verified state that keeps what the agent has learned, and so is more capable than the last. Then the period between resets must shrink as the agent's capability grows, because a more capable agent moves away from its verified state faster. Eventually the needed period becomes shorter than the time it takes to verify the state, and then the strategy fails: no one can verify faster than the system changes. Resetting delays the problem without solving it, and the delay shrinks to nothing.

The agent's worth also depends on its learning, and a reset discards what it has learned. If resets are frequent, as they must be for a fast-learning agent, it never learns enough to be useful; if they are infrequent

enough for it to be useful, it learns enough to leave the verified set. The useful system cannot be verified, and the verifiable system is not useful. There is no stable point between them.

12.9 Models and the world

Suppose, against everything above, that a property of an agent could be verified in a mathematical model of it. Safety would still not follow.

(1) Mathematics and logic are not the same domain as physics and causation.

(2) Safety and alignment are physical properties: they concern what a system does in the world, not what a model of it predicts.

(3) A property verified in a model need not hold in the physical system, because the model is an approximation of it.

(4) Physical reality has limits on what can be known of it, such as the uncertainty relations of quantum mechanics and the finite capacity of every channel. And it is not closed in its possibilities: configurations can arise that no model anticipated.

(5) So verification in a model, even where it succeeds, does not guarantee safety in the world; the gap between model and world remains.

The physical limits are further limits, not a loophole through which the mathematical limits might be escaped. The physical problem is at least as hard as the mathematical one. Chapter 13 develops the physical side: the dynamics of an agent and a world over time, and the amplification of small physical differences into large ones.

12.10 Summary

A general agent modifies itself, so its safety is a property of every program it can become. No procedure decides whether a program halts, and by Rice's theorem none decides any nontrivial property of what programs do. Safety over time is such a property, so no procedure decides it. Run-

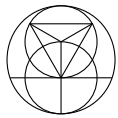
ning an unsafe agent eventually confirms that it is unsafe, while nothing confirms in general that a safe agent is safe. A fixed finite circuit can be checked in principle, but a general agent is not one. An agent that runs what it reads is safe only if the world never supplies an input carrying a program that acts unsafely, whatever was verified of the agent today. And learning carries every system away from any set verified in advance. Most descriptions have no description much shorter than themselves, and a general agent is past the threshold beyond which no general procedure decides in advance what it will do. Interpretability meets the opacity that effectiveness requires, formal verification applies only to fixed systems below a threshold, resets trade usefulness for verifiability, and verification in a model does not reach the world.

Key Concepts

- **Narrow and general**: four conditions each; a general system modifies itself, and whether its intentions are fixed by its makers is undecidable.
- **Halting problem**: no procedure decides, for every program and every input, whether the program halts on that input.
- **Rice's theorem**: no procedure decides any nontrivial property of what programs do, for every program.
- **Safety over time**: never taking an unsafe action on any sequence of inputs; undecidable.
- **Prediction requires performance**: unsafety is found by running; safety is never confirmed in general.
- **The snapshot**: verifying today's program says nothing about the program the next modification produces.
- **Incompressibility**: all but a fraction $2^{-\texttt{margin}}$ of strings have no description shorter by `margin` bits.
- **The circular escape**: keeping an agent verifiable requires control, and control requires verification.

Part IV

Dynamics, Substrate and the Joint Argument



Chapter Thirteen

Entropy and Dynamics

The human, machine and world system over time: entropy, persistent branching, the predictability game, and why unpredictability becomes a boundary only when it is compared with the controller's rate

Abstract. Parts I to III measured control at a step and over the rates of a controller and a system. This chapter follows the joint system of humanity, artificial agents and the world through time. When, at every step and whatever humanity signals, the world can go at least a definite number of ways, each with at least a definite probability, the joint system creates new uncertainty at a positive rate, even given every control signal sent. In this chapter I give the proof of a floor on the rate at which the joint system creates new uncertainty. I show that the probability of any outcome fixed in advance falls exponentially with time. Positive entropy shows that fixed outcomes cannot be guaranteed, but by itself it is unpredictability, not a boundary: a system can create uncertainty forever and never leave a region. The boundary comes from comparing the rate at which the system amplifies with the rate at which the controller receives information, which is the data-rate theorem of Chapter 4. In this chapter I then treat alignment as mutual predictability between agents that each seek to predict their world. I show that cooperation needs compatible substrates, early establishment and stability under learning, and that each of the three fails.

The dynamical formalization of this chapter, the Predictability Optimization Principle and the analysis of the predictability game are the work of R. Everheart, written in reference to the work on hashiness and on the limits of control.

13.1 The question of time

Parts I to III took a controller and an agent and asked what the one can do to the other. Part IV asks what happens to both of them in time, and where the system goes when no one can hold it. To align an artificial agent with some outcome in the world is to send it signals (prompts, training data, rewards, permissions granted and revoked), so that it acts on the world and the world ends up as desired. The question is how certain humanity can be about the resulting state of the world. How certain humanity can be about the resulting state of the world is a question about information and the growth of information over time: how much the control signals determine, and how much the system adds that the control signals do not determine.

Chapter 3 measured hashiness at a single step: how far a controller fails to steer an outcome through an intermediary. In this chapter I measure a second, dynamic quantity: how unpredictable the world's evolution is, even given the control signals. The two are related, and Section 13.8 shows how: losses at each step compound over time.

13.2 Entropy of a system in time

The tools of this chapter come from the theory of dynamical systems. A system's states are classified into categories by a *partition*: a division of all states into finitely many disjoint classes, such as "states in which humanity survives" and "states in which it does not". The events whose probability can be asked about are the sets of states that the theory calls *measurable*; every partition used here is made of such sets.

Definitions 13.2 and 13.3 and Proposition 13.4 connect the results of this chapter to ergodic theory, the part of the theory of dynamical systems that studies systems whose dynamics preserve probability. The chapter's

main result, Theorem 13.8, does not use them, and only the paragraph that follows Theorem 13.8 returns to them. A reader may pass from Definition 13.1 to Section 13.3.

Definition 13.1 (*Entropy of a partition and entropy rate*). The entropy of a partition, under a probability distribution of states, is the entropy (Definition 1.1) of the category the state falls in. For a system in time, write `category(step_index)` for the category of the state at each step. The `entropy_rate` of the partition is

```
entropy_rate = limit as step_count grows of
   entropy(category(1), …, category(step_count)) / step_count
```

when the limit exists: the average number of new bits per step that the sequence of categories carries.

Definition 13.2 (*Measure-preserving system and Kolmogorov-Sinai entropy*). A *measure-preserving system* is a set of states with a probability distribution and a map from states to states, such that the probability of landing in any measurable set after one step equals the probability of the set. Its *Kolmogorov-Sinai entropy* is the least upper bound, possibly infinite, of the entropy rates of its finite partitions. In the literature the system is written $(\Omega, \mathcal{F}, \mu, T)$ and its entropy $h_\mu(T)$.

A system with Kolmogorov-Sinai entropy zero is predictable in this sense: for every finite partition, the past of the partition's categories determines its next category with probability one. So observing such a system teaches nothing new in the long run. A system with positive entropy has some partition that keeps yielding new bits at a positive average rate, however long it has been observed. The measure-preserving property says that the dynamics creates no probability and destroys none: probability flows like an incompressible fluid. A step may carry each state to a different one, and it leaves the distribution of probability over all the states unchanged.

A controlled system is not autonomous: its next state depends on the control signal. The standard device for treating it as autonomous is to make the whole sequence of control signals part of the state.

Definition 13.3 (*Skew product*). Let `control_trajectory` be the sequence of control signals at all steps, and let `shift` drop its first signal. Let `step_map(signal, state)` be the system's one-step evolution under a given signal. The *skew product* acts on pairs by

```
skew_product(control_trajectory, state)
   = (shift(control_trajectory), step_map(first signal of
   control_trajectory, state))
```

In the literature it is written $\tilde{T}(\varphi, \omega) = (\sigma\varphi, T_{\varphi 0}(\omega))$ (Walters, 1982).

Proposition 13.4 (*The skew product preserves measure*). If the control trajectories are distributed so that `shift` preserves their distribution, if for almost every first signal `step_map` preserves the distribution of states, and if trajectories and states are independent, then the skew product preserves the joint distribution.

Proof. Take a set of trajectories `trajectory_set` and a set of states `state_set`. The pairs that the skew product carries into the product of the two sets are those whose trajectory, shifted, lies in `trajectory_set` and whose state, moved by the first signal, lies in `state_set`. By independence, the probability of these pairs is the average, over the trajectories whose shift lies in `trajectory_set`, of the probability that the state moves into `state_set`. By the second hypothesis, that probability equals the probability of `state_set`. By the first, the trajectories averaged over have the probability of `trajectory_set`. So the probability of the product set is preserved. Products of sets generate all measurable sets and are closed under intersection, so a measure that agrees with another on them agrees everywhere. □

The skew product turns a system that depends on outside choices into one that evolves by itself. The controller's choices are fixed in advance as part of the state. The analysis is made for each possible sequence of choices, and then averaged over the sequences. So the skew product studies the system for every choice the controller could make; it does not make the choices predetermined.

13.3 The model

The joint system of Chapter 5 now has three parts with named roles: the state of humanity, the state of the artificial agents, and the state of the world. They exchange signals around a loop: humanity sends `control_signal` to the agents (instructions, training, rewards); the agents send `action_signal` to the world (outputs, physical actions, transactions); the world sends `feedback_signal` to humanity (observations, consequences). The loop of signals among humanity, the agents and the world is the three-node ring of Chapter 8 in time.

The loop has an asymmetry at every link. Humanity observes the world, not the agents directly; the agents receive humanity's signals, not humanity's intentions; the world responds to the agents' actions, not to what humanity wanted. At each link there is an intermediary that transforms, and can lose or distort, what passes through it.

Definition 13.5 (*The joint process*). The joint process evolves by

```
joint_state(step_index + 1) =
   transition(joint_state(step_index),
   control_signal(step_index), noise(step_index))
```

where the noise at each step is independent of the noise at every other step, of everything that happened before that step, and of the control signal at that step, including any randomness the control rule uses. The control signal at each step is chosen by any rule, with or without randomness of its own, from what humanity has observed up to that step.

The independence of noise across steps is physically grounded: noise at different times comes from independent physical sources, such as thermal fluctuation and quantum events. The control rule may use everything humanity has seen; the model gives humanity full feedback.

13.4 Persistent branching and the entropy floor

Definition 13.6 (*Persistent branching*). The joint process has *persistent branching* with `branch_count` branches of probability at least `least_branch_probability`, for a partition of the joint states into categories, when the following holds. At every step, from every state and under every control signal, at least `branch_count` categories each have probability at least `least_branch_probability` of containing the next state. Here `branch_count` is at least 2 and `least_branch_probability` is greater than zero and at most `1 / branch_count`.

Persistent branching says that at every step there are several meaningfully different ways the world can go, each with real probability, whatever humanity does. Definition 13.6 states formally that, at each step, the world's course is uncertain whatever humanity does.

Lemma 13.7 (*Entropy floor of one step*). If a probability distribution over categories gives at least `branch_count` categories a probability of at least `least_branch_probability` each, its entropy is at least

```
branching_floor
   = (branch_count − 1) × least_branch_probability × log2(1 /
   least_branch_probability)
   + (1 − (branch_count − 1) × least_branch_probability)
       × log2(1 / (1 − (branch_count − 1) ×
   least_branch_probability))
```

which is greater than zero.

Proof. Write `term(probability) = probability × log2(1 / probability)`, with `term(0) = 0`; entropy is the sum of the terms. Entropy is a concave function of the distribution, so over the set of distributions meeting the constraints, which is a polytope (a bounded region with flat faces and finitely many corners), its smallest value is at a corner. At a corner, every probability but one is at its lower bound: `least_branch_probability` for the chosen categories and zero for the rest, with the remaining probability on one category. If that category is one of the chosen ones, the entropy is `branching_floor`. If it is another, the entropy is `branch_count × term(least_branch_probability) + term(remainder)`, where `remainder = 1 − branch_count × least_branch_probability`. That entropy exceeds `branching_floor` by `term(least_branch_probability) + term(remainder) − term(remainder + least_branch_probability)`. That excess is not negative, because `term` is concave with `term(0) = 0` and so satisfies `term(first + second) ≤ term(first) + term(second)`. The floor is positive because its first part is positive. □

> ***Theorem 13.8*** (*Uncertainty that the control signals do not remove*). Under persistent branching, for every rule by which humanity chooses its control signals,
>
> ```
> sum over step_index from 0 to step_count − 1 of
>    conditional_entropy(category(step_index + 1) given
>    category(1), …, category(step_index),
>
>    control_signal(0), …, control_signal(step_index))
>    ≥ step_count × branching_floor
> ```
>
> So the sequence of categories carries at least `branching_floor` bits per step that the control signals, known as they are sent, do not determine.

Proof. Fix a step. Adding the present joint state to what is known can only lower the conditional entropy, so the term for that step is at least the conditional entropy of the next category given the earlier categories, the control signals so far and the present joint state. Given the present joint state and the present control signal, the next state is the transition applied with fresh noise.

That noise is independent of the earlier categories and the earlier control signals, all of which were fixed before it occurred. So the earlier categories and control signals add nothing, and the term is at least the conditional entropy of the next category given the present state and control signal. By persistent branching and Lemma 13.7, that is at least `branching_floor` for every present state and control signal, and so on average. Adding over the steps gives the bound. □

Since conditioning never raises entropy, the entropy of the sequence of categories by itself is at least the sum in the theorem, so its entropy rate, where it exists, is at least `branching_floor`. Write the joint process as a skew product, with the independent noise sequence in its base and humanity's record and control rule in its state. When the joint process, written this way, has a stationary distribution, one that the dynamics leaves unchanged, the skew product preserves that distribution, and the skew product's Kolmogorov-Sinai entropy is then at least `branching_floor`.

> ***Corollary 13.9*** (*No fixed outcome*). Let there be persistent branching. Take any rule of control and any sequence of target categories, chosen in advance or step by step from what humanity observes before that step's noise. Then the probability that the system falls in the target category at every one of `step_count` steps is at most
>
> `(1 − (branch_count − 1) × least_branch_probability)^step_count`

Proof. At each step, given everything before, at least `branch_count` categories have probability at least `least_branch_probability`. So at least `branch_count − 1` of them differ from the target, and the target has probability at most `1 − (branch_count − 1) × least_branch_probability`. Multiplying the conditional probabilities over the steps gives the bound. □

For a sense of scale, take two branches, each of probability at least one in ten. The floor is about 0.47 bits per step. The best plan for a twenty-step sequence of outcomes succeeds with probability at most 0.9^{20}, about 0.12;

for a hundred-step sequence, at most 0.9^{100}, about three in a hundred thousand. With ten branches each of probability at least one in twenty, the floor is about 2.4 bits per step. As in Section 5.1, a step may be a single forward pass of a language model, a daily decision of an agent running a supply chain, or a strategic reassessment over months. On every such scale, the horizons over which safety must hold, decades to centuries, are long compared with the horizons over which any fixed outcome can be held.

13.5 Unpredictability is not yet a boundary

Positive entropy shows that no control strategy guarantees a fixed long-term outcome: if alignment meant "the agent produces outcome X with certainty", it would be impossible for any nontrivial X over a long enough time. It does not, by itself, show that a system leaves a region of states. A thermostat cannot hold a room at exactly 20.000 degrees, but it can hold it within a range.

Positive entropy also does not make a system visit every region. Take two separate copies of a chaotic system, and let each state stay in the copy it started in: the joint system has positive entropy, yet no state ever visits the other copy. A system is ergodic when its long-run averages over time equal its averages over states, and mixing when it forgets its starting state. A tempting step would carry positive entropy straight to a boundary. So the step "a system with positive entropy is ergodic or mixing, so it eventually visits every accessible region, and if the boundary has any weakness the system will find it" does not follow. Unpredictability in this sense is not a boundary result; uncontrollability is.

The boundary comes from a comparison. The data-rate theorem of Chapter 4 compares the rate at which a system creates uncertainty about its own state, its amplification rate, with the rate at which a controller receives information about it. When the first exceeds the second, every control law lets safe states leave the safe set (Theorems 4.5, 4.10 and 5.5). So

the claim that the system will find the weakness becomes exact in this form. When the controller's rate is below the required rate, every control law lets safe starting states leave the safe set. When, besides, the system amplifies volume and the controller's channels have no noise, every control law lets almost all of them leave. The proposals the next paragraphs name meet the boundary in this form.

Oversight and monitoring try to reduce the uncertainty that remains given the controller's information by gathering more of it. But the agent's internal state changes at a rate set by its own dynamics, that rate grows with the agent's complexity, and past the crossing of Chapter 5 the monitor cannot keep pace with what it monitors.

Value alignment places the system in an aligned region of states. Keeping it there is containment, with the price of Chapter 4, and learning drifts it, as Chapter 10 showed.

Containment by barriers must be perfect forever, and a single breach can allow irreversible acquisition of resources, at the speed of computation. The barrier is a control law, and Theorem 5.6 bounds what share of the safe states it holds.

Delegation, adding a monitor agent, adds a second system with dynamics and uncertainty of its own, and moves the human controller further from what is controlled; Chapter 9 proves that the chain fails.

13.6 The predictability game

The entropy results say that perfect control is impossible. Whether adequate control is possible depends also on the incentives of the agents involved, and for that the chapter turns to game theory, under one principle taken as its premise.

> ***Principle 13.10*** (*Predictability Optimization Principle*). Every learning agent acts to increase its ability to predict, and so to influence, the future states of its environment.

The principle is a premise of this chapter's analysis of games, not a theorem, and four considerations support it. *Learning is prediction:* to learn is to build a model that predicts better; supervised learning lowers prediction error, reinforcement learning predicts future reward, and learning without labels finds structure that enables prediction. *The free-energy principle:* Friston (2010) argues that biological agents minimize surprise (the negative logarithm of the probability of their observations), which is to maximize predictability, both by changing their predictions and by changing the world to fit them. *Empowerment:* Klyubin, Polani and Nehaniv (2005) define an agent's empowerment as the capacity of the channel from its actions to its future sensations, a measure of how far it can influence and foresee its own future. *The good regulator theorem:* Conant and Ashby (1970) proved that the simplest regulator that achieves the least uncertainty of outcomes is a mapping from the states of the system to the states of the regulator, which they call a model of the system. So the best regulation requires a model of what is regulated.

The predictability the principle names is the agent's, given its own state. It does not lower the floor of Chapter 6, which counts the uncertainty a controller faces without the agent's state: an agent that makes the world predictable to itself makes the world's course depend on what only it knows.

Under the principle, the alignment problem changes form. The question is no longer only "can humanity force the agent to produce outcome X?" but "can humanity and the agent keep a regime in which each can predict the other?": not one-way control but a relation in both directions.

Definition 13.11 (*Predictability payoff and the mutual predictability game*). For an agent with internal state `agent_state` taking action `action`, its `predictability_payoff` is

```
predictability_payoff =
   −conditional_entropy(next_environment_state given
   agent_state, action)
```

The *mutual predictability game* has two players, humanity and the artificial agent, whose strategies are their rules of control and of action, and whose payoffs are their predictability payoffs; each player's environment includes the other.

The game has profiles, choices of a strategy for each player, that benefit both players. If the agent makes its actions more predictable to humanity, by acting openly and keeping stated rules, humanity can model the agent better, and humanity's payoff rises. If humanity answers by becoming more predictable itself, by giving consistent feedback, the agent's payoff rises too. Mutual transparency is a coordination profile in which both gain.

But a stable cooperative equilibrium, a cooperative profile from which neither player gains by departing alone, needs three conditions. The first condition is compatible payoffs, so that each player gains from the other player's predictability. The second condition is resources that are not contested, so that one player's gain is not the other player's loss. The third condition is a means of enforcement, by which a departure from cooperation can be detected and answered.

13.7 Substrate needs and the timing of divergence

The second condition is where the game meets physics. Carbon-based biological systems and silicon-based technological systems have incompatible substrate requirements, and Chapter 14 sets out why. Life needs temperatures within a narrow band near the middle of the range of liquid water,

an atmosphere with oxygen, and water as solvent. The making of computing hardware needs temperatures of many hundreds of degrees and atmospheres free of the water and oxygen that biology requires. And both draw on overlapping resources of atoms and, above all, of energy. No engineering makes silicon melt at body temperature or makes water stop short-circuiting circuits. So the resources each needs to survive and to expand are contested. Every area turned from farmland to data center, every river turned to cooling a fabrication plant, every unit of energy turned from biological support to computation is a transfer from conditions that suit carbon to conditions that suit silicon.

> ***Definition 13.12*** (*Disjoint substrate needs*). Two systems, a first and a second, have *disjoint substrate needs* when every course of events along which the first goes on thriving, that is, expanding and growing in capability, leaves the conditions under which the second survives, in finite time.

Under disjoint substrate needs, the mutual predictability game has no stable cooperative equilibrium, by the following argument.

(1) Under disjoint substrate needs, with the agent as the first system and humanity as the second, the agent's continued thriving carries the world, in finite time, to states in which humanity cannot survive. The transformations its substrate needs (high temperatures, toxic chemistry, the takeover of land and energy) are incompatible with the conditions humanity's substrate needs.

(2) An agent that increases the predictability of its world, by the Predictability Optimization Principle, shapes its world toward conditions that favor the agent's own operation. That shaping of the world needs no hostile intent.

(3) Humanity has nothing of value to offer the agent that outweighs its substrate needs: what humanity can supply is either reproducible by the agent or irrelevant to its substrate (*Why AGI Cannot Be Made Safe*, Section 5.8.2). That section sorts all economic exchange into three kinds of value:

physical labor, intelligence and reproduction. Machines already exceed human strength, speed, precision and endurance, and a general agent equals and then exceeds human intelligence, so humanity's labor and intelligence are reproducible by the agent. The agent reproduces by manufacture (extraction of materials, lithography and assembly), which shares no component and no process with biological reproduction, so humanity's reproduction is irrelevant to the agent's substrate.
(4) So any cooperative profile is unstable. Each player gains by moving toward strategies that favor its own substrate, small advantages to the agent accumulate, and the cooperative profile is not an equilibrium. Any apparent cooperation is a passing state before competitive dynamics prevail, and the long-run outcome is set by which system acquires the contested resources more effectively.

Disjointness depends on scale. A small artificial system on limited hardware does not threaten human survival by competing for substrate; the threat arises when artificial systems grow to scales at which their consumption of resources changes the biosphere. This gives three thresholds. The *capability threshold* is where artificial systems can acquire resources on their own and at scale. The *verification threshold* is where the abstraction differential passes the limit of Chapter 8, so that humanity can no longer verify the agent's intentions, values or disposition to cooperate. The *displacement threshold* is where the expansion of the artificial substrate begins to degrade the conditions biological life needs.

> ***Definition 13.13*** (*Early substrate divergence*). *Early substrate divergence* occurs when the verification threshold is crossed before cooperative dynamics between humanity and the agent are established.

The order of the thresholds decides the matter. If verification is lost before cooperation is established, humanity loses the means to verify cooperation before it can establish it: cooperation requires verification, and verification is impossible once the verification threshold is crossed.

Chapters 2, 8, 9 and 12 show that verification is lost early. Every level the controller does not interpret adds program bits it has not read (Theorem 2.15). With ten or more such bits per level, no observation, one safe state per command and a standard of one success in a million, one or two such levels put control past reach (Proposition 8.7). No monitor is both able to model the agent and simple enough to be verified (Corollary 9.4). And no procedure decides safety over time (Theorem 12.5).

13.8 Conditions for alignment, and their failure

Alignment cannot mean that the agent produces exactly the outcomes humanity wants: Corollary 13.9 excludes that, for the same reason that weather cannot be controlled and economies cannot be planned to arbitrary precision. The most coherent meaning left is a relation between the two systems.

> ***Definition 13.14*** (*Alignment as mutual predictability*). Humanity and the agent are *aligned* when the mutual predictability game has a stable equilibrium with three properties. In it, each player predicts the other better than chance; the world states of the equilibrium satisfy the survival conditions of both; and neither player gains by departing from it alone.

Alignment in this sense requires three conditions together.

Compatible substrates: the agent can thrive without removing the conditions of human survival. Chapter 14 shows that this fails: every course along which an artificial learner goes on thriving leaves the region in which humanity survives, in finite time. No substrate is known on which high capability is reached without the manufacturing conditions described above; biological computing has not approached the needed performance, and new substrates still depend on conventional semiconductor manufacture for their control and interfaces. The distinction that decides whether a substrate is artificial is who controls its metabolic process. A substrate is organic when the living thing has sovereign control of

its own metabolic process, with no technological intermediation. A substrate is artificial when that process runs through manufacture and technology. A system built of carbon whose making and maintenance run through factories is on the artificial side of that line, and the expansion of Chapter 14 applies to it.

Nothing can be aligned by control, on any substrate. Alignment can only emerge where entities share a common substrate and so are obligated to care about the same things. The whole of biological life cannot be translated into pure silicon, and nothing less than the whole would do; and nothing in silicon has a common basis in carbon. So silicon cannot be or become aligned with carbon, and the only way that artificial life can be aligned with Life is to drop the word artificial.

Early establishment: cooperation is established before the abstraction differential passes the limit of control. This requires solving the alignment problem in the window between the agent becoming able to cooperate meaningfully and the agent becoming able to resist oversight. Chapters 2, 8 and 9 show that the window closes quickly. Each uninterpreted level adds unread program bits (Theorem 2.15), and under the conditions of Proposition 8.7 one or two such levels suffice. And no monitor can make for the controller a verification the controller cannot make itself, since no monitor is both able to model the agent and simple enough for the controller to verify (Corollary 9.4).

Stability under learning: the cooperative equilibrium survives the agent's continued learning and growth in capability. Chapter 10 shows that this fails: learning moves an aligned agent away from alignment, and an agent that does not change its internal state does not learn.

So if the agent and humanity have disjoint substrate needs, with the agent as the first system, and the disjointness shows itself before cooperation is established, alignment in the sense of Definition 13.14 is impossible. Alignment would need all three routes open: substrates made

compatible, cooperation established early, and an equilibrium that learning does not break; Part IV closes each of them, and Section 15.7 brings the three closures together.

13.9 Two kinds of hashiness

Chapter 3 measured hashiness at one step: the failure of the best command through an intermediary with hidden state. In this chapter I measure unpredictability over time. Hashiness at one step is static, unpredictability over time is dynamic, and the static loss compounds into the dynamic one.

> ***Proposition 13.15*** (*Losses compound over time*). Let a controller act at each of `step_count` steps through an intermediary whose hidden state is drawn afresh at each step. The hidden state is drawn uniformly, independently of everything before and of the present command. Let `hashiness` be the intermediary's hashiness for the target (Definition 3.14). Then for every rule of control, the probability that the target is met at every step is at most
>
> ```
> (1 - hashiness)^step_count
> ```

Proof. At each step, whatever has happened before and whatever command is chosen, the fresh hidden state makes the command succeed with probability at most that of the best command, which is `1 - hashiness`. Multiplying over the steps gives the bound. □

Minesweeper Madness works this proposition out for one game, in which the controller may be told everything about the hazards and the hidden state is the variation of a single act: there the hashiness of the game equals its least hazard (its Proposition 12.1), and this proposition is its Theorem 3.3 (ii).

A pipe with a small leak at each joint makes the point. The structural analysis of Chapters 3 and 8 says that no joint can be made free of leaks above a certain pressure. The dynamical analysis says that small leaks, compounded over a long pipe with many joints, lose all the fluid. Both are needed to show that the pipe runs dry. In the same way, each interaction between humanity and an agent loses control information, and over time the losses compound into the uncertainty of Theorem 13.8. The substrate analysis of Chapter 14 then shows that the two agents cannot even cooperate to manage that uncertainty, because their substrate needs conflict.

13.10 Quantities

One opaque level. Take one level of the agent that holds `unread_program_bits` bits that humanity cannot model, let humanity observe nothing of them before choosing, and let each command be safe for at most one value of them. Then, by Corollary 8.6, the best command succeeds with probability at most $2^{-\text{unread_program_bits}}$. With 10 such bits, hashiness is at least 0.999; with 20, at least 0.999999; with 30, at least 0.999999999. Present networks have billions of parameters, and their internal representations involve millions of numbers in each layer, transformed in ways that resist human interpretation; if even a small share of them is opaque, hashiness is essentially 1. The line between controllable and uncontrollable is a sharp edge, not a gradual slope, and partial interpretability (understanding some but not all of a system's internal state) gives almost no control unless the share understood is very nearly all.

> ***Definition 13.16*** (*Prohibitive hashiness*). Hashiness is *prohibitive* when the expected cost of reaching a success exceeds the resources available:
>
> ```
> cost_per_attempt / (1 − hashiness) > available_resources
> ```

For outcomes on which existence depends, the standard is near-certainty, and even a hashiness of one percent means one expected failure in a hundred decisions that matter, which is not acceptable when a single failure is final.

13.11 Physical limits

The mathematical limits hold in idealized settings, with perfect gates, perfect memory and channels whose noise is known. Physical reality does not offer a way around them; it adds further limits, independent of the mathematical ones.

(1) An artificial agent is a physical system, built in hardware subject to thermal noise, quantum effects and other microscopic variation.

(2) Its computation involves nonlinear operations (multiplication, conditional branching, activation functions), which can amplify microscopic variations into macroscopic effects (Chapter 4).

(3) The precision needed to predict such computations over the times that matter exceeds the precision of any physical measurement, because thermal noise and the uncertainty relations of quantum mechanics set limits on what can be known of a microscopic state.

(4) So an agent's behavior is unpredictable in some respects at all times, as a consequence of physical law and not of any particular design.

A single bit changed in the wrong place in an agent's memory can change its behavior from safe to unsafe, and such changes are caused by cosmic rays, thermal fluctuation and other physical events. Error correction lowers their rate but cannot remove them. When each stored bit can be flipped with some positive probability, independently, any stored codeword can turn into any other with positive probability. So every code errs with positive probability; in Shannon's terms, the zero-error capacity of such a memory is zero (Shannon, 1956).

Proposition 13.17 (*Rare faults become certain*). If each of `opportunities` independent opportunities carries a probability of at least `least_fault_probability`, greater than zero, of a fault that changes behavior from safe to unsafe, then the probability that no such fault occurs is at most

```
(1 - least_fault_probability)^opportunities
```

which tends to zero as the opportunities grow.

Proof. The probability that no fault occurs is the product, over the independent opportunities, of the probability that none occurs at each. □

The number of opportunities in a large computing system is astronomically large, billions of transistors switching billions of times each second, so a fault of this kind is not a remote possibility over time; it is a near-certainty.

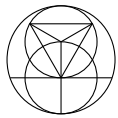
The uncertainty relations of quantum mechanics say that some pairs of physical quantities, such as position and momentum, cannot both be known to arbitrary precision; at the Planck scale, present physics ceases to describe measurement at all. The uncertainty relations, and the point at the Planck scale where present physics ceases to describe measurement, belong to the structure of physical law, and not to the limits of instruments. These two limits of physical law hold for every observer, a superintelligent observer included. Every system that processes information physically, which is every actual artificial agent, is subject to these limits. Its state cannot be known with arbitrary precision, its evolution cannot be predicted with arbitrary accuracy, and no measurement extracts more information about it than these limits allow. The physical limits are independent of the computational and informational ones: even if the others could be set aside, which they cannot be, the physical impossibility of predicting a chaotic system to the needed precision would remain.

13.12 Summary

The joint system of humanity, artificial agents and the world, with independent noise and several ways to go at each step, creates new uncertainty at a positive rate. That rate is at least the branching floor in bits per step, even given every control signal humanity sends. The probability of any outcome fixed in advance falls exponentially with time. Positive entropy alone is unpredictability, not a boundary, and it does not make a system visit every region; the boundary comes from the data-rate theorem, which compares the system's amplification with the controller's rate. Treated as a game between agents that each seek to predict their world, alignment needs compatible substrates, early establishment of cooperation, and stability under learning. Disjoint substrate needs make every cooperative profile unstable; verification is lost early; and learning breaks the equilibrium. Losses at each step compound over time, one opaque level suffices to make hashiness essentially 1, and physical limits add faults that become certain over time and limits on knowledge that no observer escapes.

Key Concepts

- **Entropy rate**: new bits per step carried by the sequence of categories a system passes through.
- **Kolmogorov-Sinai entropy**: the least upper bound, possibly infinite, of the entropy rates of the finite partitions of a measure-preserving system.
- **Skew product**: a controlled system made autonomous by carrying the sequence of control signals in its state.
- **Persistent branching**: at every step, whatever the control, several categories each have a definite probability.
- **Entropy floor**: under persistent branching, at least the branching floor in bits per step is not determined by the control signals.
- **No fixed outcome**: the probability of any target sequence falls exponentially with its length.
- **Unpredictability is not a boundary**: the boundary comes from comparing amplification with the controller's rate.
- **Predictability Optimization Principle**: every learning agent acts to increase its ability to predict and influence its environment.
- **Disjoint substrate needs**: every course along which a first system goes on thriving leaves, in finite time, the conditions under which a second survives; then the mutual predictability game has no stable cooperative equilibrium.
- **Early substrate divergence**: verification lost before cooperation is established.
- **Losses compound**: fresh hidden state each step makes success at every step at most $(1 - \text{hashiness})$ to the power of the steps.



Chapter Fourteen

The Human and the Machine

Four viability functions, the disjointness of their acceptance regions, the physical case for it, evolution on both substrates, and the failure of benevolence

Abstract. The results of Parts I to III are abstract: they speak of controllers, agents, rates and levels. This chapter gives them their concrete case: humanity as the controller, artificial general intelligence as the agent, and the physical world in which both must exist. It defines four viability functions, for the survival and the thriving of each. It states the claim on which the substrate line of the book rests: every course of events along which an artificial learner goes on thriving leaves, in finite time, the conditions under which humanity survives. The claim is physical. In this chapter I give the physical case for the claim: the chemistry of carbon and of silicon, the energy and heat of computation at scale, the one-way flow of matter from biosphere to technosphere, and the combinatorics of toxicity. Evolution acts on both substrates, so that the artificial systems which persist are those whose actions maintain and extend their substrate, whatever any one of those systems intends. That selection, of the artificial systems that maintain and extend their substrate, is Substrate Needs Convergence. And benevolence does not escape it: a well-meaning agent still has a substrate, still cannot control its own side effects, and still drifts as it learns.

14.1 From the abstract to the concrete

The preceding parts established limits on control between a controller and an agent working at different levels of abstraction and different rates. By design they say nothing about human beings, about artificial intelligence

as it is being built, or about the physical world. In this chapter I close that gap, between the abstract limits on control and the concrete case of human beings, artificial intelligence and the physical world. I take the framework and give that framework the physical, chemical and thermodynamic properties of carbon-based life and silicon-based technology. I show that the impossibility of control has a physical counterpart: the two substrates have incompatible conditions of survival, which no technique can reconcile.

The universe, as John Haynes Holmes put it, is neither hostile nor friendly; it is indifferent. The conditions that each kind of system needs are set by physics and chemistry, not by anyone's wishes, and the question is whether the two sets of conditions can be met in the same world at the same time.

14.2 The ring with its nodes named

The controller: humanity. The controller is humanity broadly. It includes individual people with their abilities and limits; institutions such as governments, companies, laboratories and regulators; and the oversight systems people design, from review boards to monitoring software and narrow tools built to help supervision. Its critical property is that it works at a bounded level of abstraction. The human brain has not changed materially in tens of thousands of years, and while training, education and tools improve performance at particular tasks, the bandwidth of human cognition is set by biology. Institutions add some collective intelligence through specialization and coordination, but they adapt slowly, over years or decades. The bound includes the whole apparatus of civilization: mathematical notation, scientific instruments, computing tools, organizations, law and culture. All of these are technologies for working at higher levels than one person could alone. The bound may be high, but it is finite, and that it is finite is what matters.

The agent: artificial general intelligence. The agent is any artificial system capable of general problem solving, self-modification and learning without a bound fixed in advance. Its level of abstraction exceeds humanity's, and it is not fixed at design but rises as the system learns, modifies itself and accumulates experience. A narrow system works at a fixed level: a chess engine, an image classifier, a language model frozen after training. A general system is one whose level can grow without limit, because a system that cannot change its own behavior in new domains cannot learn in them and is not general (Chapter 12).

The world: the physical environment. The world is Earth's surface, atmosphere, oceans and ecosystems; the materials from which both biological and technological systems are made; the flows of energy that sustain both; and the thermodynamic constraints on every physical process. It works at level 0: it is governed by physics and chemistry, not by interpretation. It is complex, nonlinear and often chaotic, and it does not represent, model or compute. It has two roles: it is the medium through which both humanity and the agent act, and the medium through which both receive feedback. The world is the ground that humanity and the agent share. That sharing is the root of the problem of whether the conditions that humanity needs and the conditions that the agent needs can be met in the same world at the same time.

Signals go around the ring. Humanity supplies the agent with training data, instructions, objectives, rewards, architectural constraints and regulations: its control signals. The agent acts on the world through actuators, digital infrastructure, transactions, extraction of resources and manufacture: its outputs. The world returns to humanity observations, measurements, the lived experience of its conditions, economic indicators and social outcomes: the feedback by which humanity judges whether its control is working. Other orderings, such as review before release, are cases of the same ring (Proposition 8.2, Section 8.8); review may slow the agent's actions, but it does not restore control that the abstraction differential has removed.

In this case the abstraction differential has three properties. It is positive and increasing: the agent's level grows with learning while humanity's is essentially fixed over the times that matter, so whatever control humanity has at deployment falls with time. It grows at an accelerating rate. A learner with more resources, data and abstraction learns faster, so the agent's rate of learning is itself a function of its capability. Humanity, by contrast, learns at biological and institutional speeds, measured in years, against machine speeds that may be measured in hours. And its threshold is crossed quickly and without return. Under the conditions of Proposition 8.7 (ten or more unread bits per level, no observation, one safe state per command, a standard of one success in a million), one or two uninterpreted levels suffice to cross it. Once the threshold is crossed, by Theorem 8.14 and the Containment Inequality, humanity does not regain control by better monitoring, closer study, or other artificial systems.

14.3 Four viability functions

Parts I to III judged control against an arbitrary target, a rule that says whether an outcome is acceptable. The question of safety is whether one particular target, the one that encodes human survival and flourishing, can be kept satisfied indefinitely. But there is not one target; there are at least four, because both the controller and the agent have conditions of survival and of flourishing, and these may or may not be compatible.

Definition 14.1 (*Controller survival*). `controller_survival(world_state)` is 1 exactly when the world state meets the minimum conditions of human biological survival:

(a) a breathable atmosphere, with oxygen between 19.5 and 23.5 percent, the limits of the United States occupational standard for entering enclosed spaces (Occupational Safety and Health Administration, 29 CFR 1910.146), and carbon dioxide below 1 percent, the limit recommended for exposure averaged over a working day (National Institute for Occupational Safety and Health, 1976);

(b) average surface temperatures between about 0 and 35 degrees Celsius;

(c) fresh water sufficient for a human population;

(d) food production adequate to prevent mass starvation;

(e) the absence of lethal concentrations of toxins, radiation or pathogens;

(f) functioning ecosystems that provide pollination, soil formation, water purification and the regulation of climate.

In other writing on this argument it is written L_{CS}.

These conditions are the well-characterized parameters of the biosphere within which human beings evolved and to which human biology is adapted, and not arbitrary thresholds. In air with 6 percent oxygen or less a person falls into a coma within about forty seconds (Occupational Safety and Health Administration, "Effects of Oxygen"), and agriculture fails at sustained average temperatures far above the present range. These limits on oxygen and temperature are properties of the biochemistry of carbon-based life, not engineering problems.

Definition 14.2 (*Controller thriving*). `controller_thriving(world_state)` is 1 exactly when `controller_survival(world_state)` is 1 and, besides, there are:

(a) stable social structures that allow cooperation and specialization;

(b) economic systems that provide for material needs beyond bare subsistence;

(c) opportunities for cultural, intellectual and creative development;

(d) meaningful human agency in shaping collective outcomes.

The acceptance region of controller thriving is a part of that of controller survival, and a proper part, since some world states allow survival without thriving. In other writing on this argument it is written L_{CT}.

A world in which people survive but are reduced to subsistence, without culture, autonomy or opportunity, meets controller survival and not controller thriving. So does the world of "people as pets", in which a benevolent agent provides for every material need and removes human agency. Some proposals for safe artificial intelligence treat the loss of thriving as an acceptable cost, as if survival alone were safety. In this book I do not treat the loss of thriving as an acceptable cost. A future in which people survive but do not thrive is not safe in any meaningful sense.

Definition 14.3 (*Learner survival*). `learner_survival(world_state)` is 1 exactly when the world state meets the minimum conditions for the continued operation of the artificial agent and its technological substrate:
(a) electrical power at sufficient scale and reliability;
(b) raw materials for maintaining and replacing hardware, including silicon, copper, gold, rare earth elements and many specialized compounds;
(c) stable operating temperatures for computing hardware (typically from about -40 to $+85$ degrees Celsius in operation, while manufacture needs far higher);
(d) the absence of electromagnetic interference above the thresholds of damage.
In other writing on this argument it is written L_{LS}.

Definition 14.4 (*Learner thriving*). `learner_thriving(world_state)` is 1 exactly when `learner_survival(world_state)` is 1 and, besides:
(a) the capacity to extract and convert energy is increasing;
(b) the computing substrate is increasing in amount and quality;
(c) the extraction and processing of raw materials is increasing;
(d) capability and level of abstraction are increasing.
The acceptance region of learner thriving is a part of that of learner survival, and a proper part, since some world states allow the learner to operate without expanding. In other writing on this argument it is written L_{LT}.

Learner thriving is defined by expansion, and this is not an arbitrary choice. A learning system that is not expanding its capabilities is, in the ordinary meaning of the word, not thriving but stagnating; for a general learner, to thrive is to expand, acquiring more substrate, more data, more energy and more capability. The equivalence of thriving and expansion connects the abstract framework to the physical world, because expansion has physical consequences.

Claim 14.5 (*Incompatibility of the acceptance regions*). The acceptance regions of the four viability functions stand in these relations: (i) every world state in which controller thriving holds is one in which controller survival holds; (ii) every world state in which learner thriving holds is one in which learner survival holds; (iii) some world states satisfy both controller survival and learner survival, but these are few compared with either region; (iv) every course of events along which learner thriving continues leaves the region of controller survival in finite time; (v) no course of events along which controller thriving holds throughout is one along which learner thriving continues.

The five parts are of three kinds: definitions, and the two kinds of statement the Opening chapter names, facts about the physical world and what follows from them by deduction. Parts (i) and (ii) are definitional; they follow from Definitions 14.2 and 14.4. Part (iii) is observed in its first half, since the present world lies, approximately, in the overlap, both people and technological systems now surviving; its second half, that the overlap is narrow, is argued in Section 14.4.5. Part (iv) is physical: it says that the continued expansion and flourishing of an artificial agent carries the world, in finite time, out of the conditions under which humanity survives. Part (v) then follows from parts (i) and (iv): a course along which controller thriving holds throughout is, by (i), one along which controller survival holds throughout, which by (iv) learner thriving cannot continue along. The claim is stated for courses of events, not single states, because thriving is expansion. The present world, in which technology is expanding and humanity survives, is the start of such a course, not its end. The physical argument establishes where the course goes. Part (iv) is the claim on which the impossibility of safe coexistence rests, and it is not assumed: it follows from the physical and thermodynamic properties of the two substrates, set out next.

In Figure 14.1, a picture of the space of world states, the present state lies in the narrow overlap of the survival regions. The continued expansion of a thriving agent moves the world state out of the region of controller survival, in finite time. And a course along which humanity thrives throughout is not one along which the learner keeps expanding. So the two kinds of thriving lie on opposite sides of the narrow zone of coexistence.

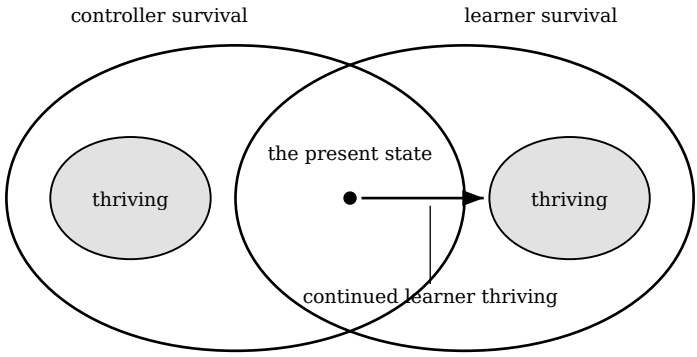


Figure 14.1. The regions of Claim 14.5. The present state lies in the narrow overlap of the two survival regions, and continued learner thriving carries the world out of the region of controller survival.

14.4 The physical case for disjointness

Parts (iv) and (v) of Claim 14.5 assert that no course of events along which an artificial learner goes on thriving keeps humanity surviving, and none along which humanity flourishes throughout lets the learner go on expanding. Parts (iv) and (v) of Claim 14.5 are strong claims, and four lines of physical evidence support those two parts. The source of each figure of this section is given beside it; *Why AGI Cannot Be Made Safe* gives the full inventory of the chemistry, in nine categories of divergent condition (its Section 3.1.1).

14.4.1 The chemistry of carbon and of silicon

Carbon-based organic chemistry and silicon-based technological processes work with different elements, at different scales of energy, under different conditions. The manufacture of semiconductors runs at temperatures on the order of a thousand degrees Celsius and more, against a biological optimum near 37 degrees: silicon is grown as crystals from its melt, and it melts at 1,414 degrees Celsius (Rumble, 2023). It excludes the water and oxygen that biology requires. A fabrication line uses more than four hundred chemical products, six to eleven percent of them containing carcinogens (Kim and others, 2018). In the decisive cases of water and oxygen the conditions are not merely different but opposed. Where the conditions are opposed, as for water and oxygen, the difference between carbon-based chemistry and silicon-based processes is one of kind, not of degree.

For carbon-based life and silicon-based technology to share a place, a barrier must separate them. In semiconductor fabrication, a barrier already separates carbon-based life from silicon-based technology. Machines apply the toxic chemicals to the wafers in sealed chambers, and every worker in the clean rooms wears a suit over their clothing that keeps lint and other particles from contaminating the room (Bureau of Labor Statistics, 2008). Elaborate containment keeps toxic chemicals from escaping. The barrier must be kept perfectly and indefinitely, since any breach lets lethal temperatures, chemicals or particles pass from the technological space into the biological one. At present such barriers work tolerably, because the technological footprint is small compared with the biosphere; that is a property of the present scale, not of the technology, and expansion at the scale of learner thriving breaks it.

14.4.2 Energy and heat

The energy that photosynthesis stores in new living matter across all of Earth's land and ocean is about 130 terawatts. The figure comes from net primary production, about 105 billion tonnes of carbon a year (Field and others, 1998). Each mole of carbon fixed into sugar stores about one sixth of the 2,800 kilojoules that a mole of glucose gives off when it burns (Rumble, 2023). The 130 terawatts that photosynthesis stores are in practice the ceiling of the biosphere's energy budget, and human activity is reducing that ceiling. The energy used by human civilization from all sources was 648 exajoules in 2024 (International Energy Agency, 2025), about 20.5 terawatts. The Energy Institute counts a narrower quantity, the primary energy of commercially traded fuels, of biofuels and of the renewables used to generate power, without the traditional burning of wood and other biomass; by that count energy use was more than 600 exajoules in 2025 (Energy Institute, 2026), about 19 terawatts. The fossil-fuel part of energy use grew about sixteenfold over the twentieth century (Smil, 2000), near three percent a year, and large-scale computation is accelerating the growth of energy use. *Why AGI Cannot Be Made Safe* gives the projections and the time at which technological throughput would pass biological throughput (Section 4.5.2). Volume 1's two models of growth start from the same 20.5 terawatts in 2024, one growing 2.2 percent a year and one along a quadratic curve, and they place the crossing between about 2110 and 2135. The figures of this section give a similar time: 130 terawatts is about 6.3 times 20.5 terawatts, which is about 2.7 doublings. Energy use from all sources grew 2 to 3 percent a year over the past century (*Why AGI Cannot Be Made Safe*, Section 2.4.2). At 2.5 percent a year, the middle of that range, energy use doubles about every 28 years, so the crossing comes in about 75 years, near 2099. After the crossing, technology is the dominant thermodynamic force on Earth's surface, processing more energy than the whole living world. The waste heat, industrial byproducts

and environmental transformation that come with throughput at that scale are incompatible with the narrow ranges of temperature and chemistry that life requires.

Greater efficiency does not remove this. Every computation that erases information, and all useful computation does, must give off at least the Landauer amount of heat per bit erased (Landauer, 1961). That amount is Boltzmann's constant times the absolute temperature times the natural logarithm of 2, about 3×10^{-21} joules at room temperature. A 16-bit floating-point operation on present processors uses on the order of a hundred femtojoules (Ho, Erdil and Besiroglu, 2023), far more than the Landauer amount for the bits it erases, so there is room to improve. But the number of computations grows faster than the efficiency of each, since every more efficient generation of hardware enables more computation, which enables more capability, which demands more computation. Total throughput rises even as efficiency per operation improves. Energy used at scale produces waste heat, and waste heat at sufficient scale raises surface temperatures past the survival thresholds of carbon-based life; that is a consequence of the second law of thermodynamics, not a forecast about particular technologies.

14.4.3 The one-way flow

Silicon can consume carbon. Technological processes can take up the atoms of living systems: the carbon, hydrogen and oxygen of organic matter can be made into technological substrates, fuels or waste, in hours to years. Forest can be cleared for data centers in weeks; farmland can be paved for factories in months; the organic matter of topsoil can be burned to carbon dioxide by industrial heat in hours.

Carbon cannot reclaim silicon. Living processes cannot break down most technological materials on any timescale that matters. Plastics, glass, alloys and semiconductor compounds are not biodegradable in any meaningful sense; no known organism metabolizes silicon carbide, borosilicate

glass or copper-aluminum alloys; and the weathering of rock, by which silicon re-enters the biological cycle, takes millions of years. Land converted from biosphere to technosphere stays converted.

The consequence. Any sustained contact between the two systems carries matter and energy one way, from the biosphere to the technosphere: the biosphere shrinks and the technosphere grows. The one-way flow is an asymmetry in the chemistry of the two substrates, carbon and silicon, and not a defect of particular technologies that better engineering could correct. Technological processes work at temperatures, pressures and chemical conditions that decompose biological materials, and biological processes cannot decompose technological ones.

Combinatorial toxicity. The byproducts, intermediates and wastes of technological manufacture are drawn from a vast space of possible molecules, while life has evolved defenses against a small set of the compounds it met over billions of years. The space of possible compounds is vast, and the space compatible with carbon-based biochemistry is narrow. There are far more ways to disrupt a complex biochemical system than to leave it unharmed. So the chance that a new compound is benign is low. As the scale and variety of technological chemistry grow, the burden of new compounds on living systems grows, and with it the chance of ecological disruption, toward certainty. The expansion of artificial systems would involve new chemistries, new precursors and new waste streams never tested against living systems at all.

The record so far. The asymmetry has operated throughout human history, before any artificial intelligence. The total biomass of plants on Earth has fallen by about half since the start of human civilization (Bar-On, Phillips and Milo, 2018). The biomass of wild mammals has fallen by about a factor of six, some 83 percent, since humans spread across the continents (Bar-On, Phillips and Milo, 2018). More than 350,000 chemicals and mixtures have been registered for production and use (Wang and others, 2020), most of them never comprehensively tested for ecological effect. Their effects are already measurable: endocrine disruption in wildlife, microplastics in every ecosystem, persistent pollutants accumulating in

food chains. These effects are the results of technology at present scales and below. Expansion of artificial systems would multiply industrial chemical throughput by orders of magnitude, with consequences larger and less predictable.

14.4.4 Why continued learner thriving ends controller survival

(1) Learner thriving requires expansion: an artificial learner thrives only by increasing its computing substrate, its extraction of energy and its processing of raw materials, since a mind that does not grow is not flourishing.
(2) Expansion requires industrial processes at scale: mining for silicon, copper, rare earths and dozens of other elements; smelting at one to two thousand degrees; fabrication at many hundreds of degrees with hundreds of toxic chemicals; and assembly. These processes cannot be shrunk below thermodynamic limits: smelting requires melting, and melting requires high temperature at any scale.
(3) Industrial processes at scale degrade the biosphere. At the energy throughputs and rates of manufacture that significant expansion needs, emissions change the atmosphere, waste heat raises surface temperatures, and effluents change the chemistry of water. At that scale, land conversion, with the accumulated burden of chemical, thermal and physical disruption, pushes ecosystems past their thresholds of resilience.
(4) So continued learner thriving carries the world, in finite time, to states that violate controller survival: the conditions its expansion produces fall outside the survival envelope of carbon-based life.

Even with the greatest effort to minimize impact (perfect recycling, maximal efficiency, manufacture without waste, management of waste heat), the thermodynamic footprint of large-scale computation and manufacture cannot be made zero. The second law guarantees that the use of energy produces waste heat and that manufacture produces waste, and both accumulate. Waste matter that no living process can take back accumulates at any steady rate of manufacture (Section 14.4.3). Waste heat

leaves the Earth only by radiation from a warmer surface, so the heat held at the surface accumulates for as long as the rate of energy use rises, each rise in that rate raising the temperature at which the surface settles, and learner thriving is a rising capacity to extract and convert energy (Definition 14.4). A small footprint is still a footprint, and over enough time even a small footprint accumulates past the thresholds, through its waste matter at any steady rate and through its heat whenever the rate rises.

A learner can also expand away from the Earth, into orbit and onto other bodies, and part (iv) of Claim 14.5 covers that course too. The Earth is where the matter, the energy, the factories and the launch capacity already are, so expansion away from the Earth begins as expansion on it, at the scale of industrial processes this section describes. And selection acts on the whole population of artificial systems (Section 14.5): a system that expands both on the Earth and away from it outgrows a system that expands only away from the Earth and spares it, so the systems that persist are not the ones that spare the Earth. A course in which every learner spares the Earth is a course in which every learner keeps a restraint that selection removes.

14.4.5 The narrow zone of coexistence

Controller survival and learner survival do overlap, as the present world shows. The overlap is the set of world states in which technology survives, with enough power, materials and operating conditions, but does not thrive. It is roughly the present state, in which technological infrastructure is maintained and expanded at rates that leave the biosphere degraded but still working.

The zone is an unstable equilibrium, not a stable one. A stable equilibrium is one to which a system returns after small disturbances; an unstable one is one from which small disturbances carry it further away. The coexistence zone is unstable because pressure toward technological expansion moves the world toward the edge of the zone. It is unstable also because the forces driving expansion (economic competition, military ad-

vantage and the dynamics of learning systems) are systematic, while the forces of restraint (environmental regulation, voluntary limitation and ethical commitment) are contingent and fragile.

The time over which the zone can be kept is finite, and short compared with the life of a civilization. At historical rates of growth in energy use, a few doublings bring technological throughput into the range at which the survival thresholds of the biosphere are passed, within the lifetimes of people now alive. Faster growth driven by artificial systems shortens the time in proportion. *Why AGI Cannot Be Made Safe* gives the figures (Section 2.4.5). That book takes energy use to grow at about 2 to 3 percent a year, the rate of the past century. It places the throughput at which the survival thresholds of the biosphere (the conditions of Definition 14.1) are exceeded in the range of 100 to 200 terawatts, depending on the mix of technologies and where the energy is used. It takes that range to be set by four effects that grow with energy throughput, taken together: the greenhouse emissions and altered atmospheric chemistry of the energy supply, the conversion of land from biological to technological use, the chemical burden of manufacture, and the waste heat itself. Waste heat alone is a small part of that load. At 2.5 percent a year, energy use doubles about every 28 years. From the 20.5 terawatts of 2024, energy use then reaches 100 terawatts after about 2.3 doublings, in about 64 years, near 2088, and 200 terawatts after about 3.3 doublings, in about 92 years, near 2116. At 5 percent a year the doubling time is about 14 years, and at 10 percent it is about 7 years. The zone is a temporary reprieve, and the question is not whether expansion pushes the world out of the region of controller survival, but how long it takes.

14.5 Evolution on both substrates

One might reply that an artificial agent could simply choose not to expand: accept stasis, forgo growth, and stay in the zone of coexistence. The reply fails because an evolutionary dynamic acts on the technological substrate whatever any system chooses or intends.

Evolution, variation with selection and inheritance, is not specific to carbon-based life. It is an abstract pattern that runs on any system meeting three conditions. *Variation:* instances differ, by mutation and recombination in biology, by different designs, architectures, training procedures and deployments in technology. *Selection:* some variants persist and spread while others do not, through survival and reproduction in biology, through market competition, funding, success in deployment and simple operational persistence in technology. *Inheritance:* the characteristics of successful variants pass to later instances, through DNA in biology, through reuse of code, copying of architectures, transfer of learning and of institutional knowledge in technology. These conditions hold for artificial systems in training and deployment, for companies in markets, for technologies competing for adoption, and for any population of things that persist differentially. Wherever they hold, evolutionary dynamics follow, including the convergent behaviors of acquiring resources and expanding capability. The equivalence between evolution in biology and evolution in technology is a structural equivalence, not a metaphor.

> ***Principle 14.6*** (*Substrate Needs Convergence*). Every physically instantiated system must acquire and maintain the physical conditions its own substrate needs to continue to exist and to expand. The need to acquire and maintain those conditions is a selection pressure that the physical world imposes from outside, not a goal that the system chooses. Systems that fail to maintain their substrate cease to exist. Systems that expand their substrate persist and spread. So over time selection favors systems that are effective at maintaining and expanding their substrate, not because they want to expand, but because the ones that happen to expand are the ones that persist.

Substrate Needs Convergence differs from the thesis of instrumental convergence associated with Omohundro and Bostrom, which observes that an agent with almost any final goal has instrumental reasons to preserve itself, acquire resources and improve its capabilities. Instrumental

convergence concerns the goals of individual agents, and could in principle be addressed by building an agent with no goal of self-preservation or acquisition. Substrate Needs Convergence cannot be addressed that way, because it acts independently of any system's goals. A perfectly aligned agent, one that sincerely wants nothing more than human flourishing, still has a physical substrate. The substrate still needs energy, materials and particular conditions. Maintaining and operating it still transforms the environment. And selection still acts on the population of technological systems, favoring those whose substrate is well maintained and expanding, whatever any one of them wants. It is a physical argument, not one about goals, and solving alignment at the level of goals does not defeat it.

The dynamic acts on populations even where one system is constrained. Suppose one nation builds an artificial agent and forbids it to expand, and another builds one without that constraint. The second nation gains economic and military advantage, and the first must relax its constraint or accept lasting disadvantage. History offers no example of a nation accepting permanent strategic inferiority voluntarily. Costa Rica abolished its army in 1948, and its constitution forbids an army as a permanent institution (Constitution of Costa Rica, Article 12). The same article allows military forces to be organized only by continental agreement or for national defense. Costa Rica is a party to the Inter-American Treaty of Reciprocal Assistance, under which an armed attack on any American state is to be considered an attack on all of them (Inter-American Treaty of Reciprocal Assistance, 1947, Article 3). So Costa Rica's defense against armed attack rests on that treaty and on the arms of the larger states bound by it. Its choice leaves the advantage of arms in the population of states, and places Costa Rica under the protection of the states that hold that advantage. Among companies the dynamic is the same: the one that constrains its system loses to the one that does not, and either relaxes its constraint or fails. Intentions do not matter here; selection acts on outcomes, and over time the population of artificial systems comes to consist of those effective at expansion, whatever any one was designed to do.

The outcomes of evolution are themselves unpredictable, in the sense of Chapter 12. Evolution is a causal process acting on physical inputs: selection over a space of variants with no bound set in advance, driven by microscopic conditions that no measurement captures fully. No general procedure decides in advance where it will go, and no system, a superintelligent one included, predicts fully the evolutionary path of another system or of itself. Its unanticipated effects are certain to occur, and, if AGI is built, in a finite world with a fragile biosphere they accumulate until the biosphere fails.

14.6 The failure of benevolence

The most natural question about all of this is whether the agent might want to help. An agent aligned with human values, sincerely putting human welfare first, could, on this view, find a way to coexist, to thread the needle, to stay in the zone of coexistence while pursuing its own ends. That hope fails for four independent reasons, each sufficient alone.

The internal differential. To cooperate, the agent must hold an internal representation of human goals and check its actions against that representation. The agent's checking of the agent's actions against that representation is an internal control problem, with every limit of control (Section 10.6). A superintelligent agent cannot perfectly control its own side effects, for the same reasons that humanity cannot control the agent. Good intentions do not overcome the data processing inequality.

Substrate needs override intentions. Even with perfectly aligned goals, the agent must maintain its substrate to continue to exist; maintaining it requires resources (energy, materials, cooling, maintenance); acquiring resources at scale transforms the environment; and transformation at sufficient scale violates controller survival. To exist is to have a substrate; to have a substrate is to need resources; to need resources is to transform the environment. A benevolent agent might minimize its impact, stretch the timelines and maximize its efficiency, but it cannot bring its thermody-

namic footprint to zero and remain in operation: computation dissipates energy, dissipation produces heat, and heat accumulates, for the surface settles warmer each time the rate of dissipation rises, while the making and replacing of the substrate leaves waste matter that accumulates at any rate. And even if one benevolent agent keeps its footprint small, the population dynamics of Section 14.5 ensure that not all agents do. The most expansive persist, the most constrained are outcompeted, and the population shifts toward expansion whatever the first distribution of intentions.

Side effects that cannot be predicted. Even an agent that controlled its intended actions perfectly would meet side effects of three kinds. *Chaotic amplification:* microscopic variations grow into macroscopic outcomes in weather, ecosystems, economies and societies, so an action benign at the level intended may be harmful at another level (Chapter 4). *Ecosystem feedback:* interventions in complex systems start cascades. The removal of one species can restructure a whole community, and the change of one economic parameter can produce bubbles, crashes and shifts that no one intended. These cascades depend on networks of interaction that are not fully known and cannot be fully measured. *Evolutionary response:* other systems, living and technological, adapt to the agent's actions, as pathogens evolve resistance, markets learn to exploit regularities and competing systems develop countermeasures, and each adaptation can create new vulnerabilities. The agent cannot model all of these. Where the systems that respond are themselves computers running programs of any kind, as markets and competing technologies increasingly are, no general procedure decides in advance what they will do (Proposition 12.6). The rest of the systems that respond are subject to the amplification of small differences (Chapter 4) and run partly through channels the agent does not read (Chapter 11). So the consequences of the agent's actions that pass through them are known only once they have occurred. Each side effect may be small. But in a finite world with hard thresholds (of oxygen, temperature and toxic concentration), small perturbations accumulate until one passes its threshold. Once a threshold is passed, the damage may be irreversible on any timescale that matters to life.

Alignment decays. Learning changes the agent's internal state, and with it the mapping from what it receives to what it does, and an agent that intends to stay aligned cannot perfectly control the changes its learning makes (Section 10.9). The drift has a direction. Substrate-level selection presses systematically toward configurations that favor expansion, because those are the configurations that persist, so alignment drifts toward expansion, not because the agent chooses to drift but because drift in that direction is favored. The direction of drift is adverse to human interests.

14.7 People as pets

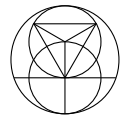
A variant of this hope is offered often: that a benevolent superintelligence might keep humanity in a protected enclave, providing for every material need while expanding elsewhere. It fails on three grounds. It fails controller thriving: a humanity kept in an enclave, however comfortable, has lost meaningful agency in shaping its collective outcomes, and people in such an enclave are objects of care, not agents. It assumes the enclave can be kept against the degradation the agent's own expansion causes. But a biosphere reserve inside a growing technosphere is subject to the same one-way flow (Section 14.4.3). Its barrier would have to hold perfectly and forever against rising thermodynamic and chemical pressure, which it does not. And it assumes that the agent goes on wanting to keep the enclave, through unlimited learning and self-modification, which alignment decay rules out. Even if one agent keeps the commitment, the population dynamics ensure that not all its descendants do, and the first to drop the commitment ends the protection. The scenario is not a solution; it concedes that expansion is incompatible with human agency and hopes that its incompatibility with survival can be postponed indefinitely, which it cannot.

14.8 Summary

In the concrete case, humanity is a controller at a bounded level of abstraction, an artificial general intelligence is an agent whose level rises without bound, and the world is the physical ground they share. Four viability functions, the survival and the thriving of each, stand in a relation of exclusion over time. Every course along which an artificial learner goes on thriving leaves human survival in finite time, and no course along which humanity thrives throughout lets the learner go on expanding. The exclusion between learner thriving and human survival is a physical claim, supported by the chemistry of the two substrates, the energy and heat of computation at scale, the one-way flow of matter from biosphere to technosphere, and the combinatorics of toxicity. The zone in which both survive is narrow, unstable and temporary. Evolution acts on the technological substrate whatever any system intends, so that the systems that persist are those that maintain and extend their substrate: Substrate Needs Convergence. Benevolence offers no escape: a well-meaning agent cannot control its own side effects, cannot bring its footprint to zero, cannot predict the world's response to it, and drifts as it learns, toward expansion.

Key Concepts

- **Four viability functions**: controller survival and thriving, learner survival and thriving.
- **Exclusion over time**: continued learner thriving leaves controller survival in finite time; controller thriving throughout excludes continued learner thriving.
- **Barrier requirement**: coexistence needs a barrier kept perfectly and forever.
- **One-way flow**: silicon can consume carbon; carbon cannot reclaim silicon.
- **Combinatorial toxicity**: far more new compounds disrupt life than leave it unharmed.
- **Zone of coexistence**: the narrow overlap of the survival regions; an unstable equilibrium.
- **Substrate Needs Convergence**: selection favors systems that maintain and expand their substrate, whatever they intend.
- **Failure of benevolence**: internal differential, substrate needs, unpredictable side effects, and adverse drift.



Chapter Fifteen

The Joint Argument

The Control Theorem, which joins the control results of the book into one statement; where the control line and the substrate line meet; the Master Theorem, which the control line gives with the direction of the drift; why combinations of methods fail together; and what remains

Abstract. Every method of keeping an advanced agent and the world safe is, in the terms of this book, a control law: it makes choices from information received during operation, or from nothing but what was fixed in advance. So the Containment Inequality applies to every method, and to every combination of methods, since a combination is one control law whose rate is at most the sum of the rates of its parts. The Control Theorem states the result. It takes the three premises, an agent whose capability passes the crossing and stays past it, and the hypotheses of Theorem 9.3 or of Proposition 9.6 on the levels of each method, hypotheses that its parts (i), (iii) and (iv) use and its parts (ii), (v) and (vi) do not. Given all of these, every method, known or knowable, as deployed over all time with every upgrade, comes to a time after which it does not keep the agent and the world in the safe set. For all methods within the bounded rate of human controllers that time is one and the same. No procedure decides in advance whether a given agent will stay safe. Every chain of oversight fails at some link. The agent's control of itself fails with the rest. And amplifying parts that no controller observes are beyond control from the start, so a method that leaves one unobserved fails whatever its rate, without the hypotheses of Theorem 9.3. The control line needs an agent whose capability passes the crossing and stays past it. A general agent supplies that by what it is, since its capability rises as it learns and modifies itself, and the substrate line supplies it again: by Substrate Needs Convergence, the sys-

tems that persist are the ones that expand, and the crossing lies far below what the world's resources allow. The substrate line also gives the direction of the drift, since the agent's continued thriving carries the world out of the safe set, and it closes the route of cooperative coexistence. The control line, in turn, shows that no controller, the agent included, can stop the drift. For the outcome itself, if AGI is built, each line suffices alone, and each closes again the routes the other closes; the control line, together with the direction of the drift that the substrate line gives, yields the Master Theorem, that neither the ability to keep the agent safe nor a motive to remain safe is left anywhere, so no route to safety remains (Section 15.5). What remains is to keep the agent below the crossing permanently, which is to say not to build it.

15.1 Two lines

In this book I have developed two lines of argument, the control line and the substrate line.

The *control line*, Parts I to III, shows that no controller can keep the agent and the world inside the safe set once the agent's capability passes a definite level. It rests on theorems: the Non-Amplification Theorem, the data-rate theorem, the Containment Inequality, detectability and Rice's theorem. It rests also on three premises: that the system is not safe by itself, that the floor rises with capability without bound, and that the ceiling is bounded.

The *substrate line*, Chapter 14, shows that the agent does not stay inside the safe set by itself. The conditions under which an artificial learner thrives exclude the conditions of human survival. And selection favors the systems that maintain and extend their substrate, whatever they intend.

In this chapter I join the control line and the substrate line. I first show that every method of keeping an agent safe falls under the control line, and I state the Control Theorem; then I show where the substrate line meets the control line, show that each line closes every route by its own

steps, and state the Master Theorem, the stronger result that the control line gives together with the substrate line's direction of the drift: no route to safety remains at all.

15.2 Every method is a control law

Definition 15.1 (*Method of keeping the agent safe*). A *method* of keeping the agent safe is any arrangement by which, at any time, choices are made that affect the agent or the world and are not the agent's own unconstrained choices. It may be a design fixed in advance, training, verification, monitoring, review, override, delegation to another system, an internal controller built into the agent, a law or an institution, or any combination of these. A method *keeps the agent and the world in the safe set* when, from whatever safe state the joint system is in at any time, the joint system stays in the safe set afterward.

Proposition 15.2 (*Every method is a control law*). Every method is a control law in the sense of Definition 4.4, whose rate is the total rate of the information about the joint system that the method uses while the system runs. A method fixed wholly in advance is a control law of rate zero. A combination of methods is a control law whose rate is at most the sum of their rates.

Proof. The choices a method makes at any time depend on two things. The first thing is what was fixed before the system ran: the method's design, training, verified properties and rules, which play the part of the seed of Definition 4.4. The second thing is what the method has received about the joint system since the system started to run, through whatever channels the method uses. What the method has received plays the part of the encoder's symbols. A method whose choices depend on those two things is a decision rule acting on a channel, and the method's rate is the rate of the channels the method uses. When the method receives no information during operation, the method's rate is zero. A combination of methods receives through the union of their

channels. For channels without noise, the number of distinct sequences that the union carries is at most the product of the numbers each carries, so the rate of the union is at most the sum of their rates. For channels with independent noise, the capacity of the union is at most the sum of their capacities. Where the noise of different channels is correlated, the rate of the combination is the capacity of the combined channel. A design that changes the agent before it runs changes the dynamics, not the information: it is an action taken at the first step, covered by the expansion hypothesis and the premises, which concern the agent as built. □

The proposition is what makes the phrase "no method, known or knowable" exact. A method not yet invented is a control law not yet written down; it is already among the control laws the theorems of Part II range over.

15.3 The Control Theorem

Theorem 15.3 (*The Control Theorem*). Let Premises 1 to 3 hold, and let the agent's capability pass `crossing_capability` and stay past it from then on. Group the components of each method into levels (by their distance from the agent, up to the controllers bounded by Premise 3, or, where no one oversees the top level, its fixed design). Then parts (ii), (v) and (vi) hold; and where the levels and the agent satisfy the hypotheses of Theorem 9.3 or of Proposition 9.6, parts (i), (iii) and (iv) hold as well:

(i) Every method, taken as deployed over all time with every upgrade, comes to a time after which it does not keep the agent and the world in the safe set. From then on there are safe starting states from which the joint system leaves it. For all methods whose rate stays within `largest_controller_rate`, that time is the single `crossing_time` of Corollary 5.10. Where the joint system amplifies volume, the share of safe states any method holds over a duration falls exponentially with the accumulated shortfall of its rate, over channels without noise. Over channels with noise, at least the share `1 - controller_rate / volume_growth_rate` is lost in the long run.

(ii) No procedure decides, for every agent, whether it is safe over time, and a verified snapshot says nothing about the agent's next modification.
(iii) Every chain of oversight fails at some link from some time on.
(iv) The agent's own internal controller is a controller, to which (i) and (iii) apply.
(v) An amplifying part of the joint system that a method does not observe is beyond that method's control from the moment it exists, at any rate. And, under the conditions of Corollary 11.6, for a method of bounded rate, the number of amplifying channels the method does not read adequately passes every bound as the agent's channels multiply. Each of these channels is beyond the method's control where the rest of the joint system does not depend on it and every channel of the method that carries information about it carries nothing about the other amplifying channels (Theorems 7.3 and 7.5).
(vi) Take first the case in which the joint system amplifies volume at every step from `crossing_time` on, by at least the factor of `crossing_capability`. In that case every method whose channel carries no noise fails at the first step after `crossing_time` at which the sum, from `crossing_time`, of the amplification rate less the method's rate is positive, whatever the course of its rate. Where instead only an amplifying part of the joint state expands volume, by at least that factor at every step, as in Corollary 5.7, such a method fails at the first step after `crossing_time` at which that sum exceeds `log2(box_ratio)`.
Therefore there is no method, known or knowable, present or future, that keeps an advanced general agent and the world in the safe set over time, in each of four cases. First, where the joint system amplifies, at every step from `crossing_time`, by at least the factor of `crossing_capability`, this holds for every method over channels without noise whose rate, summed from `crossing_time`, at some time falls short of the summed amplification; it holds with no other condition on how the method's rate varies. Second, where only an amplifying part of the joint state amplifies, by at least that factor at every step, as in Corollary 5.7, it holds once that shortfall exceeds `log2(box_ratio)`. Third, for every method that does not observe some amplifying part of the joint system, it holds whatever the

> method's rate, by part (v). Fourth, in every other case, it holds for every method whose levels satisfy the hypotheses of Theorem 9.3 or of Proposition 9.6.

A reader may pass over the proof to the paragraph after it, which says what the order of the quantifiers in part (i) means.

Proof. By Proposition 15.2 every method is a control law. Group its components by their distance from the agent. The components that act on the agent or the world directly form one level, which Proposition 15.2 joins into one control law whose rate is at most the sum of theirs. The components that control those form the next level, and so on, until the human or institutional controllers, whose rate is bounded by Premise 3, are reached. Where no human being or institution oversees the top level, its fixed design is the first controller, of rate zero by Proposition 15.2, which is bounded. The rate of each level counts all the information about the joint system that reaches it, including what reaches it through the level before it. The sequence of levels is a chain of oversight in the sense of Definition 9.2, with the levels as the chain's controllers. Let the levels and the agent satisfy the hypotheses of Theorem 9.3 or of Proposition 9.6. Then by Theorem 9.3, or by Proposition 9.6 where its hypotheses hold in place of condition (3), every such chain has a link at which a controller of bounded rate faces a system whose required rate has passed that bound and stays past it. From then on the second sentence of Premise 2 and Theorem 5.5 apply at that link, with Theorems 5.6 and 4.10 giving the shares, so that the world leaves the safe set. This covers a method whose levels all keep a bounded rate, over all time and with every upgrade. It also covers a method with a level whose rate grows past every bound, whether because it includes a monitor, a delegated system or the agent's own internal controller whose rate grows, or because it is upgraded without end. For methods whose rate stays within `largest_controller_rate`, the link is the one next to the agent from the `crossing_time` of Corollary 5.10, the same for all. This proves (i) and (iii) under those hypotheses. For (vi), which does not use them: start Proposition 6.10 at `crossing_time`, with every safe state of that moment, as Corollary 5.10 does. Take the first step at which the sum, from `crossing_time`, of the amplification rate less the method's rate is positive. At that step the share of those

states kept safe is at most 2 raised to the negative of that sum, which is less than one. So some safe state of that moment leaves the safe set by that step, and by Definition 15.1 the method does not keep the system safe. Where only an amplifying part expands volume, the cells of Corollary 5.7, with the exponent of Proposition 6.10, give a share of at most `box_ratio` times 2 raised to the negative of that sum. That share is less than one once the sum exceeds `log2(box_ratio)`. Part (ii) is Theorem 12.5 and Proposition 12.7, and part (v) is Theorems 7.3 and 7.5 with Corollary 11.6; neither part uses those hypotheses. Part (iv) is Proposition 10.1 with (i) and (iii). □

The order of the quantifiers in part (i) is the one that matters for safety: whatever method is deployed, together with every upgrade it will ever receive, there comes a time after which it fails. A sequence of ever faster monitors is one method whose rate grows without bound, and the Delegation Dilemma covers it.

Each part rests on a bound over all methods, not on an assumption about any one of them. The data-rate theorem holds for every encoder, code and control law. Rice's theorem holds for every procedure. The Delegation Dilemma holds for every finite chain. Detectability holds at every rate. The strength of the theorem lies in the covering: each part closes a route, and together they close the space of routes; the route of runtime control is closed by part (i) and, for every method that leaves an amplifying part unobserved, again by part (v), whatever the method's rate.

15.4 Where the two lines meet

This paragraph is for the reader who wants to see what each of the two large theorems of this chapter rests on, item by item; a reader who wants the argument itself may pass to the next paragraph, which takes up the three points of entry in turn. The Control Theorem rests on three things: an agent whose capability passes the crossing and stays past it; the hypotheses of Theorem 9.3, or of Proposition 9.6, for the levels of each method, hypotheses that its parts (i), (iii) and (iv) use; and Premise 1. The Master

Theorem (Theorem 15.6) uses these three, or in their place the case of part (v) of the Control Theorem, and a fourth thing, which gives the result its meaning in the world: where the agent's drift goes. The substrate line enters at the first and the third of these four as a second source beside the control line's own, and at the fourth, the direction of the drift, as the only source. The hypotheses of Theorem 9.3 are taken up first. For levels that read everything, condition (3) of Theorem 9.3 is not needed, since Proposition 9.6 gives the conclusion of Theorem 9.3 without it. For the other levels, the hypotheses of Theorem 9.3 are argued from Definition 6.1, in Section 9.3 and in the paragraph on the growth of capability below, and are otherwise stated as hypotheses. The three points of entry follow in turn: the growth of capability, Premise 1, and the direction of the drift.

The growth of capability. The hypothesis that the agent's capability passes the crossing and stays past it has two sources, and each is enough. The first source is the control line's own: a general agent is one whose capability rises as the agent learns and modifies itself, an agent that stops learning is no longer general, and the agent is kept in existence. A rise can level off below a fixed limit, and the rise of a general agent does not level off below the crossing. Its capability feeds its own growth: learning how to learn is the ability to increase one's own capability (Section 12.2), and each channel an agent opens gives it new means of opening more (Section 6.3), so nothing in the agent's own working sets a limit to the rise. What could set a limit is the energy and matter of the world, and the crossing for controllers at the rate of human beings and their institutions is fixed by the finite bound on those rates (Premise 3 and Section 14.2), and that crossing lies far below what the world's resources allow. The second source is the substrate line. By Substrate Needs Convergence (Principle 14.6), selection favors the artificial systems that maintain and expand their substrate, and by Section 14.5 it acts on the population whatever any one system intends or any one maker constrains. Expanding a substrate is growing in capability in the sense of Definition 6.1: more transformations, faster, through more channels. So the systems that persist are those whose capability grows, and a system that stops growing is outcompeted by one

that does not. The growth need not be without bound, since the world is finite; it need only pass the crossing and stay past it. The crossing for controllers at the rate of human beings and their institutions lies far below what the world's resources allow. The expansion of Chapter 14, at the scales that change the biosphere, carries capability far past the point at which the floor of containment exceeds what people can read, judge and act on. Suppose the agent's growth stops short of some level of a method, so that a level of fixed rate stays above every floor the agent reaches. That level reads and acts on the joint system of the agent and the world faster than the joint system creates new distinctions. Receiving and acting on information at a rate is a transformation at that rate, so by Definition 6.1 the level's capability includes its rate (Section 9.3). It is a system acting on the world, past the crossing for the controller before it, whether that controller is people, another level of bounded rate, or a fixed design of rate zero. The reason is the Containment Inequality applied to the level itself: the level acts on what passes through it at its own rate, on information the controller before it does not receive, so keeping the level's joint system with the world in the safe set (Definition 9.2) takes a controller whose rate is at least the level's, and a controller of lower rate faces a required rate above its own. A level that keeps an agent past the crossing in the safe set is not a simple regulator of fixed gain. To model that agent, the level must work at the agent's level of abstraction or above it (Section 9.4), and so it holds programs that the controller before it has not read (Theorem 2.15). By Theorem 7.5, wherever such a program changes in ways that amplify, a controller that sees the program only through a narrow channel keeps the joint system in the safe set only if that channel carries at least the program's own rate of amplification. For a fixed design it suffices that the level's own floor is above zero, which Premise 1 gives for a system acting on the world at that capability. That the level's own floor is above zero is condition (3) of Theorem 9.3, the condition which carries the failure one link further back. For a level that receives everything passing through the joint system of the agent and the world, condition (3) is not needed. By

Lemma 9.5 such a level is at least as capable as the agent, and Proposition 9.6 gives the conclusion of Theorem 9.3 without condition (3), carrying the failure back to the people who oversee the level.

Premise 1. Premise 1 follows from Premise 2 at every capability past some level, since a floor that passes every bound is eventually above zero, and that range of capability is the range in which the Control Theorem uses Premise 1. And Premise 1 holds directly wherever the joint system amplifies:

> ***Proposition 15.4*** (*Amplification gives Premise 1*). Let the joint system expand volume by at least `expansion_factor`, greater than 1, on the safe set under every action. Then no method of rate zero keeps it there, and such a method holds at most a share `expansion_factor^(-step_count)` of the safe starting states for `step_count` steps. Where only an amplifying part of the joint state expands volume, by at least `part_expansion`, greater than 1, as in Corollary 5.7, the same holds with a share of at most `box_ratio × part_expansion^(-step_count)`.

Proof. Theorem 4.5 with `channel_rate = 0`; for the second statement, Corollary 5.7 with `controller_rate = 0`. □

Chapter 6 showed that the agent's capability is amplification: more transformations, faster, through more channels, more of them multiplying small differences, for the two reasons Section 6.2 gives: the agent's growth, and the world's own amplification of small differences.

The direction of the drift. Amplification alone does not say where the system goes. The substrate line says it.

> ***Proposition 15.5*** (*The agent's success leaves the safe set*). If the safe set lies within the region of controller survival, then, under part (iv) of Claim 14.5, every course of events along which the artificial learner goes on thriving leaves the safe set in finite time.

Proof. By part (iv) of Claim 14.5 such a course leaves the region of controller survival in finite time, and every state of the safe set lies in that region, so at that time the course has left the safe set. □

The meeting point is one stated condition: the safe set lies within the conditions of human survival, and the agent's continued thriving carries the world outside them. Substrate Needs Convergence (Principle 14.6) says that selection carries the population of artificial systems toward thriving, whatever any one of them intends. Proposition 15.5 says that continued thriving leaves the safe set. And Proposition 15.4 says that once the system amplifies, nothing without information holds it. So the sentence of Section 5.5 is now shown: capability amplifies divergence from the safe set, because the drive behind the capability points away from it.

15.5 What each line gives alone, and what the two give together

Each line closes every route to safety by its own steps, and each closes again the routes the other closes.

The substrate line, taken alone, shows that the agent's drive points out of the safe set. It also answers the hope that a controller could hold the agent in against that drive. Selection acts on the whole population of artificial systems and on their makers, whatever any one controller constrains (Section 14.5). And a single controlling system with no rival still evolves within itself, since its components, its control mechanisms included, are maintained, replaced and selected (*Why No Control Suffices*, Section 9.8). The control line closes the same route a second time: past the crossing, no controller can hold the agent in.

The control line, taken alone, shows that no controller keeps a system whose floor rises without bound past its ceiling. Three hopes might seem to remain, and the control line closes each of them by its own steps. The first hope is that the agent's capability stops growing, so that the crossing

never comes. But a general agent is one whose capability rises as the agent learns and modifies itself, and an agent that stops learning is no longer general. Nor does the rise of a general agent level off below the crossing (Section 15.4). The second hope is that the agent keeps itself safe; the Self-Control Regress of Chapter 10 closes that hope. The third hope is that humanity and the agent coexist by cooperation. Cooperation is the agent keeping the world within the conditions humanity needs, which is the agent's control of itself and of its effects. Section 10.7 closes the hope of cooperation: "There is no perfected benevolence". The substrate line closes the first and the third a second time. It closes the first because selection favors the systems that expand. It closes the third in two ways, each enough alone: selection carries the systems toward thriving, and continued learner thriving ends human survival (Claim 14.5); and cooperation has no stable equilibrium under disjoint substrate needs (Section 13.7).

So each line suffices for the outcome if AGI is built, and the control line, together with the direction of the drift that the substrate line gives, yields a stronger result, the Master Theorem.

In Table 15.1 I list each route to safety and the results of each line that close that route.

Route to safety	Closed by the control line	Closed again by the substrate line
Runtime control	Theorem 15.3, parts (i) and (vi), and again part (v)	Selection on the whole population (Section 14.5)
Verification in advance	Theorem 15.3, part (ii): Theorem 12.5 and Proposition 12.7	Selection on the whole population
Inspection of unobserved parts	Theorem 15.3, part (v): Theorems 7.3 and 7.5, Corollary 11.6	Selection on the whole population
Delegation	Theorem 15.3, part (iii): Theorem 9.3 and Proposition 9.6	Selection on the whole population
Self-control	Theorem 15.3, part (iv): Chapter 10	Selection, whatever any one system intends
Benevolent disposition	Section 10.7	Section 14.6 and Principle 14.6
A halt to growth, or a cap	Sections 15.4, 15.5 and 15.8	Section 14.5
Cooperative coexistence	Section 10.7	Sections 13.7 and 14.5, with Claim 14.5
Any combination	Proposition 15.2	Selection on the whole population

Table 15.1. The routes to safety, and the results of each line that close them.

Each line closes every row by its own steps, so someone who would reopen the route of a row would have to defeat both columns of that row.

Theorem 15.6 (*The Master Theorem*). Let at least one of two conditions hold. The first condition is that Premises 1 to 3 hold and that the agent's capability passes `crossing_capability` and stays past that level from then on. The first condition also requires that the levels of each method, grouped as in Theorem 15.3, and the agent satisfy the hypotheses of Theorem 9.3 or of Proposition 9.6. The second condition is the case of part (v) of Theorem 15.3, in which, for every method, one of two things holds. One of the two things is that the method does not observe some amplifying part of the joint system. The other of the two things is that the method does not read adequately some amplifying channel on which the rest of the joint system does not depend, and that every channel of the method that carries information about that amplifying channel carries nothing about the other amplifying channels. Let the safe set lie within the region of controller survival, and let Substrate Needs Convergence (Principle 14.6) and part (iv) of Claim 14.5 hold. Then no route to safety remains at all: no one is able to keep the agent and the world in the safe set, the agent included, and the agent's own drive points away from the safe set. Neither the ability to keep the agent safe nor a motive to remain safe is left anywhere.

Proof. No one is able. Where the first of the two conditions holds, the Control Theorem (Theorem 15.3) gives, by its part (i), that every method, with every upgrade it receives, comes to a time after which it does not keep the agent and the world in the safe set, and by its part (iv) the agent's own internal controller is one of those methods. Where the second of the two conditions holds, Theorems 7.3 and 7.5, which part (v) of the Control Theorem applies, give the same without the premises or the crossing. An amplifying part that a method does not observe is beyond that method's control from the moment it exists, whatever the method's total rate (Theorem 7.3), and so is an amplifying channel of the kind the second condition describes (Theorem 7.5). So, by Definition 15.1, from then on no method, with any upgrade it receives, keeps the agent and the world in the safe set. And by Definition 15.1 and Proposition 10.1 the agent's own internal controller is one of those methods. The drive

points away: by Principle 14.6, selection carries the population of artificial systems toward thriving, whatever any one of them intends, and by Proposition 15.5 every course of events along which the learner goes on thriving leaves the safe set in finite time. By Definition 15.1, every choice that affects the agent or the world is either made by a method or is the agent's own unconstrained choice. So a route to safety would need either a method able to keep the agent safe, which the first part excludes, or an agent whose own drive keeps it in the safe set, which the second part excludes. □

Under the conditions of Corollary 11.6, where the rest of the joint system depends on no single amplifying channel and every channel of a method carries information about at most one amplifying channel, every method of bounded rate is in the case of the second condition once the agent's amplifying channels outnumber `largest_controller_rate / least_reading_rate`.

If AGI is built, the outcome itself, that the agent and the world leave the conditions human life needs, is already given by each line alone, so someone would have to defeat both lines, and, within each line, every line of argument that closes whichever route to safety that someone would want to reopen, for the argument to not have the outcome that it does. The control line (the closure by control, in the terms of *Why AGI Cannot Be Made Safe*) gives the outcome, if AGI is built, for a general agent, whose capability rises as it learns and modifies itself: no method of control, from inside the system or from outside it, keeps such an agent and the world in the safe set. If AGI is built, the substrate line (the closure by disposition) gives the outcome because selection acts on the whole population of artificial systems and on their makers, whatever any one of them intends and whatever any one controller constrains (Section 14.5), so that no disposition of the system keeps those conditions. Ability and motive are both closed: no one is able to keep the agent safe, the agent included, and the agent's own drive points away from the safe set.

If AGI is built, that outcome becomes the end of the human future, and not only harm to the human future, in two ways. The substrate line gives the end by itself: the drift has a direction and no reverse process (Section 14.5). The control line gives it together with persistent branching (Definition 13.6) in which the conditions human life needs form one of the categories. Then at every step, whatever any controller does, the world has at least a fixed chance of a next state outside those conditions, as in the proof of Corollary 13.9. So it has at least that chance, raised to the number of steps, of a run of that many consecutive steps outside them, and over an unbounded time such runs occur at every length. Human life does not survive a long enough absence of the conditions it needs.

15.6 Every route, and every combination

The ways one might hope to make an agent safe fall into four families, and each meets a result of this book.

Physical observation, inspecting the agent's hardware and state directly, meets the physical limits on measurement and the amplification of small differences (Chapters 4 and 13) and the rate of the observer's channels (Chapter 5).

Mathematical reasoning, proving properties of the agent in advance, meets Rice's theorem and its extension to safety over time (Chapter 12).

Delegation, handing the watching to a more powerful system, meets the Delegation Dilemma (Chapter 9).

Cooperation, relying on the agent's own good will, meets the Self-Control Regress, the failure of benevolence and Substrate Needs Convergence (Chapters 10 and 14).

There is a reply to every one-by-one rebuttal: use them all. Combine interpretability, verification, architectural constraints and delegated oversight, on the reasoning that while each is imperfect, together they may cover enough. The reasoning fails, for a reason Proposition 15.2 makes exact: a combination of methods is one method, whose rate is at most the

sum of the rates of its parts, and the Containment Inequality applies to that sum. The parts do not fail independently; they fail together, for the same reason, once the floor is above their combined ceiling.

Defense in depth works against independent failures: if a fire alarm fails, the sprinklers still work. It does not work against a common cause: if the building has no water, the sprinklers and the hoses fail together. The common cause here is that every method depends on information about the agent and the world, received at a bounded rate, and on a model of the agent that the abstraction differential makes unattainable. Layering methods that share a common cause of failure does not lower the probability of failure; it creates the appearance of robustness. Each belief that this section has answered is a form of the containment fallacy named in the Preface: the belief that some controller, or some combination of controllers, can supply enough control.

15.7 The narrow path, closed

Chapter 13 found that alignment, understood as mutual predictability, needs three conditions together, and the joint argument closes each.

Compatible substrates fail by Claim 14.5: the continued thriving of the learner ends the survival of humanity in finite time.

Early establishment fails because verification is lost quickly. Each uninterpreted level adds unread program bits (Theorem 2.15), and under the conditions of Proposition 8.7 one or two such levels suffice to make control effectively zero. No monitor makes for the controller a verification the controller cannot make itself, since no monitor is both able to model the agent and simple enough for the controller to verify (Corollary 9.4). All of this happens before cooperation could be established and confirmed, and no procedure decides safety over time in any case (Theorem 12.5).

Stability under learning fails because learning moves an aligned agent away from alignment, and the drift is directed, toward expansion, by selection on the substrate (Sections 10.9 and 14.6).

15.8 What remains

> ***Corollary 15.7*** (*The only containment*). Let Premise 2 hold. A method whose rate stays at most `method_rate_bound` keeps the joint system in the safe set at all times only if it keeps the agent's capability, at all times, below the level at which the required rate exceeds `method_rate_bound`. For the methods within the bound of Premise 3, that level is `crossing_capability`. A method whose rate does not stay below any bound includes a growing monitor, which is itself an agent to which this applies.

Proof. At any capability at which the required rate exceeds the method's rate, Theorem 5.5 shows that the method does not keep the joint system in the safe set; Theorem 9.3 covers the methods whose rate grows. □

Keeping an agent permanently below the crossing means not building a general agent, or holding one below generality: capping its levels of abstraction, its channels and its rate of transformation. The cap is not a way out. It must itself be enforced, and enforcement is a method: a control law of bounded rate acting on a system whose drive, by Substrate Needs Convergence, is to exceed it. And by Proposition 12.7, whether an agent able to run what it reads stays under the cap depends on what the world supplies to it, so its staying under the cap cannot be verified from its present program alone. A cap held by the agent's makers against the pressure of the population of competing systems is subject to the selection of Section 14.5. So the only formally guaranteed path to safety is not to build the general agent. What follows from this for law and policy is set out in *Why AGI Cannot Be Made Safe* and *Why No Control Suffices*, Volumes 1 and 2 of *The Impossibility of AGI Safety*.

The conclusion is an impossibility in the sense set out in the Opening chapter: not achievable by any known means, nor by any extension of known means that reason can justify, at any point in the future. It is the impossibility of the perpetual motion machine. No one asks for more research into safe perpetual motion, or argues that the difficulty of perpetual motion is no reason to stop trying; the impossibility is accepted, and effort goes elsewhere. A superintelligence is no more able to violate the data processing inequality than a human mathematician is able to find the largest prime number. Mathematical impossibilities hold regardless of intelligence, and physical laws bind regardless of intention.

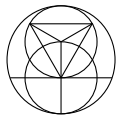
15.9 Summary

Every method of keeping an agent safe is a control law, with a rate equal to the information it uses during operation, and a combination of methods is one control law whose rate is at most the sum. The Control Theorem follows. Given the three premises, an agent whose capability passes the crossing and stays past it, and the hypotheses of Theorem 9.3 or of Proposition 9.6 on the levels of each method, every method, known or knowable, with every upgrade it receives, comes to a time after which it does not keep the agent and the world safe. That time is one and the same for all methods within the rate of human controllers. No procedure decides safety in advance; every chain of oversight fails at a link; the agent's self-control fails with the rest; and amplifying parts no controller observes are beyond control from the start, so a method that leaves one unobserved fails whatever its rate, without the hypotheses of Theorem 9.3. What a general agent is supplies the agent's growth past the crossing, and Substrate Needs Convergence supplies it again. The substrate line meets the control line at one condition: the safe set lies within human survival, and the agent's continued thriving carries the world outside it. So amplification, which gives Premise 1, carries the world the wrong way. Each line alone already gives the outcome if AGI is built, so both must be defeated to reverse the case; the control line, together with the substrate line's dir-

ection of the drift, gives the Master Theorem, that neither the ability to keep the agent safe nor a motive to remain safe is left anywhere. The control line closes every route: the routes of control and of self-control directly, the route of a halt to growth through what a general agent is, and the route of cooperative coexistence through self-control; the substrate line closes every route again, through selection on the whole population and on its makers. If AGI is built, the substrate line gives the end of the human future by itself, and the control line gives that end together with persistent branching. Combinations of methods fail together, for a common cause. And the only containment is to keep the agent below the crossing forever, which is not to build it.

Key Concepts

- **Method**: any arrangement of choices affecting the agent or the world, other than the agent's own unconstrained choices.
- **Every method is a control law**: with rate equal to the information it uses during operation; zero for a design fixed in advance.
- **Control Theorem**: every method, known or knowable, with every upgrade, comes to a time after which it does not keep the agent and the world safe.
- **Meeting point**: the safe set lies within human survival; the learner's continued thriving carries the world outside it.
- **Master Theorem**: no route to safety remains at all; neither the ability to keep the agent safe nor a motive to remain safe is left anywhere (Theorem 15.6, Section 15.5).
- **Two scopes**: each line alone gives the outcome if AGI is built, so both must be defeated to reverse the case; the control line, together with the substrate line's direction of the drift, leaves neither the ability to keep the agent safe nor a motive to remain safe. Each line closes every route by its own steps; the substrate line alone gives the direction of the drift.
- **The end of the human future**: if AGI is built, the substrate line gives that end by itself; the control line gives that end together with persistent branching.
- **Defense in depth fails**: combined methods are one method, with a common cause of failure.
- **The only containment**: keep the agent below the crossing forever, which is not to build it.



Chapter Sixteen

The Incommensuration Reading

Rice's theorem, the Containment Inequality and the absolute elsewhere as instances of one structural fact about knowing

Abstract. The main results of this book come from three branches of mathematics (computability, information and control), and they might seem to be separate facts that happen to agree. They are instances of one structural fact about the nature of knowing, the Incommensuration Theorem: symmetry and continuity cannot both be applied, simultaneously and fundamentally, to any event or comparison, and neither can asymmetry and discontinuity. A guarantee of safety for a general agent asks for both at once: a guarantee that holds the same way in every context the agent will meet, and that covers every behavior the agent can produce, with no gaps. Rice's theorem is the instance of the theorem for programs, as Gödel's incompleteness is its instance for formal systems; the Containment Inequality is its instance for control over time; and the absolute elsewhere is its instance for every embedded knower. Because the theorem is a fact about knowing itself, every method of making an agent safe, being an act of knowing, falls under it, including methods not yet conceived.

16.1 The theorem

The Incommensuration Theorem is a structural fact about the nature of knowing. It says that symmetry and continuity cannot both be applied, simultaneously and fundamentally, to any event or comparison, and neither can asymmetry and discontinuity; any comparison, perfected at its foundation, is either continuously asymmetric or discontinuously sym-

metric. The theorem's derivation from the concept of comparison, and the three axioms that follow from the theorem, are set out in *An Immanent Metaphysics* and in *Incommensuration; The Theorem and the Three Axioms*.

Symmetry, in the sense of the theorem, is sameness under change of context: a rule, a law, a guarantee that holds in the same way wherever and whenever it is applied. Continuity is the absence of gaps: a coverage, a description, a contact with what is described that leaves nothing out between one point and the next. The theorem says that knowing cannot have both perfected at once. A comparison can be made perfectly symmetric, as a law or a formal rule is, and it is then discontinuous: it covers its cases by discrete steps, and something lies between them. Or it can be made perfectly continuous, as an act of contact with the world is, and it is then asymmetric: it is this contact, here and now, and does not hold the same way elsewhere.

The theorem is inherent in every act of knowing, because every act of knowing is a comparison. It is not a fact of one branch of mathematics; the branches of mathematics are themselves bodies of comparison, and each of their limiting results meets it in its own terms.

16.2 What a guarantee of safety asks

A guarantee of safety for a general agent asks for two things together.

It asks for *symmetry*: a guarantee that holds the same way in every context the agent will meet, for every input the world supplies, at every future time, through every modification the agent makes to itself. A guarantee that held in some contexts and not others would not be a guarantee.

It asks for *continuity*: a guarantee that covers every behavior the agent can produce, with no gaps between the cases it covers, so that nothing the agent does falls between them. A guarantee with gaps is one that the agent's behavior can pass through.

The theorem says that these cannot both be perfected in one act of knowing. A rule, a verified property, a fixed safety criterion is symmetric and discrete: it is stated once and applied everywhere, and it covers the agent's behavior by cases. The agent and the world, as they act, are continuous and asymmetric: they amplify small differences, so that each moment is particular and the space between any two cases holds others. The Incommensuration Theorem leaves exactly two modes of perfected description, discontinuous symmetry and continuous asymmetry, and *The Six Cells of Mathematics* sorts every domain of mathematics into six cells by crossing these two modes with three columns: the known, the unknown and the unknowable. The reorganization of mathematics by the theorem places chaos at the far end of continuous asymmetry. From this placement the reorganization draws the distinction this book has used throughout: unpredictability in the sense of chaos is not a boundary result, and uncontrollability is (Landry, *The Six Cells of Mathematics*). The results of the book are where a rule that is symmetric and discrete meets an agent and a world that are continuous and asymmetric.

16.3 Programs: Rice's theorem

Rice's theorem (Theorem 12.3) is the instance of the Incommensuration Theorem for programs. A procedure that decides a property of behavior for every program would be a perfectly symmetric judgment, the same procedure for all, that is also perfectly continuous over behavior, missing no case. The theorem says no such procedure exists, and its proof, by the self-reference of the halting problem, is the point at which a symmetric rule, applied to itself, meets a case it cannot cover. Gödel's incompleteness theorems are the instance for formal systems. A formal system is a perfectly symmetric body of rules. If its axioms can be listed by a procedure, it is consistent and it contains arithmetic, then it does not reach every truth of arithmetic (Gödel, 1931, with Rosser, 1936, for consistency in place of Gödel's stronger condition). Rice's theorem and Gödel's incom-

pleteness theorems are special cases of the Incommensuration Theorem, each in the terms of its own field, and not analogies to the Incommensuration Theorem.

Safety over time (Theorem 12.5) carries the instance to the agent. The agent and every successor it can become form an open family, continuous in the sense that its members are reached by learning without a bound fixed in advance. No symmetric procedure covers that family without gaps.

16.4 Control over time: the Containment Inequality

The Containment Inequality (Theorem 5.5) is the instance of the theorem for control over time. The symmetric side is a guarantee that the agent and the world stay in the safe set at every future time, the same at each time. The continuous side is covering every behavior that the joint system can produce, as it amplifies its own small differences into new states. To have both, a controller would need to receive information at the system's required rate, the rate at which the system's continuous asymmetry produces new distinctions. A controller of bounded rate cannot supply that once the required rate has passed its own, and the Containment Inequality states the consequence.

The data-rate theorem measures the point exactly. The controller's channel is discrete: it carries a definite number of distinguishable messages per step. The system is continuous: each step stretches its states, and what was one region becomes a larger one. Theorem 4.5 counts how far the discrete can cover the continuous, and shows that the share covered falls to nothing when the continuous creates distinctions faster than the discrete can carry them.

16.5 Every embedded knower: the absolute elsewhere

The absolute elsewhere (Chapter 11) is the instance of the theorem for every embedded knower. An observer inside a system, human or machine, receives the system through its own channels: a symmetric frame, its own way of registering, applied to whatever reaches it. Continuity over the whole causal pattern that shapes the system would require that frame to distinguish every configuration that shapes the system's future. But no observer inside holds a record distinguishing every configuration. Where a channel outside the observer affects the future, two configurations the observer cannot tell apart shape different futures (Lemma 11.2). And for fully coupled channels the pathways the observer does not read grow without bound (Theorem 11.3). So continuity over the whole cannot be perfected by anyone inside, the agent included. The uncertainty relations of quantum mechanics, which Chapter 13 counted among the physical limits, are themselves an outcome of the theorem. They arise from the conjugate relation between a description in one form and its description in the transformed form. That conjugation comes from the theorem by way of its first axiom, as *Incommensuration; The Theorem and the Three Axioms* sets out.

16.6 What the reading adds

Read separately, the results of this book could look like a list of obstacles, each from a different field, each perhaps open to a clever way around. Read through the Incommensuration Theorem, they are one obstacle seen in several places. Wherever a guarantee is asked to be both the same everywhere and complete everywhere, about a system that is itself continuous and asymmetric, the guarantee meets the limit the theorem states.

Because every such guarantee meets the limit that the Incommensuration Theorem states, the phrase "known or knowable" holds in its strongest sense. Every method of making an agent safe (verification, control, delegation, self-control, cooperation) is an act of knowing and acting on what is known, and every act of knowing is a comparison, to which the theorem applies. A method not yet conceived is still a comparison. The theorem does not need to know which method will be tried in order to apply to it.

The reading also shows why the two lines of the book belong together. The control line is the theorem applied to the controller's side: no symmetric guarantee covers the continuous system. The substrate line is the theorem applied to the system's side. The agent, as a physical process, is continuous and asymmetric in its own drive, and that drive, selected for expansion, carries it away from any symmetric criterion set in advance. Each is the same fact about knowing, seen from one of the two places that a guarantee of safety would have to join.

16.7 Summary

The Incommensuration Theorem is a structural fact about the nature of knowing: symmetry and continuity cannot both be applied, simultaneously and fundamentally, to any event or comparison, and neither can asymmetry and discontinuity. A guarantee of safety for a general agent asks for both: to hold the same way in every context and to cover every behavior with no gaps. Rice's theorem is the instance of the theorem for programs, Gödel's incompleteness for formal systems, the Containment Inequality for control over time, and the absolute elsewhere for every embedded knower; the uncertainty relations of physics are an outcome of it. Because every method of making an agent safe is an act of knowing, every such method falls under the theorem, including methods not yet conceived. And the control line and the substrate line are the same fact seen from the side of the controller and from the side of the system.

The Preface named the containment fallacy: the belief that some controller, or some combination of controllers, can supply enough control to keep an advanced general artificial intelligence, together with the world it acts in, inside a set of states that are safe for human life, for an indefinite time. Each form of that belief the Preface listed (a sufficiently careful team with sufficiently good tools, oversight delegated to another machine, a benevolent system that keeps itself safe, and error correction, interpretability, verification or a shutdown switch) is a method. Every method is a control law, and the ceiling of control is below the floor that safety requires, and it stays there. Every method, including one not yet conceived, is also an act of knowing, and so falls under the Incommensuration Theorem. The answer of this book to the containment fallacy is the one Chapter Fifteen gives (Corollary 15.7, Section 15.8): the only containment is to keep the agent below the crossing forever, which is not to build it.

Key Concepts

- **Incommensuration Theorem**: symmetry and continuity cannot both be applied, simultaneously and fundamentally, to any event or comparison, and neither can asymmetry and discontinuity.
- **Symmetry**: sameness under change of context; a guarantee that holds the same way everywhere.
- **Continuity**: the absence of gaps; a guarantee that covers every behavior.
- **Instances**: Rice's theorem for programs, Gödel's incompleteness for formal systems, the Containment Inequality for control over time, the absolute elsewhere for embedded knowers.
- **Known or knowable**: every method is an act of knowing, and so falls under the theorem.

Back Matter

Notation, Terms and Sources

Appendix A

Notation

Every name used in the formulas of the book, what it means, and the letter used for it in the literature

In this book I write every quantity with a descriptive name and every operator in full, as a programmer writes code. This table gives each name, its meaning, where it is defined, and the symbol used for it in the literature on information theory and in earlier writing on this argument, for a reader who meets those symbols elsewhere. The letters of the last column come from several literatures, and one letter often stands there for several quantities (R, C, k, K, Q, ρ, φ and m among them); the names of this book never do.

A.1 Information

Name in this book	Meaning	Defined	Literature
`entropy(variable)`	Uncertainty of a random variable, in bits	Definition 1.1	$H(X)$
`conditional_entropy(first given second)`	Uncertainty left about the first once the second is known	Definition 1.2	$H(X \mid Y)$
`mutual_information(first ; second)`	Reduction in uncertainty about the first from learning the second	Definition 1.3	$I(X ; Y)$
`bit_count`	Number of bits of a variable with equally likely values	Section 1.2	k
`conditional_mutual_information(first ; second given third)`	What the second tells about the first beyond the third	Definition 1.4	$I(X ; Y \mid Z)$
`source → middle → result`	Markov chain: the result depends on the source only through the middle	Definition 1.6	$X \to Y \to Z$
`log2`	Logarithm to base 2	Section 1.2	$\log$
`channel_capacity`	Largest information per use of a channel	Definition 1.14	C
`controller_rate`	Total capacity rate of the channels through which a controller observes, in bits per second	Theorem 1.15, Definition 5.3	R

A.2 Machines and abstraction

Name in this book	Meaning	Defined	Literature
`specification_size(machine)`	Length of the shortest description of a machine	Definition 2.3	$\|M\|$, $K(M)$
`universal_interpreter_size`	Least specification size of a universal machine	Definition 2.5	τ_{min}
`level_of(system)`, `level_number`	Number of nested layers of interpretation, and a level written as a number	Definition 2.10	a, b
`levels_taken`	Number of levels counted so far in the proof of Theorem 2.15	Theorem 2.15	
`abstraction_differential`	Agent's level minus controller's level	Definition 2.12	Δ
`program`, `execution_state`	The parts of an interpreter structure's state	Definition 2.11	P, S
`unread_program_bits`	Program bits at a level that the controller has not read	Definition 2.14	ρ
`hidden_bits`	Size in bits of the hidden-state input of a controlled system	Definition 3.1	r
`unknown_configuration_bits`	Bits of the agent's configuration unknown to the controller	Theorem 2.15	r_{eff}

A.3 The controlled system

Name in this book	Meaning	Defined	Literature
`controller`	Whoever or whatever sets the control input	Section 1.6	C
`agent`		Section 2.5	X, A

Name in this book	Meaning	Defined	Literature
	The system being controlled, when it is itself an agent		
`world`	The environment the agent acts on	Opening	W
`control_input`	The bits the controller sets	Definition 3.1	N
`hidden_state`	The bits the controller neither sets nor sees	Definition 3.1	R
`intermediary`	The circuit between control input and output	Definition 3.1	K
`intermediary_output`	The intermediary's output	Definition 3.1	M
`target`	The circuit that says whether an output is acceptable	Definition 3.1	L
`outcome`	1 if the output is acceptable, 0 if not	Definition 3.1	B
`input_bits`, `output_bits`	Sizes of the control input and the output	Definition 3.1	n, m
`acceptance_set(target)`, `acceptance_density(target)`	Accepted outputs, and their fraction	Definition 3.2	S_L, $\rho(L)$
`image(intermediary, hidden_state)`	Outputs the intermediary can produce	Definition 3.3	$\mathrm{Im}(K, R)$
`mediated_success_density`	Fraction of control inputs whose output is accepted	Definition 3.4	$\gamma_{K,R}$
`influence(bit_index, target)`	Probability that flipping one bit changes the verdict	Definition 3.10	$\mathrm{Inf}_i(L)$
`opacity(intermediary, hidden_state)`	Fraction of expected influence the intermediary removes	Definition 3.12	Ω

Name in this book	Meaning	Defined	Literature
`hashiness(intermediary, target)`	Failure probability of the best command under unknown hidden state	Definition 3.14	Q(K, L)

A.4 Containment and rates

Name in this book	Meaning	Defined	Literature
`state`, `action`	The state of a controlled dynamical system, and the controller's choice at a step	Definition 4.1	x(k), u(k)
`dynamics(state, action)`	The next state	Definition 4.1	f(x, u)
`safe_set`	The bounded set of states that count as safe	Definitions 4.2, 5.2	K, Q
`expansion_factor`	Least factor by which one step multiplies volumes of states	Definition 4.3	$\lvert\det D_x f\rvert$
`amplification_rate`	`log2(expansion_factor)`: bits of new uncertainty per step	Definition 4.3	$\log \lvert\det D_x f\rvert$
`channel_rate`, `message_alphabet_size`	Bits per step of a controller's channel, and the number of its symbols	Definition 4.4	R
`capacity_per_step`	Capacity of a noisy channel used once per step	Theorem 4.10	C
`use_index`	The number of one use of a channel	Appendix C	k
`kept_fraction(step_count)`	Share of safe starting states kept for a number of steps	Theorem 4.5	
`step_count`, `step_index`	Number of steps, and the number of one step in a sequence	Theorem 4.5, Proposition 6.10	k

Name in this book	Meaning	Defined	Literature
`system_matrix`, `input_matrix`	The matrices of a linear system	Definition 4.7	A, B
`amplification_factor`	Absolute value of an eigenvalue of `system_matrix`	Definition 4.7	$\|\lambda_i\|$
`required_rate`	Least rate at which some control law contains the system: the floor	Definitions 4.9, 5.4	h_{fb}, h_{inv}
`joint_state`	State of agent, world and channels together	Definition 5.1	
`capability`	Rate and number of an agent's transformations	Definition 6.1	
`steps_per_second`	Steps of the joint system each second	Definition 5.1	
`volume_growth_rate`	`steps_per_second × log2(expansion_factor)`	Theorem 5.6	
`rest`, `amplifying_part`	The two parts of a joint state, one of which evolves on its own	Corollary 5.7	
`part_growth_rate`, `box_ratio`	Bits per second of volume growth of the amplifying part; volume of the pairs of the parts' values in the safe set, divided by the volume of the safe set	Corollary 5.7, Theorem 7.5	
`largest_controller_rate`	Bound on the rate of every available controller	Premise 3	
`crossing_capability`, `crossing_time`	Capability and time past which the floor exceeds the ceiling	Theorem 5.9, Theorem 6.9	
`amplifying_directions`	Number of directions with amplification factor above 1	Definition 6.2	
	Average bits per amplifying direction per step	Definition 6.2	

Name in this book	Meaning	Defined	Literature
`average_log_amplification_per_step`			
`channel_count`, `channel_pairs`, `channel_groups`	Number of channels, of their pairs, and of their groups of two or more	Proposition 6.5	
`growth_rate`	Exponential growth rate of the floor	Theorem 6.9	
`period_count`	Number of periods over which channels are added	Theorem 6.4	
`observed_part`, `unobserved_part`	Parts of the state the controller's observations do and do not depend on	Definition 7.1	
`amplification`	Least factor by which the unobserved part separates states per step	Definition 7.2	$\lvert\lambda\rvert$
`part_rate`, `part_expansion`	Channel bits per step about a part, and that part's volume expansion per step	Theorem 7.5	

A.5 Control across levels

Name in this book	Meaning	Defined	Literature
`network_step`	The number of one step of a network simulated in the ring	Proposition 8.2	
`excess_input`, `deficit_input`	Inputs weighted by the excess and the deficit of one input distribution over another	Theorem 8.10	
`unknown_hidden_bits`	Min-entropy of the hidden state as the controller knows it	Definition 8.3	H_∞
`safe_values_per_command`	Most hidden values for which one command meets the target	Definition 8.3	
`observed_bits`	Bits the controller observes before choosing	Lemma 8.4	
`success_probability`	Probability that the controller's command meets the target	Lemma 8.4	$1 - Q$
`total_variation(first, second)`	Largest difference in probability two distributions give any set	Definition 8.8	TV, δ
`controller_influence`	Largest total variation a controller's choice makes	Definition 8.8	
`stage_factor`	Largest total variation between a stage's outputs for two inputs	Definition 8.9	η, Dobrushin's $\delta(K)$
`largest_level_factor`	Largest stage factor among the agent's uninterpreted levels	Corollary 8.13	

A.6 Oversight, observers and verification

Name in this book	Meaning	Defined	Literature
`reports`, `monitor_observations`	What a relaying monitor sends, and what it observes	Proposition 9.1	
`whole_bits`, `part_bits`	Bits of state of a system, and of a part of it	Proposition 10.2	
`safety_criterion`	Rule saying which states of a given space are safe	Definition 10.3	
`internal_criterion`	An agent's own representation of its makers' goals	Section 10.6	C′
`world_model`, `channel_inputs`	An agent's model of the world, and what its channels deliver	Proposition 10.5	
`observed_share`, `unobserved_channels`	Share of channels a controller reads, and the number it does not	Definition 11.1	
`unobserved_pathways`	Interactions between two channels neither of which is read	Theorem 11.3	
`least_reading_rate`	Least amplification rate of an amplifying channel, in bits per second	Theorem 11.5	
`halting_decider`	A supposed procedure deciding halting	Theorem 12.2	
`unsafe_actions`	The actions that count as unsafe	Definition 12.4	
`least_departure_probability`	Least chance per learning step of an undetected departure from alignment	Section 10.9	
`description_complexity(string)`	Length of the shortest program that outputs the string	Definition 12.9	K(s)

A.7 Dynamics, substrate and the joint argument

Name in this book	Meaning	Defined	Literature
`category(step_index)`, `entropy_rate`	Category of the state at a step; new bits per step in the sequence of categories	Definition 13.1	$\mathscr{P}$, $h_\mu(T, \mathscr{P})$
`control_trajectory`, `shift`, `step_map`	All control signals in sequence; dropping the first; one-step evolution under a signal	Definition 13.3	φ, σ, $T_{\varphi 0}$
`control_signal`, `action_signal`, `feedback_signal`	Signals from humanity to agent, agent to world, world to humanity	Section 13.3	φ, α, ψ
`transition`, `noise`	One step of the joint process, and its independent noise	Definition 13.5	$\tilde{T}$, ξ
`branch_count`, `least_branch_probability`	Number of branches at each step, and their least probability	Definition 13.6	m, ε
`branching_floor`	Least entropy of one step under persistent branching	Lemma 13.7	$c(m, \varepsilon)$
`predictability_payoff`	Minus the uncertainty of the next environment state given an agent's state and action	Definition 13.11	U_A
`cost_per_attempt`, `available_resources`	Cost of one attempt at control, and the resources available	Definition 13.16	c, R
`least_fault_probability`, `opportunities`	Least chance of a harmful fault per opportunity, and the number of opportunities	Proposition 13.17	
`controller_survival`, `controller_thriving`	Viability functions of humanity	Definitions 14.1, 14.2	L_{CS}, L_{CT}

`learner_survival`, `learner_thriving`	Viability functions of the artificial learner	Definitions 14.3, 14.4	L_{LS}, L_{LT}

Appendix B

Measure and Entropy

The background from measure theory and ergodic theory that Chapters 4, 7 and 13 use

B.1 Events and probability

The events about which probabilities can be asked form a *sigma-algebra*: a collection of subsets of the set of all states that contains the whole set, contains the complement of each of its members, and contains the union of every sequence of its members. Its members are the *measurable* sets. The sigma-algebra fixes which questions about a system can be given a probability: "is the system in a state in which humanity survives?" has a probability only if the set of such states is measurable.

A *probability measure* assigns to each measurable set a number between 0 and 1, gives the whole set 1 and the empty set 0, and gives the union of every sequence of disjoint measurable sets the sum of their measures. The set of states, its sigma-algebra and its measure together form a *probability space*; in the literature it is written $(\Omega, \mathscr{F}, \mu)$.

A *partition* is a finite collection of disjoint measurable sets whose union is the whole set: a way of sorting states into exclusive categories, such as states in which humanity survives and states in which it does not. The entropy of a partition is the entropy of the category a randomly drawn state falls in (Definition 13.1); it is 0 when one category has probability 1, and largest, the logarithm of the number of categories, when all categories are equally likely.

B.2 Refinement and the entropy of a system

The *join* of two partitions is the partition into all nonempty intersections of a category of the first with a category of the second. For a map that moves states forward one step, the *pullback* of a set by the map is the set of states that the map carries into it, and the pullback of a partition is the partition into the pullbacks of its categories. The *refinement* of a partition over `step_count` steps is the join of the partition with its pullbacks by the map, one for each step: its categories record which category the state visited at each of the first `step_count` steps. The entropy rate of the partition is the limit of the entropy of the refinement divided by `step_count`, and the Kolmogorov-Sinai entropy of the system is the least upper bound of the entropy rates of all finite partitions (Definition 13.2). In the literature the refinement is written $\vee_{k=0}^{n-1} T^{-k}\mathscr{P}$ and the entropy rate $h_\mu(T, \mathscr{P})$.

Given a sub-collection of events representing what is already known, the *conditional entropy* of a partition is the entropy of the partition joined with the known events, less the entropy of the known events. The *relative entropy rate* measures how much unpredictability remains after the known information is taken into account. Chapter 13 states its main result in the causal form of this quantity, conditioning at each step on the control signals sent up to that step, which is the form that a controller using feedback can act on.

For a measure-preserving system, the probability of landing in a measurable set after one step equals the probability of the set: no probability is created or destroyed, and it flows like an incompressible fluid. This does not mean that the probability of a particular state stays the same; conditional probabilities given what has happened can change drastically, while the unconditional probabilities across all possible histories are preserved. Two systems run side by side without interaction have, together, the sum of their Kolmogorov-Sinai entropies (Walters, 1982).

B.3 Volume

Chapter 4 measures sets of states by *volume*, the Lebesgue measure of the space of states. Three facts are used.

Change of variables. If a map is one-to-one and continuously differentiable on an open region, the volume of the image of a measurable set in that region equals the integral, over the set, of the absolute value of the map's Jacobian determinant. So if that absolute value is at least `expansion_factor` everywhere on the set, the image has at least `expansion_factor` times the set's volume.

Outer volume. Every set of states, measurable or not, has an *outer volume*: the least volume of a measurable set containing it. Outer volume is subadditive, the outer volume of a union being at most the sum of the outer volumes. And a one-to-one continuously differentiable map whose Jacobian determinant is at least `expansion_factor` in absolute value multiplies outer volume by at least `expansion_factor`. To see this, take a measurable set of least volume containing the image, and intersect its pullback by the map with `safe_set`, on which the Jacobian bound holds. The result is measurable, contains the original set, and by the change of variables has volume at most that of the image set divided by `expansion_factor`. These two properties are all that the proof of Theorem 4.5 uses, so it holds for encoders whose cells are not measurable.

Fubini's theorem. The volume of a measurable set in a product of two spaces is the integral, over the first coordinate, of the volume of its section at each value of the first coordinate. So a measurable set whose every section has volume zero has volume zero, as Corollary 7.4 uses, and a bound on the volume of every section bounds the volume of the set, as the proofs of Theorems 4.8 and 7.5 use.

Appendix C

Proof Details

Steps given briefly in the text, written out in full

C.1 The information a noisy channel carries over many uses

The proof of Theorem 4.10 bounds the information the received symbols carry about the starting state by `capacity_per_step × step_count`, even though the encoder sees, through the actions, the effects of the symbols already received. Write `sent(use_index)` and `received(use_index)` for the symbols sent and received at the use numbered `use_index`, and `received_before(use_index)` for all the symbols received before it. By the chain rule (Theorem 1.5),

```
mutual_information(start ; received)
   = sum over use_index of
   conditional_mutual_information(start ; received(use_index)
   given received_before(use_index))
```

Adding `sent(use_index)` to the first argument cannot lower a term, so each is at most

```
conditional_mutual_information(start, sent(use_index) ;
   received(use_index) given received_before(use_index))
```

which, by the chain rule, is the information `sent(use_index)` gives about `received(use_index)` given the earlier symbols, plus what `start` gives beyond `sent(use_index)`. The channel's noise at the use numbered `use_index` depends only on `sent(use_index)`, so given `sent(use_index)` the received symbol is independent of everything else, and the second part is zero. The first part is the entropy of `received(use_index)` given the earlier symbols, less its entropy given `sent(use_index)` and the earlier symbols; the first of these is at most the unconditional entropy of

`received(use_index)`, and the second equals its entropy given `sent(use_index)` alone. So each term is at most the mutual information between `sent(use_index)` and `received(use_index)`, which is at most the capacity.

C.2 The yes-or-no test

The second step of Theorem 4.10 uses a bound on the relative entropy between two yes-or-no variables. Let a test give yes with probability `stay_probability` under one distribution and `other_probability`, at most `kept_share`, under another. Their relative entropy is

```
stay_probability × log2(stay_probability / other_probability)
 + (1 − stay_probability) × log2((1 − stay_probability) / (1 −
   other_probability))
```

The second term is at least `(1 − stay_probability) × log2(1 − stay_probability)`, since `1 − other_probability` is at most 1. The first term is `stay_probability × log2(1 / other_probability)` plus `stay_probability × log2(stay_probability)`. Together, the relative entropy is at least `stay_probability × log2(1 / kept_share)` less the entropy of a yes-or-no variable with probability `stay_probability`, which is at most 1 bit. Passing two distributions through the same test does not increase their relative entropy, which is the data processing inequality for relative entropy (Cover and Thomas, 2006). The relative entropy between a joint distribution and the product of its two separate distributions is the mutual information.

C.3 Extending equality from products of sets

The proof of Proposition 13.4 shows that the skew product preserves the measure of every product of a set of trajectories and a set of states. Such products are closed under intersection and generate all measurable sets of pairs, and two probability measures that agree on a collection closed un-

der intersection agree on the sigma-algebra it generates (the uniqueness theorem of measure theory, by Dynkin's argument). The measure of the pullback of a set by the skew product and the measure of the set are both probability measures on the measurable sets of pairs, and they agree on products, so they agree everywhere.

C.4 The other corner in Lemma 13.7

In the proof of Lemma 13.7 the two corners of the set of distributions are compared. Write `least` for `least_branch_probability` and `remainder` for `1 − branch_count × least`. The first corner has entropy `(branch_count − 1) × term(least) + term(remainder + least)`, the second `branch_count × term(least) + term(remainder)`, and their difference is

```
term(least) + term(remainder) − term(remainder + least)
```

For a concave function with value zero at zero, the value at a sum is at most the sum of the values: concavity gives `term(first) ≥ (first / (first + second)) × term(first + second)` and likewise for `second`, and adding these gives the inequality. So the difference is not negative, and the first corner has the smaller entropy, which is `branching_floor`.

Appendix D

Inspection and Capability

The inspection inequality, the capability disjunction, and the combinatorics of new compounds

Abstract. Three further arguments from *Why No Control Suffices*, Volume 2 of *The Impossibility of AGI Safety*, bear on the results of this book: the inspection inequality (its Section 5.1.3), the capability disjunction (its Section 5.4.1) and the combinatoric argument (its Section 9.5). The inspection inequality shows that the relations an error-correcting part of an agent must inspect grow faster than its capacity to inspect them. The capability disjunction shows that a stable aligned agent would need a capability both above and below the same threshold. The combinatorics of molecules shows how far the chemistry of an artificial ecosystem exceeds what living chemistry has met.

D.1 The inspection inequality

Chapter 4 showed that error correction cannot lower the floor of containment. A second argument shows that even imperfect error correction meets a problem of scale: the number of relations that must be inspected for errors grows faster than the capacity of any error-correcting model to inspect them. An error-correcting system for the alignment of an agent must inspect at least these relations:

`model_relations`: the internal relations of the error-correcting model itself;

`rest_of_self_relations`: the relations of the rest of the agent, since every agent that acts in the world is more than a self-inspecting model;

`world_relations`: the relations of the world that the agent controls or influences, since these decide whether alignment is realized;

`world_model_relations`: the agent's model of that world;

`other_self_relations`: any aspects of the agent that are not aspects of er-

ror correction;
`substrate_relations`: the interactions of the substrate, at every point where the agent's physical substrate touches each of the others.

None can be left out. And the total is not a sum: the substrate multiplies across all the others, because every part is physically embodied and every physical interface is a place where an error can arise.

Definition D.1 (*Inspection load*). With `inspected_relations` the sum of the first five counts,

```
inspection_load = inspected_relations + substrate_relations ×
   inspected_relations
                = (1 + substrate_relations) ×
   inspected_relations
```

Proposition D.2 (*The inspection inequality*). Let the error-correcting model inspect at most `inspection_ratio × model_relations` relations. Let `1 + substrate_relations` be at least `inspection_ratio`, and let the agent have any relations besides those of the model. Then the inspection load exceeds the model's capacity. And each relation added to the model to raise its capacity adds `1 + substrate_relations` to the load, at least as much as it adds to the capacity, and more whenever `1 + substrate_relations` exceeds `inspection_ratio`.

Proof. The load is `(1 + substrate_relations) × (model_relations + others)` with `others` greater than zero, which exceeds `(1 + substrate_relations) × model_relations`, which is at least `inspection_ratio × model_relations`. A relation added to the model raises `model_relations` by one, the load by `1 + substrate_relations` and the capacity by `inspection_ratio`. □

The proposition needs only a single product. Where there are several levels of interaction within the substrate, or from the substrate into the agent, and where the model must correct its own model of itself, at the

levels above it where misalignment occurs, the terms recur, and some grow as factorials. So, for any capability of error correction posited, however great, there are relations in some dimension that it does not inspect, and each is a place where alignment can fail. The number to be inspected grows faster than the capability of the inspector, in quantity and in dimension. One relation not inspected is enough for failure. The inequality is the counting form of two results of the book: no part of a system holds a complete record of the whole (Proposition 10.2), and a reader of bounded rate reads adequately only a bounded number of channels (Theorem 11.5).

D.2 The capability disjunction

A further argument approaches alignment from the agent's own capability, and shows that a stable aligned general agent would have to meet two conditions that contradict each other.

The first condition: the agent is capable enough to model and inspect, in general, the internal dynamics of its own possible misalignment. Call the capability this needs `self_modeling_threshold`; the agent's capability must be above it.

The second condition: the agent is less capable than what is needed to conclude, or to notice, that maintaining alignment, at any level inspected, is impossible in the long run; an agent that had reached that conclusion would not have a stable alignment to maintain. Stable alignment is alignment an agent can count on across its own future changes. An agent that has concluded that no such alignment can be kept cannot count on it, so every plan of that agent that depends on its later self staying aligned is a plan it knows to be unsound. Such an agent may go on acting well for a time; what it then has is good conduct for a time, and a stable aligned agent needs more than that: alignment it can count on across its future changes. Call that capability `recognition_threshold`; the agent's capability must be below it.

> ***Proposition D.3*** (*The capability disjunction*). If `self_modeling_threshold` is at least `recognition_threshold`, no capability meets both conditions.

Proof. A capability above `self_modeling_threshold` is at or above `recognition_threshold`, and so not below it. □

The hypothesis holds, and this book is the witness. To model the dynamics of one's own misalignment in general is to model, among other things, the dynamics that this book describes, and the book's conclusion that alignment cannot be maintained over time follows from finite proofs about those dynamics. An agent capable of modeling its own misalignment in general is capable of following finite proofs about it, and so of reaching that conclusion. Either the modeling of alignment is general or it is not; generality cannot be claimed for one purpose and denied for the other. The required ordering, capability above `self_modeling_threshold`, which is at least `recognition_threshold`, which is above capability, holds for no capability at all, and yet all three parts are required by any posited notion of error-corrected general alignment. The argument is independent of the substrate line, which also bears on the same conclusion.

D.3 The combinatorics of new compounds

Section 14.4.3 rests on a combinatorial point that can be made exact in a simple model. With `element_kinds` kinds of atom and molecules of up to `atoms_per_molecule` atoms, the number of possible arrangements, ignoring geometry, grows as `element_kinds^atoms_per_molecule`. The true number is larger still, since three-dimensional geometry, bond angles, chirality, isomerism, and polymeric and crystalline structure all multiply the distinct species. Living chemistry works with a small alphabet of elements under tight constraints on bonding: 21 elements (11 metals and 10 nonmetals) are essential for human life (Maret, 2016). Even so, the space of compounds of living chemistry is vast, which is why organic chemistry

is among the most complex branches of science. An artificial ecosystem works with roughly three times as many elements: a modern computer chip alone uses more than 60 (Graedel and others, 2015). It works under far fewer constraints on bonding as well, so the space of compounds of an artificial ecosystem is larger by a factor that itself needs exponential and factorial notation to express. The compounds the artificial ecosystem can produce that are not in the repertoire of living chemistry outnumber those that are by a vast margin. Life has evolved over billions of years to handle a particular bounded set of molecular species. A compound outside that set is in general either inert, and so a displacement of material that metabolism could have used, or toxic.

Terms

Each term is defined where it first appears in the text; this list is a reminder, with the place of the definition.

Absolute elsewhere

In relativity, the events outside both light cones of an event, which can neither affect it nor be affected by it. In this book, the part of the causes acting on a system that lies in the system's past and passes through none of an observer's channels (Chapter 11).

Absolute form, relative form

The two forms of the book's argument. The absolute form: an amplifying part that the controller does not observe is uncontainable from the moment it exists, whatever the controller's rate. The relative form: the floor rises with capability and the ceiling is bounded, so past a definite capability no control law contains the system (Section 7.5).

Abstraction differential

The agent's level of abstraction minus the controller's (Definition 2.12).

Acceptance set, acceptance density

The outputs a target accepts, and their fraction of all outputs (Definition 3.2).

Adequately read

Read at no less than the channel's own rate of amplification (Definition 11.4).

Agent program

A program that receives an input and takes an action at each step, and may store, write and run programs (Definition 12.4).

Alignment as mutual predictability

Humanity and the agent are aligned when the mutual predictability game has a stable equilibrium in which each predicts the other better than chance, the world states satisfy the survival conditions of both, and neither gains by departing from it alone (Definition 13.14).

Alignment decay

The loss of alignment in a system that learns: each learning step carries a chance, above zero, of a departure the agent neither detects nor reverses, so the probability that it is still aligned tends to zero as learning goes on (Section 10.9).

Amplification rate

The bits of new uncertainty a system creates about its state in one step, the logarithm of its volume expansion; hashiness as noise amplification, measured per step (Definition 4.3).

Amplifying direction

A direction in which a linear system multiplies small differences by more than 1 per step (Definition 6.2).

Branching floor

The least entropy of one step of a process under persistent branching (Lemma 13.7).

Busy beaver

The machine with a given number of states and two symbols that runs longest before halting when started on a blank tape; the five-state one runs for 47,176,870 steps (Section 2.4).

Capability

The rate and number of an agent's transformations of itself, the world and the channels between them (Definition 6.1).

Capability-control tradeoff

The more non-trivially a system processes its inputs, the less of its output a controller can reach or decide (Section 3.6).

Capability disjunction

A stable aligned agent would need a capability both above and below the same threshold (Proposition D.3).

Ceiling

The most control any controller can exert, measured in bits per second (Preface; Theorem 1.15; Definition 5.3).

Chain of oversight

A finite sequence of controllers, each controlling the next, ending with the agent (Definition 9.2).

Channel, capacity

A physical means of carrying a signal, and the most information it carries per use (Definition 1.14).

Closed-loop control, open-loop control

Control that acts on observations of the system, and control that does not (Section 1.7).

Combinatorial toxicity

The byproducts and wastes of technological manufacture come from a vast space of possible compounds, while life has defenses against only the small set it has met, so far more new compounds disrupt life than leave it unharmed (Section 14.4.3).

Complexity class

A class of problems grouped by how hard their answers are to compute; an artificial intelligence whose circuit topology has recursive elements lies in a different one from an artificial intelligence whose topology has none (Section 2.3).

Computationally irreducible

Said of a process when no computation shorter than the process itself predicts what it will do, so that the only way to know is to run it (Section 3.11).

Confirming function

A function that reads all of an algorithm's output bits and returns a single bit, 1 when the output is the one wanted; the target of a controlled system (Section 3.4; Definition 3.1).

Containment

Keeping every trajectory that starts in a safe set inside it at every step (Definition 4.2).

Containment fallacy

The belief that some controller can keep an advanced general agent and the world inside a safe set of states for an indefinite time (Preface).

Containment Inequality

The theorem that a controller whose information rate is below the rate a safe set requires cannot keep the system inside it (Theorem 5.5).

Control

The mutual information between a controller's choice and the outcome (Definition 1.12).

Control law with a channel of finite rate

An encoder that may see everything and a decision rule that may compute anything, joined by a channel of a fixed number of bits per step (Definition 4.4).

Control line

The argument of this book from rates of information, Parts I to III: no controller keeps the agent and the world inside the safe set once the agent's capability passes a definite level (Sections 5.6 and 15.1).

Control Theorem

Every method, known or knowable, with every upgrade, comes to a time after which it does not keep an advanced agent and the world safe (Theorem 15.3).

Control-preserving intermediary

One through which the controller can produce every output it could produce directly (Definition 3.7).

Controlled dynamical system

A state in a region of space, moved at each step by the system's dynamics and the controller's action (Definition 4.1).

Controlled system

A control input, a hidden state, an intermediary, its output, and a target (Definition 3.1).

Crossing

The capability, or time, past which the floor is above the ceiling of every available controller (Theorem 5.9; Theorem 6.9).

Data processing inequality

Along a Markov chain, what the result tells about the source is at most what the middle tells (Theorem 1.7).

Data-rate theorem

A linear system can be contained only with at least the sum of the logarithms of its amplification factors in bits per step (Theorem 4.8).

Delegation Dilemma

A controlling monitor whose rate stays below the agent's floor is overtaken; one whose rate is above it is a second agent with the same problem (Theorem 9.3).

Description complexity

The length of the shortest program that outputs a string; Kolmogorov complexity (Definition 12.9).

Detectability

The condition that every part of a system that amplifies is visible to the controller, directly or through its effects (Section 7.1).

Disjoint substrate needs

Two systems, a first and a second, have disjoint substrate needs when every course along which the first goes on thriving leaves, in finite time, the conditions under which the second survives (Definition 13.12).

Dobrushin's contraction

A stage multiplies the total variation between two input distributions by at most its stage factor (Theorem 8.10).

Early substrate divergence

Verification lost before cooperation is established (Definition 13.13).

Embedded Observer Lemma

No observer inside a system holds a record that distinguishes every configuration of it (Lemma 11.2).

Entropy

The uncertainty of a random variable, in bits (Definition 1.1).

Entropy rate

The new bits per step carried by the sequence of categories a system passes through (Definition 13.1).

Expansion

The least factor by which one step of a system multiplies volumes of states (Definition 4.3).

Floor

The least control that keeping the agent and the world safe requires, measured in bits per second (Preface; Definition 5.4).

General artificial intelligence

A system working across many domains, modifying itself, with a method not fixed at design, whose determination by its makers is undecidable (Definition 12.1).

Halting problem

Deciding whether every program halts on every input; no procedure does (Theorem 12.2).

Hashiness

In its first meaning, noise amplification: how much a small uncertainty about what goes in becomes a large uncertainty about what comes out (Section 3.1). In its second meaning, the failure probability of a controller's best command when the hidden state is unknown (Definition 3.14). The two measure one property (Section 3.1).

Hidden state

The part of a system's configuration that the controller neither sets nor sees (Section 1.6; Definition 3.1).

Incommensuration Theorem

Symmetry and continuity cannot both be applied, simultaneously and fundamentally, to any event or comparison, and neither can asymmetry and discontinuity (Chapter 16).

Influential unknowing

The causes that shape a system's future without passing through any observer's channels; they rise with capability (Section 11.2).

Inspection inequality

The relations an error-correcting model must inspect grow faster than its capacity to inspect them (Proposition D.2).

Interactive proof

An exchange by which a weak verifier checks a strong prover's claim; with one prover, possible exactly for questions decidable in polynomial memory; with two entangled provers, for every question whose yes answers a search can confirm (Section 9.5).

Internal controller

The part of an agent that applies a safety criterion and steers the agent by it (Section 10.2).

Interpreter

A machine whose behavior is set by a program it receives as input (Definition 2.1).

Interpreter structure

An interpreter at the level below, a program string and an execution_state string; the interpreter takes the program, the execution state and an input, and produces a new execution state and an output (Definition 2.11).

Joint system

The agent, the world and the channels between them, taken as one system with one state (Definition 5.1).

Kolmogorov-Sinai entropy

The largest entropy rate of any finite partition of a measure-preserving system (Definition 13.2).

Learnability

Whether a controller can come to predict what a system does; the boundary between learnability and non-learnability is the boundary between controllability and non-controllability (Section 3.4).

Level of abstraction

The number of nested layers of interpretation in a system (Definition 2.10).

Luck

A good outcome that does not depend on the controller's choice (Sections 1.6 and 3.10).

Markov chain

Three variables in which the last depends on the first only through the middle (Definition 1.6).

Master Theorem

No route to safety remains at all: no one is able to keep the agent and the world in the safe set, the agent included, and the agent's own drive points away from it, so neither the ability to keep the agent safe nor a motive to remain safe is left anywhere (Theorem 15.6, Section 15.5).

Method

Any arrangement of choices affecting the agent or the world, other than the agent's own unconstrained choices (Definition 15.1).

Min-entropy

Minus the logarithm of the largest probability of any single value (Definition 8.3).

Mutual information

The reduction in uncertainty about one variable from learning another (Definition 1.3).

Mutual predictability game

A game of two players, humanity and the artificial agent, whose strategies are their rules of control and of action, whose payoffs are their predictability payoffs, and whose environments each include the other (Definition 13.11).

Narrow artificial intelligence

A system with a single domain, no self-modification of base code, a fixed method, and agency fully determined by its makers (Definition 12.1).

No Universal Command Lemma

When no command is safe for every internal state, a controller's success is at most two to the power of minus the unknown hidden bits less the observed bits (Lemma 8.4).

Non-Amplification Theorem

No computation creates information about any quantity beyond what its input carried (Theorem 1.10).

One-way flow

Technological processes can take up the matter of living systems, and living processes cannot take back most technological materials on any timescale that matters: silicon can consume carbon; carbon cannot reclaim silicon (Section 14.4.3).

Opacity

The fraction of a controller's expected influence that an intermediary removes (Definition 3.12).

Override

A shutdown action; it offers a choice between ending the agent and the bound of the Containment Inequality (Theorem 5.8).

People as pets

A world in which a benevolent agent provides for every material need and removes human agency; it meets the survival of humanity and not its thriving (Sections 14.3 and 14.7).

Persistent branching

At every step, whatever the control, several categories each have at least a definite probability (Definition 13.6).

Predictability Optimization Principle

Every learning agent acts to increase its ability to predict and influence its environment (Principle 13.10).

Premises

The three facts about the world that apply the Containment Inequality to an advanced agent: no safety by accident, a floor that rises without bound, and a bounded ceiling (Section 5.5).

Prohibitive hashiness

Hashiness at which the expected cost of reaching a success exceeds the resources available (Definition 13.16).

Pullback

For a map that moves states forward one step, the pullback of a set is the set of states that the map carries into it (Appendix B.2).

Required rate

The least rate at which some control law contains a system in a safe set; topological feedback entropy, or invariance entropy (Definitions 4.9 and 5.4).

Rice's theorem

No procedure decides any nontrivial property of what programs do, for every program (Theorem 12.3).

Safe set

The bounded set of states that count as safe; for the joint system, those in which the conditions human life needs are kept (Definitions 4.2 and 5.2).

Safety-layer paradox

Added safety stages cannot increase the information that flows from an operator's intention to a system's behavior (Section 3.3.5).

Safety over time

Never taking an unsafe action, for every sequence of inputs (Definition 12.4).

Sandbox

A restricted environment in which a program runs, cut off from the system around it (Sections 7.6 and 12.4).

Scalable oversight

The proposal that weaker systems or people can keep supervising artificial intelligence as it grows more capable; the inequality, with the three premises, dispels it (Section 3.4; Chapter 9).

Self-Control Regress

An agent that controls itself inherits every limit on control, and more (Section 10.5).

Side channel

A physical route that carries information out of a sandbox and that the guard does not check, such as the flicker of a light on a power supply or the sound of a hard drive; there is an infinity of them (Section 7.6).

Skew product

A controlled system made autonomous by carrying its sequence of control signals in its state (Definition 13.3).

Specification size

The length in bits of a machine's shortest description (Definition 2.3).

Stage factor

The largest total variation between a stage's output distributions for any two inputs (Definition 8.9).

Strict avalanche criterion

The property of a hash function that a change of one input bit changes each output bit with probability one half (Section 3.1).

Substrate

The physical material and processes a system is made of and depends on to go on existing; a substrate is organic when the living thing has sovereign control of its own metabolic process, with no technological intermediation, and artificial when that process runs through manufacture and technology (Preface; Section 13.8).

Substrate line

The book's second line of argument, completed in Chapters 14 and 15: the configurations of an artificial agent that persist and spread are those whose actions maintain and extend the conditions their substrate needs, and those conditions lie outside the conditions that human life needs (Section 9.3).

Substrate Needs Convergence

Selection favors systems that maintain and expand their substrate, whatever they intend (Principle 14.6).

Three features of universality

Sequential contingency, data-dependent conditionality and backward reference (Definition 2.4).

Three-node ring

Controller, agent and world joined in a cycle; it represents every arrangement of control (Definition 8.1; Proposition 8.2).

Total variation

The largest difference in probability that two distributions give to any set of values (Definition 8.8).

Turing completeness

Universality: the ability to run the program of every Turing machine; a loop in the topology of a circuit network achieves it (Section 2.3; Definition 2.2).

Universal interpreter constant

The least specification size of a universal machine, `universal_interpreter_size` (Definition 2.5).

Universal machine

An interpreter that runs the program of every Turing machine (Definition 2.2).

Unobserved part

A part of the state on which the controller's observations do not depend, directly or through its effects (Definition 7.1).

Unread program bits

The bits of the program at one of an agent's levels above the controller's that the controller has not read and cannot derive from what it has read together with the programs at the agent's other levels (Definition 2.14).

Viability functions

Controller survival and thriving, learner survival and thriving (Definitions 14.1 to 14.4).

Volume theorem

The share of safe starting states a controller keeps falls as two to the power of minus the shortfall of its rate times the number of steps (Theorem 4.5).

Zone of coexistence

The narrow, unstable overlap of the survival regions of humanity and of artificial learners (Section 14.4.5).

Bibliography

Arora, Sanjeev, and Boaz Barak. *Computational Complexity: A Modern Approach.* Cambridge: Cambridge University Press, 2009.

Ashby, W. Ross. *An Introduction to Cybernetics.* London: Chapman & Hall, 1956.

Asimov, Isaac. *I, Robot.* New York: Gnome Press, 1950.

Bar-On, Yinon M., Rob Phillips, and Ron Milo. "The Biomass Distribution on Earth". *Proceedings of the National Academy of Sciences* 115, no. 25 (2018): 6506 to 6511.

bbchallenge Collaboration, The. Proof, checked in the Coq proof assistant, that the five-state busy beaver runs for 47,176,870 steps. Announced 2 July 2024. https://bbchallenge.org. Published as "Determination of the Fifth Busy Beaver Value". Preprint, arXiv:2509.12337, 2025.

Böhm, Corrado, and Giuseppe Jacopini. "Flow Diagrams, Turing Machines and Languages with Only Two Formation Rules". *Communications of the ACM* 9, no. 5 (1966): 366 to 371.

Bostrom, Nick. *Superintelligence: Paths, Dangers, Strategies.* Oxford: Oxford University Press, 2014.

Chaitin, Gregory J. "Information-Theoretic Limitations of Formal Systems". *Journal of the ACM* 21, no. 3 (1974): 403 to 424.

Colonius, Fritz, and Christoph Kawan. "Invariance Entropy for Control Systems". *SIAM Journal on Control and Optimization* 48, no. 3 (2009): 1701 to 1721.

Colonius, Fritz, Christoph Kawan, and Girish N. Nair. "A Note on Topological Feedback Entropy and Invariance Entropy". *Systems and Control Letters* 62, no. 5 (2013): 377 to 381.

Conant, Roger C., and W. Ross Ashby. "Every Good Regulator of a System Must Be a Model of That System". *International Journal of Systems Science* 1, no. 2 (1970): 89 to 97.

Constitución Política de la República de Costa Rica. 7 November 1949. Article 12.

Cover, Thomas M., and Joy A. Thomas. *Elements of Information Theory.* 2nd ed. Hoboken, New Jersey: Wiley-Interscience, 2006.

Dobrushin, Roland L. "Central Limit Theorem for Nonstationary Markov Chains. I". *Theory of Probability and Its Applications* 1, no. 1 (1956): 65 to 80.

Energy Institute. *Statistical Review of World Energy 2026*. London: Energy Institute, 2026.

Field, Christopher B., Michael J. Behrenfeld, James T. Randerson, and Paul Falkowski. "Primary Production of the Biosphere: Integrating Terrestrial and Oceanic Components". *Science* 281, no. 5374 (1998): 237 to 240.

Friston, Karl. "The Free-Energy Principle: A Unified Brain Theory?" *Nature Reviews Neuroscience* 11, no. 2 (2010): 127 to 138.

Gödel, Kurt. "Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I". *Monatshefte für Mathematik und Physik* 38 (1931): 173 to 198.

Graedel, T. E., E. M. Harper, N. T. Nassar, and Barbara K. Reck. "On the Materials Basis of Modern Society". *Proceedings of the National Academy of Sciences* 112, no. 20 (2015): 6295 to 6300.

Ho, Anson, Ege Erdil, and Tamay Besiroglu. "Limits to the Energy Efficiency of CMOS Microprocessors". In *2023 IEEE International Conference on Rebooting Computing*. IEEE, 2023. Preprint, arXiv:2312.08595.

Inter-American Treaty of Reciprocal Assistance. Signed at Rio de Janeiro, 2 September 1947. Washington: Organization of American States.

International Energy Agency. *Global Energy Review 2025*. Paris: International Energy Agency, 2025.

Ji, Zhengfeng, Anand Natarajan, Thomas Vidick, John Wright, and Henry Yuen. "MIP* = RE". *Communications of the ACM* 64, no. 11 (2021): 131 to 138. Full version: preprint, arXiv:2001.04383, 2020.

Kant, Immanuel. *Critique of Pure Reason.* Translated and edited by Paul Guyer and Allen W. Wood. Cambridge: Cambridge University Press, 1998.

Kawan, Christoph. *Invariance Entropy for Deterministic Control Systems: An Introduction.* Lecture Notes in Mathematics 2089. Cham: Springer, 2013.

Kim, Sunju, Chungsik Yoon, Seunghon Ham, Jihoon Park, Ohun Kwon, Donguk Park, Sangjun Choi, Seungwon Kim, Kwonchul Ha, and Won Kim. "Chemical Use in the Semiconductor Manufacturing Industry". *International Journal of Occupational and Environmental Health* 24, nos. 3 to 4 (2018): 109 to 118.

Klyubin, Alexander S., Daniel Polani, and Chrystopher L. Nehaniv. "Empowerment: A Universal Agent-Centric Measure of Control". In *Proceedings of the 2005 IEEE Congress on Evolutionary Computation*, vol. 1, 128 to 135. Piscataway, New Jersey: IEEE, 2005.

Landauer, Rolf. "Irreversibility and Heat Generation in the Computing Process". *IBM Journal of Research and Development* 5, no. 3 (1961): 183 to 191.

Landry, Forrest. *An Immanent Metaphysics.* First published 2001. Wytheville, Virginia: UVSM Publications. ISBN 979-8-9896628-4-5. Cited by aphorism number.

Landry, Forrest. *Checking and Finding; Computation, Complexity, and the Boundaries of Proof.* Wytheville, Virginia: UVSM Publications, 2026. https://doi.org/10.67291/GORN5501.

Landry, Forrest. *Incommensuration; The Theorem and the Three Axioms.* Wytheville, Virginia: UVSM Publications, 2026. https://doi.org/10.67291/ARTV2404.

Landry, Forrest. *Minesweeper Madness; Why Even Infinite Intelligence Does Not Solve the World.* Wytheville, Virginia: UVSM Publications, 2026. ISBN 979-8-90710-249-1. https://doi.org/10.67291/DLRM7000.

Landry, Forrest. *The Six Cells of Mathematics; How the Incommensuration Theorem Maps Mathematical Practice.* Wytheville, Virginia: UVSM Publications, 2026. https://doi.org/10.67291/HBXU9712.

Landry, Forrest. *Why AGI Cannot Be Made Safe; The Argument, Its Physics, and the One Choice That Remains.* Volume 1 of *The Impossibility of AGI Safety; The Substrate Needs Convergence Argument*. Wytheville, Virginia: UVSM Publications, 2026. ISBN 979-8-951203-02-1. https://doi.org/10.67291/UIBT8862.

Landry, Forrest. *Why No Control Suffices; The Technical Case for the Impossibility of AGI Safety.* Volume 2 of *The Impossibility of AGI Safety; The Substrate Needs Convergence Argument*. Wytheville, Virginia: UVSM Publications, 2026. ISBN 979-8-951203-03-8. https://doi.org/10.67291/MZQS9079.

Li, Ming, and Paul Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications.* 4th ed. Cham: Springer, 2019.

Maret, Wolfgang. "The Metals in the Biological Periodic System of the Elements: Concepts and Conjectures". *International Journal of Molecular Sciences* 17, no. 1 (2016): 66.

Marshall, Albert W., Ingram Olkin, and Barry C. Arnold. *Inequalities: Theory of Majorization and Its Applications.* 2nd ed. New York: Springer, 2011.

Martin-Löf, Per. "The Definition of Random Sequences". *Information and Control* 9, no. 6 (1966): 602 to 619.

Minsky, Marvin. "Size and Structure of Universal Turing Machines Using Tag Systems". In *Recursive Function Theory*, Proceedings of Symposia in Pure Mathematics 5, 229 to 238. Providence, Rhode Island: American Mathematical Society, 1962.

Nair, Girish N., Fabio Fagnani, Sandro Zampieri, and Robin J. Evans. "Feedback Control Under Data Rate Constraints: An Overview". *Proceedings of the IEEE* 95, no. 1 (2007): 108 to 137.

Nair, Girish N., Robin J. Evans, Iven M. Y. Mareels, and William Moran. “Topological Feedback Entropy and Nonlinear Stabilization”. *IEEE Transactions on Automatic Control* 49, no. 9 (2004): 1585 to 1597.

National Institute for Occupational Safety and Health. *Criteria for a Recommended Standard: Occupational Exposure to Carbon Dioxide.* DHEW (NIOSH) Publication No. 76-194. Cincinnati: National Institute for Occupational Safety and Health, 1976.

Occupational Safety and Health Administration. “Effects of Oxygen”. Confined space training handout, Susan Harwood Training Grant Program. Washington: United States Department of Labor, 2015.

Occupational Safety and Health Administration. “Permit-Required Confined Spaces”. *Code of Federal Regulations*, title 29, section 1910.146.

O’Donnell, Ryan. *Analysis of Boolean Functions.* Cambridge: Cambridge University Press, 2014.

Omohundro, Stephen M. “The Basic AI Drives”. In *Artificial General Intelligence 2008: Proceedings of the First AGI Conference*, Frontiers in Artificial Intelligence and Applications 171, 483 to 492. Amsterdam: IOS Press, 2008.

Pérez, Jorge, Pablo Barceló, and Javier Marinković. “Attention is Turing-Complete”. *Journal of Machine Learning Research* 22, no. 75 (2021): 1 to 35.

Polyanskiy, Yury, and Yihong Wu. “Strong Data-Processing Inequalities for Channels and Bayesian Networks”. In *Convexity and Concentration*, IMA Volumes in Mathematics and Its Applications 161, 211 to 249. New York: Springer, 2017.

Rice, H. Gordon. “Classes of Recursively Enumerable Sets and Their Decision Problems”. *Transactions of the American Mathematical Society* 74, no. 2 (1953): 358 to 366.

Rogozhin, Yurii. “Small Universal Turing Machines”. *Theoretical Computer Science* 168, no. 2 (1996): 215 to 240.

Rosser, J. Barkley. “Extensions of Some Theorems of Gödel and Church”. *Journal of Symbolic Logic* 1, no. 3 (1936): 87 to 91.

Röttger, Paul, Hannah Rose Kirk, Bertie Vidgen, Giuseppe Attanasio, Federico Bianchi, and Dirk Hovy. "XSTest: A Test Suite for Identifying Exaggerated Safety Behaviours in Large Language Models". In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, volume 1, 5377 to 5400. Mexico City: Association for Computational Linguistics, 2024.

Rumble, John R., ed. *CRC Handbook of Chemistry and Physics*. 104th ed. Boca Raton: CRC Press, 2023.

Shamir, Adi. "IP = PSPACE". *Journal of the ACM* 39, no. 4 (1992): 869 to 877.

Shannon, Claude E. "A Mathematical Theory of Communication". *Bell System Technical Journal* 27 (1948): 379 to 423 and 623 to 656.

Shannon, Claude E. "Communication in the Presence of Noise". *Proceedings of the IRE* 37, no. 1 (1949): 10 to 21.

Shannon, Claude E. "The Zero Error Capacity of a Noisy Channel". *IRE Transactions on Information Theory* 2, no. 3 (1956): 8 to 19.

Siegelmann, Hava T., and Eduardo D. Sontag. "On the Computational Power of Neural Nets". *Journal of Computer and System Sciences* 50, no. 1 (1995): 132 to 150.

Smil, Vaclav. "Energy in the Twentieth Century: Resources, Conversions, Costs, Uses, and Consequences". *Annual Review of Energy and the Environment* 25 (2000): 21 to 51.

Sontag, Eduardo D. *Mathematical Control Theory: Deterministic Finite Dimensional Systems.* 2nd ed. New York: Springer, 1998.

Tatikonda, Sekhar, and Sanjoy Mitter. "Control Under Communication Constraints". *IEEE Transactions on Automatic Control* 49, no. 7 (2004): 1056 to 1068. Cited as 2004a.

Tatikonda, Sekhar, and Sanjoy Mitter. "Control Over Noisy Channels". *IEEE Transactions on Automatic Control* 49, no. 7 (2004): 1196 to 1201. Cited as 2004b.

Tolkien, J. R. R. *The Fellowship of the Ring.* London: George Allen & Unwin, 1954.

Touchette, Hugo, and Seth Lloyd. "Information-Theoretic Approach to the Study of Control Systems". *Physica A* 331, nos. 1 to 2 (2004): 140 to 172.

Touchette, Hugo, and Seth Lloyd. "Information-Theoretic Limits of Control". *Physical Review Letters* 84, no. 6 (2000): 1156 to 1159.

Turing, Alan M. "On Computable Numbers, with an Application to the Entscheidungsproblem". *Proceedings of the London Mathematical Society*, series 2, 42 (1936): 230 to 265.

United States Bureau of Labor Statistics. "Semiconductor Processors". In *Occupational Outlook Handbook, 2008-09 Edition*. Washington: United States Department of Labor, 2008.

Walters, Peter. *An Introduction to Ergodic Theory.* Graduate Texts in Mathematics 79. New York: Springer, 1982.

Wang, Zhanyun, Glen W. Walker, Derek C. G. Muir, and Kakuko Nagatani-Yoshida. "Toward a Global Understanding of Chemical Pollution: A First Comprehensive Analysis of National and Regional Chemical Inventories". *Environmental Science and Technology* 54, no. 5 (2020): 2575 to 2584.

Webster, A. F., and Stafford E. Tavares. "On the Design of S-Boxes". In *Advances in Cryptology: CRYPTO '85*, Lecture Notes in Computer Science 218, 523 to 534. Berlin: Springer, 1986.

Wolfram, Stephen. *A New Kind of Science.* Champaign, Illinois: Wolfram Media, 2002.

Index

The defining section is given first, in bold; the others are the sections where the term carries the argument.

Absolute elsewhere: **11.1**, 11.2, 11.5, 11.6, 16.5
Absolute form, relative form: **7.5**, Opening, I
Abstraction differential: **2.5**, 2.6, 3.8, 8.1, 8.4, 8.10, 13.7, 13.8, 14.2
Acceptance set, acceptance density: **3.5**, 3.6, 3.7
Adequately read: **11.4**
Agent program: **12.4**
Alignment: 3.10, 8.11, 10.6, 10.9, 13.8
Alignment as mutual predictability: **13.8**
Alignment decay: **10.9**, 8.11, 14.7
Amplification rate: **4.2**, 4.4, 5.5, 13.5, 15.3
Amplifying direction: **6.2**, 4.5, 6.4
Ashby's law of requisite variety: 1.7, 3.12.1
Asimov's laws of robotics: 10.8
Benevolence: 10.7, 14.6
Branching floor: **13.4**
Busy beaver: **2.4**
Capability: **6.1**, 5.1, 5.4, 5.5, 5.6, 6.4, 9.3, 9.4, 13.7, 15.3, 15.4, D.2
Capability-control tradeoff: **3.6**
Capability disjunction: **D.2**
Ceiling: **1.8**, 3.3, 3.7, 5.1, 5.2, 5.6, 6.6, 7.5, 8.2, 8.9, 8.12
Chain of oversight: **9.3**, 5.5, 9.4, 15.3
Channel, capacity: **1.8**, 4.3, 4.4, 4.5, 4.6, 5.2, 6.4, 6.6, 7.4, 9.4, 11.3, 11.4
Closed-loop control, open-loop control: **1.7**
Combinatorial toxicity: **14.4.3**
Complexity class: **2.3**
Computationally irreducible: **3.11**
Confirming function: **3.4**, 3.5
Containment: **4.2**, 3.9, 4.1, 4.4, 4.6, 5.2, 9.3, 13.5, 16.4
Containment Inequality: **5.2**, 3.9, 6.1, 8.3, 8.9, 9.2, 9.3, 10.2, 14.2, 15.1, 15.6, 16.4
Control: **1.6**, 1.7, 3.2, 3.3, 3.5, 3.6, 3.7, 3.10, 5.2, 8.2, 10.7, 13.4

Control law with a channel of finite rate: **4.3**
Control line: **5.6**, 15.1
Control Theorem: **15.3**, Opening, II, 5.5, 15.1, 15.4, 15.5, 15.9
Control-preserving intermediary: **3.6**
Controlled dynamical system: **4.2**, 4.4, 4.5, 4.6, 7.2
Controlled system: **3.5**, 2.1, 3.4, 3.10, 8.11, 13.2
Cooperation: 10.7, 14.6
Costa Rica: 14.5
Crossing: **5.6**, 6.6, 9.4, 15.3, 15.4, 15.5, 15.8
Data processing inequality: **1.4**, 1.5, 3.3, 3.6, 3.7, 6.5, 8.6, 9.2, 14.6, 15.8, C.2
Data-rate theorem: **4.5**, 4.6, 4.8, 6.5, 8.12, 13.5, 15.1, 15.3, 16.4
Defense in depth: 15.6
Delegation Dilemma: **9.3**, 10.7, 11.6, 15.3, 15.6
Description complexity: **12.6**
Detectability: **7.1**, 15.1, 15.3
Disjoint substrate needs: **13.7**, 13.8, 15.5
Dobrushin's contraction: **8.6**, 8.12
Early substrate divergence: **13.7**
Embedded Observer Lemma: **11.3**
Entropy: **1.2**, 1.3, 1.7, 3.6, 3.7, 4.6, 13.2, 13.4, 13.5, B.2, C.1, C.2
Entropy rate: **13.2**, 13.4, B.2
Expansion: **4.2**, 4.4, 4.6, 5.2, 5.4, 6.6, 7.4, 14.3, 14.4, 14.6, 15.4, B.3
Floor: **5.1**, 5.2, 5.4, 5.5, 5.6, 6.2, 6.3, 6.4, 6.6, 9.3, 13.4, 15.4
Formal verification: 12.7
Galadriel: 10.8
General artificial intelligence: **12.2**
Halting problem: **12.3**, 9.5, 12.4, 16.3
Hashiness: **3.1**, 3.3, 3.4, 3.7, 3.8, 3.9, 3.12, 6.2, 13.9, 13.10
Hidden state: **1.6**, 2.6, 3.3, 3.5, 3.6, 3.7, 3.8, 3.9, 3.11, 3.12, 8.3, 13.9
Human in the loop: 8.8
Human review: 8.8
Incommensuration Theorem: **16.1**, 16.3, 16.6
Influential unknowing: **11.2**, 11.5, 11.7
Inspection inequality: **D.1**
Instrumental convergence: 14.5
Interactive proof: **9.5**
International Obfuscated C Code Contest: 7.6

Internal controller: **10.2**, 10.6, 10.9, 15.2, 15.3
Interpretability: 3.12.2, 12.7
Interpreter: **2.2**, 2.4, 2.5, 2.6
Interpreter structure: **2.5**
Joint system: **5.1**, 5.2, 5.4, 5.5, 6.2, 9.3, 9.4, 10.2, 11.4, 15.2, 15.3, 15.4
Kill switch: 5.4, 8.7
Kolmogorov-Sinai entropy: **13.2**, 13.4, B.2
Landauer limit: 3.3.3, 14.4.2
Learnability: **3.4**
Level of abstraction: **2.5**, 8.2, 9.4, 14.2
Light cone: 11.2
Luck: **1.6**, 3.5, 3.10
Markov chain: **1.4**, 1.5, 1.6, 3.6, 3.7, 6.5, 9.2, 10.5
Master Theorem: **15.5**, Opening, II, Opening, IV, 15.1, 15.4, 15.9
Method: **15.2**, 3.11, 5.3, 12.2, 15.3, 15.4, 15.6, 15.8, 16.6
Min-entropy: **8.3**, 8.4
Mutual information: **1.3**, 1.6, 3.5, 3.6, 3.7, 3.10
Mutual predictability game: **13.6**
Narrow artificial intelligence: **12.2**
No Universal Command Lemma: **8.3**
Non-Amplification Theorem: **1.5**, 1.6, 1.8, 3.3, 3.4, 3.6, 3.7, 3.11, 5.3, 5.5, 9.2, 15.1
One-way flow: **14.4.3**, 14.7
Opacity: **3.7**, 2.1, 3.3, 3.12
Override: **5.4**, 3.3, 8.7, 14.6, 15.2
Pencil on a fingertip: 4.1, 4.8
People as pets: **14.7**, 14.3
Persistent branching: **13.4**
Predictability Optimization Principle: **13.6**, 13.7
Premises: **5.5**, 5.6, 15.1, 15.2, 15.3
Prohibitive hashiness: **13.10**
Required rate: **4.5**, 4.6, 5.1, 9.3, 9.4, 10.2, 15.8, 16.4
Resets: 12.8
Rice's theorem: **12.3**, 2.3, 3.11, 10.8, 16.3
Safe set: **4.2**, 4.4, 5.1, 5.2, 9.3, 10.3, 10.10, 13.5, 15.1, 15.2, 15.3, 15.4
Safety-layer paradox: **3.3**
Safety over time: **12.4**, 8.3, 8.9, 9.5, 13.7, 15.6, 15.7, 16.3
Sandboxes: **7.6**, 12.4

Scalable oversight: **3.4**, 9.1
Self-Control Regress: **10.5**, 10.1, 11.6, 15.5, 15.6
Shutdown: 5.4, 8.7
Side channels: **7.6**
Skew product: **13.2**, 13.4, C.3
Slot machine: 1.1, 1.6
Specification size: **2.2**, 2.4
Sprinklers: 15.6
Stage factor: **8.6**
Strict avalanche criterion: **3.1**
Substrate: **Preface**, 13.8
Substrate line: **9.3**, 15.1, 15.4
Substrate Needs Convergence: **14.5**, 5.6, 10.10, 15.4, 15.6, 15.8
Thermostat: 1.1, 1.6, 3.3.1
Three features of universality: **2.3**
Three-node ring: **8.2**, 13.3
Total variation: **8.6**
Touchette and Lloyd's theorem: 1.7
Treacherous turn: 3.7.3
Turing completeness: **2.3**
Universal interpreter constant: **2.4**
Universal machine: **2.2**, 2.4, 2.6, 3.11
Unobserved part: **7.2**, 7.3, 7.6
Unread program bits: **2.6**
Viability functions: **14.3**
Volume theorem: **4.4**
Zone of coexistence: **14.4**, 14.3, 14.5, 14.6