The Elements of Intelligence [1] [2]

Yingkui Lin

April 2019

[1]This is the part one of a two-part project, as an update to the article *Naturally* written in 2014, and the other part is *A Guide to Life*

[2]You can find more resources for this book at https://github.com/lin/eofi, and https://github.com/lin/guide for *A Guide to Life*

Contents

Preface

To solve all problems, there is only one problem to solve, which is the problem of problem solving. So, why not solve it?

This book is a summary of my personal expedition to attack this problem. It tries to generalize and contain nearly everything I know about learning, solving and thinking. The simple assumption is that I believe we can simulate how human brain thinks and make it solve problems better than a gifted person. By saying all problems, what I really mean is that we can solve all problems with certain level of complexity that's feasible to solve by a group of gifted human, like how to cure cancer and send human to Mars.

The programming language of this book is Javascript (ES6).

The purpose of this book is to: 1. Layout the right structure. 2. Raise the right questions. 3. Rather solving it

The analyse is two-folded, one for computation and mathematics, and another is human intelligence.

We can't solve all the problems, the answer to some problems might hide in a tiny corner of the computational universe. But as we see, human brains are capable to lead us to many exiciting places, if we could mimic human brains brilliancy, that will be enough promising.

To solve the problem, we start with an initial scan, and by assuming the answer is hidden within touch, a purposeless exploration starts. By increasing the searching area, we reach more and more, and explore deep and deep. In such solving techniques, at least in my own opinion we can mimic a human solver with intelligence score around 140 (top .25%). Maybe we still can't simulate how magicians like Newton and Gauss solve, but if we could have millions of MIT graduates as our solving slaves, how can we be more greedy at this time?

Around 350 years ago, physics started and reached peak at early 20th century. Computer science started in 1936,

---

The structure of this book:

1. The normal computation.

Beginning with distinguishability.

The last section will be loads of definitions.

2. The abstraction.

Beginning with tightness of distinguishability.

All sorts of pattern recognization feature extraction ability feature relationship all kinds of this related concepts

And end with a subsection of Layers of abstraction.

$$a \to \{a\} \to \{\{a\}\} \to \{\{\{a\}\}\}$$

Then, The major part of this chapter should end with equivalence of computation.

This is a wonderful chapter. The computational Facts

Now it's a late age of information and beginning age of intelligence, artificial intelligence will conquer the world where information can merely solve a small set of easier problems. In a foreseeable situation, all major disease will be cured. People can experience a simulation of any imaginable rules of the world, so real that you can't tell the difference from reality. You can sleep for 500 years, and resurrection at another planet. The possibility of life is unlimited.

The highlight of this whole book might be the emphasize on the importance of distinguishability.

The first chapter should be very short. Because it simply revisits the computation and computer system knowledges. Here we define distinguishability as a short introduction

The second should discuss distinguishability in details. And focusing on all kinds of abstract level computations.

The third chapter should discuss problems in details, how to define a problem and how to categorize problems in types. The hardest problem in this chapter in that

This book tries to generalize everything I know about learning and problem solving.

This is a book full of idiotic discuss, but through those simple to human yet complex to computer examples.

The grammar and vocabulary is a disaster, and I have to do this way so I don't waste too much computation power to less importance formating to the content.

Subjects lead to this result.

1. Computer Systems
2. Theory of Computation
3. Computational Complexity Theory
4. (Shannon) Information Theory
5. Algorithmic Information Theory
6. Pattern Recognization and Machine Learning
7. Deep Learning

If one person doesn't know about computer science, he doesn't understand the world very well, even true for giants like Eintein and Hilbert.

My understanding on computer science and mathematics.

This is a bold book. There are amazing books in various topics, both deep and brilliant. But I always want to read a book which have following features:

1. accessable.
2. talking about nearly everything.
3. thinking in the highest level of abstraction.

What I am not sure yet are:

1. the importance of graph and linking
2. the definition of problem
3. why confident induction is working (solved!)

1. understanding how human thinks
2. how to build a general artificial intelligence
3. how things are related to real life situations

Full of bad examples, cause searching the right examples requires too much time.

Chapter 1

Computation

The only thing that's reliable is the definition. The only linkage or the connection is the reuse of the definition.

There are only two types of information: data and function. while we can consider function as a piece of data. and the third one is introduced as abstraction

1.1 Distinguishability

1.1.1 100 Examples of Distinguishability

This is a personal $\bigcirc \rightarrow$ project to give as much as real life example possible. Since I can experience reunderstand the world again and again. So I have to write them down as a way to show the importancy of this fundamental concepts.

To realize the importancy of distinguishability is the biggest revelation in my life.

1. Human brain should have several similiar areas of distinguish data.
2. Those areas are the same.
3. To distinguish is the most important functionality of the brain.
4. distinguish with a degree works great both for discrete objects and for the class level objects.

Your brains are capable to distinguish for tolerance level of clarity.

When it is easy to distinguish then no detail distinction is needed. If things are get strict, deeper inspect should be considered.

When we extrapolate, what we're based on is that we can't distinguish things for all object in that class under those circumstances.

Everything we do everyday is basic four actions: distinguish things, recognize things, transform things and associate things.

Distinguishability permeates every aspects of intellectual pursuit.

Cue is what you can distinguish from environments but also indistinguishable from time to time.

I believe what makes us so superior is because we could quickly find out something distinguishable in a degree.

Most of the time we only care it's distinguished or not.

Information means distinguished objects after distinction. But distinguishability means the level beneath information.

The foundation of information lies on distinguishability. How contrast and how unique an object is.

Real-Life Examples

Let's say you would like to use command line 'git clone https://github.com/lin/eofi.git', while this is exactly distinguishability is. The whole idea of computer science is based on distinguishability.

Any identifier is used to distinguish from each other. Your username in the internet is to distinguish you from others. The number system is to distinguish things, like you ID number. We use number to distinguish things. Address numbers. IP address. The binary system can be used to distinguish things. (00, 01, 10, 11)

This is mapping distinguishable things to another distinguishable things. This is also the case of replaceable definition.

The following is for ambiguity, not for distinguishability.

Another great example is the capital letters. GoT or got,

- Are you real?
- Well, if you can't tell, does it matter?

Indistinguishability

Simulation of the World

If you play a lot of video games, sometimes you can't distinguish it from reality.

While you watch a visual effect movie, you sometimes can't distinguish it from reality or magic.

Relativity Theories

The principle of relativity states that you can't distinguish systems of reference by testing laws of physics.

For the general relativity, it starts with a thought experiment by Einstein, that you can't distinguish from an elevator falls freely and a space elevator accelarates at the same rate as gravitational acceleration.

Turing Test

We couldn't distinguish from real people and artificial intelligence.

In some narrow areas, like shown by using Google Duplex to make a restaurant reservation.

Archimedes Eureka

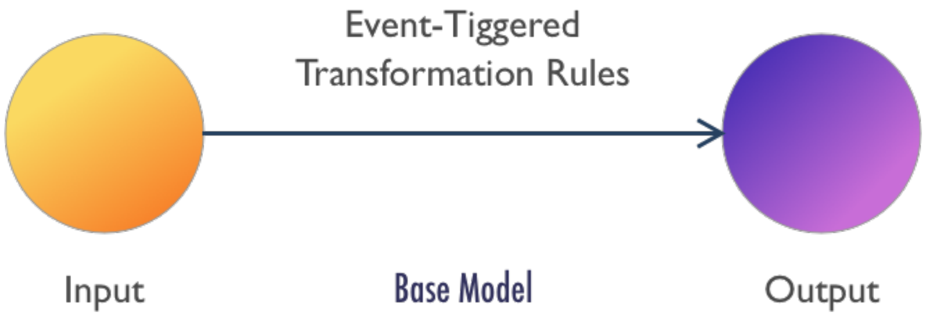


Figure 1.1: Computation Models

1.1.2 Reference and Dereference

1.2 Composition

1.3 Transformation

One thing I have to remember is that you have to make the instruction work for all instant. This is the foundation of this book. You have to make abstraction before you make transformation.

The transformation you define is for all not for instance. Even in turing machine, you move left twice, you don't care about the one you missed, because that is not what you care, no matter what symbol is on that square, it's the same instruction. This is the abstraction of operation, on operation to deal with all situations.

What intuitive really means is the automatic autocorrection based on the event trasnformational rules in brains, that processed in the subconscious. And the stregth of the neurons is how intuitive it will be.

1.3.1 Events

The first three are from an article by Chaitin[1].

The rules mean under what conditions do what.

By saying conditions, first you have to know how to recognize conditions and what is recognizable conditions, of course that's when you definable, which in turn means distinguishable.

By saying doing, that means you have to know what is allow to transform.

If we have clearly defined the **computing** procedure, we can easily compute. That means the input is distinguishable from others and the transformation rule is clear and distinguishable. For example:

```
def add(x)
  return x + 1
```

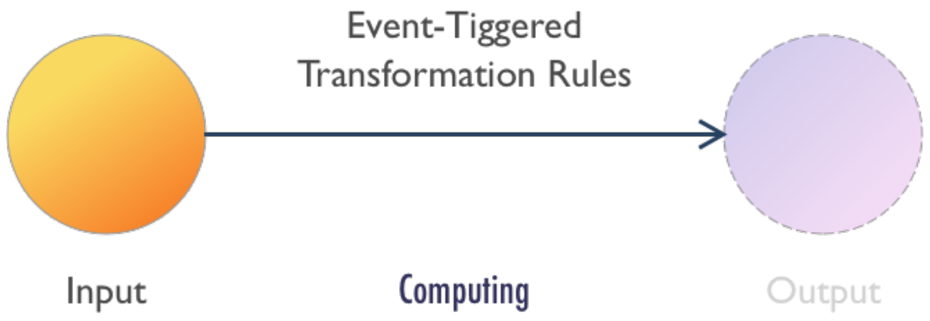


Figure 1.2: Schematic Definition of Computation

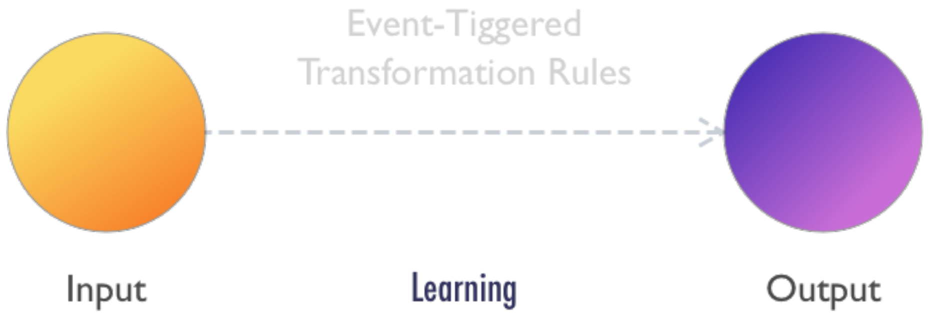


Figure 1.3: Schematic Definition of Learning

And your goal is to compute:

```
add(3)
```

Whereas **learning** is that you know the input and output, but you don't know the transformation rules.

For example:

```
add(3) = 4
add(4) = 5
add(5) = 6
add(6) = 7
```

And your goal is to compute:

```
add(10)
```

Searching is to find an instance which satisfies certain criteria in terms of computing algorithm.

For example:

```
def add(x)
  return x + 1
add(m) = 5
```

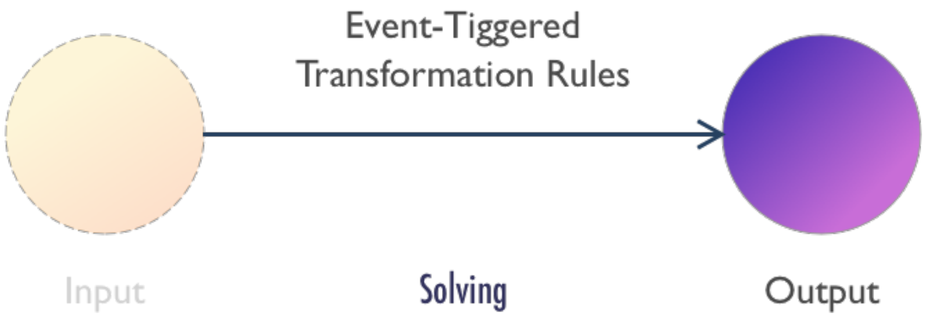


Figure 1.4: Schematic Definition of Solving

And your goal is to compute:

```
m
```

1.3.2 Inputs and Outputs

1.4 Computation

1.4.1 Read, Write and Store Information

1.4.2 Conditional Checking and Jump

Here is quite important. If you are using the conditional check, you require more about the data, more specific part of the information, more abilities is on the requirements. So, in this case, you are define a new class of things, you implicitly that this transformation will do different(distinguishability) for different class, so be aware. But if there is no conditional check, that means you don't care about the data content, only its distinguishability. So any example will do the same.

The transformation doesn't care, doesn't bother.

So the transformation is care is the distinguishability in the context(code block).

When you follow the steps of the instruction, you can sense it (of course this can be don mechanically) some things remain and some things don't change the picture.

Examples

1.4.3 Boundary of Computation

Turing Completeness

What I guess is that the reason of the robustness of Turing machine is that it can simulation any well-defined thought, the distinguishability at its highest level. No ambiguity is allowed. Everything you can distinguish clearly can be represented in the system. Every rule you can say precisely can be emulated in the machine. Every transformation every possible computation is within the power of a Turing machine.

Church-Turing Thesis

The Turing machine are capable of doing following abilities:

1. store and retrive data. 2. transform data to another one. 3. able to compute in a sequencial order. 4. able to jump to a specific instruction position.

1.5 Computation with Incomplete Information

1.5.1 Overview

The name system is using binary numbering, where the dashed objects are 1s and the solid ones are 0s. For example, Type 3 is 011, which in binary is 3.

As we can assume that Turing machine is the foundation of all. Then, we have to be able to define things which is distinguishable. So here are all the distinguishable parts

The Type 0 is a complete fact that we have all the input, the instruction and the output. The Type 1 is simply the normal computation. Nothing fancy about Type 1. All the programs we all running is Type 1.

Type 2 is when we don't know what exactly is the transformation rules. In this case, we are dealing with physics and science, where we know input and output and guess what't the governing rules underneath. We can't 100% sure about the result, all we can know is that for some sense we can't distinguish the rules underneath and the rule we present at a certain level. While we mean learning, this is exactly the case.

Type 4 is when we knows the rule and output and guess the input. Here we are doing engineering and solving a searching problem. Also, in this type, we are using it as a sieve or a checker, to define a class, which means we are use type 4 to abstract. Type 4 is as important as type 1, as it's the driven force of reduce the computational cost as we will discuss in chapter 4.

Type 5 is basically the function we write in daily routine programming. Where you using some symbol to denote some operands in the transformation. As we will discuss in chapter 2, the equivalence of computation of Type 5 is one of the core functionality in abstract thinking.

Type 3 is when you don't know the function as well as the output, you only know the input, while in this case, it's like you type $f(3)$, nothing intereting happens here as far as I can see. But if you insist say at least we know the input, then in this case the thing is that you could compute with this information. This will be helpful when we discuss it as solving problems as purposeless transformation.

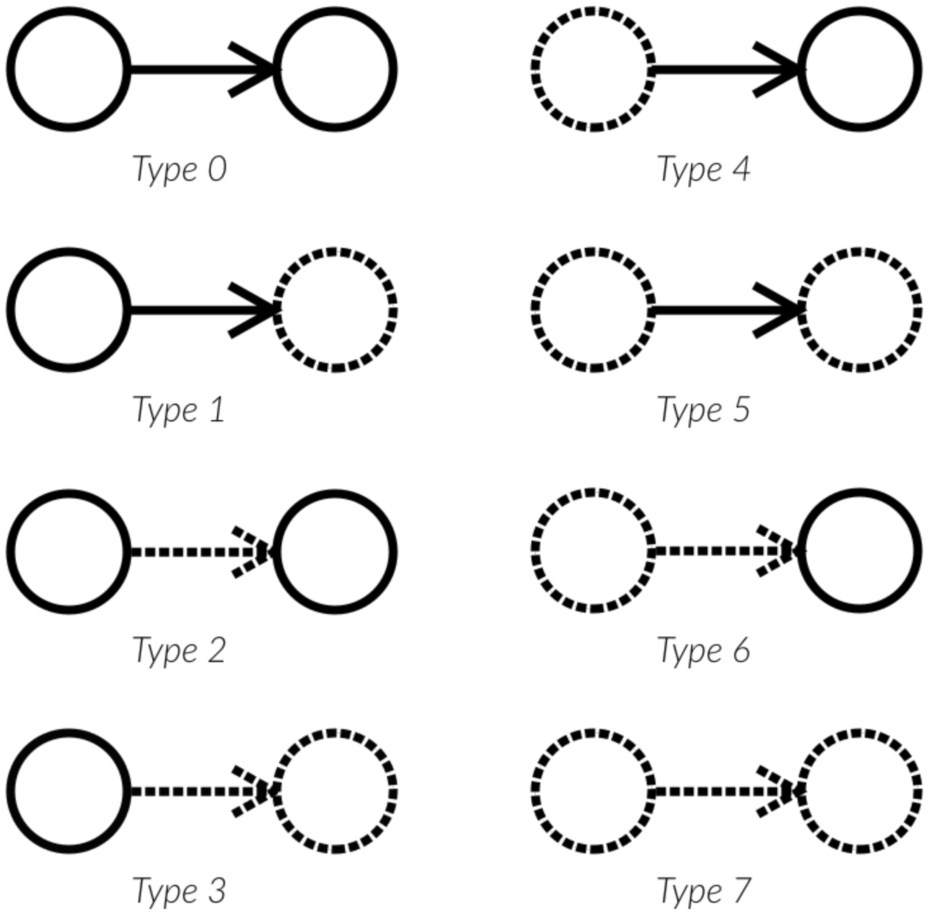


Figure 1.5: All Possible Types of Incomplete Computation

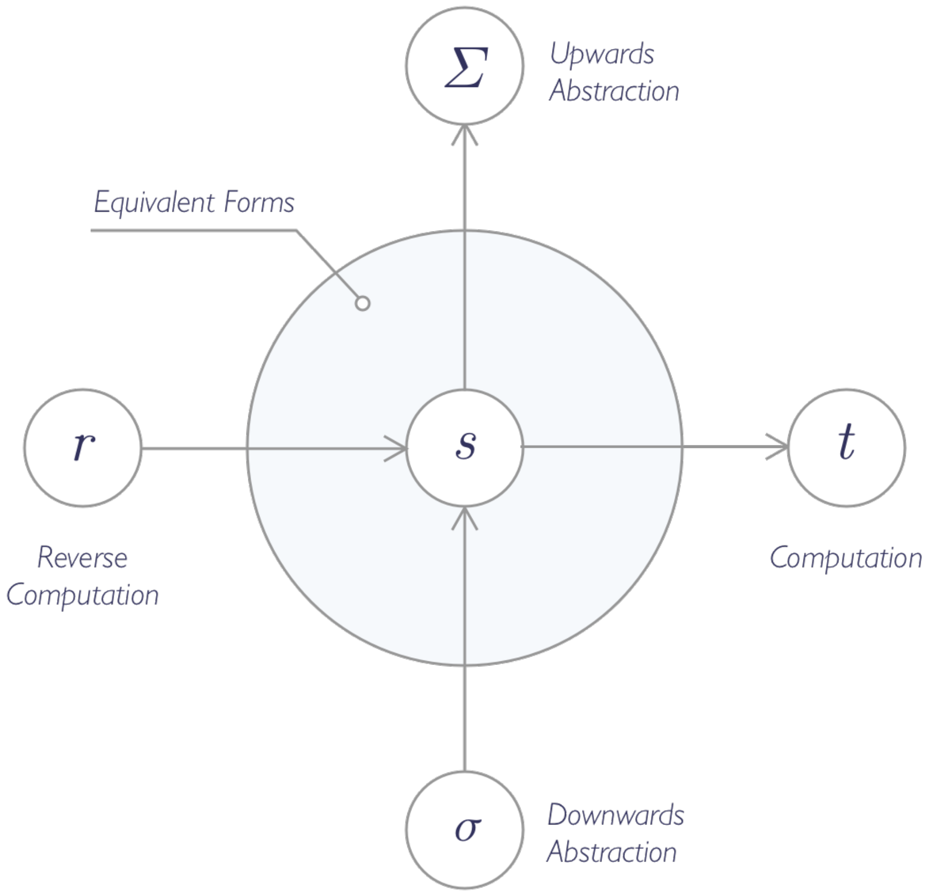


Figure 1.6: All Possible Moves from a Valid Statement

Type 6 is also boring as type 3, where it's like asking $f(x) = 3$, where you don't know any thing about the function and the input. But the same as Type 5, it will be helpful when we discuss about the inverse implications.

Type 7 is $f(x)$, this would be the highest abstraction we can get, and all the above abstraction can be transform back to this form.

For each type, you can always find a equivalence form of it and also you can combine (or link) them to perform a more complicated thing. And combine and abstract is the common themes.

1.5.2 Classification and Abstraction

1.5.3 Induction and Learning

1.5.4 Reverse Implications

1.5.5 Symbolic Functions

1.5.6 Computational Facts

1.5.7 Combination of Computational Facts

Chapter 2

Abstraction

Abstraction is an old business, from the syllogism of Aristotle to the recent machine learning. We are using it every day without notice, it reduces our reductant works, it makes education a viable action, it helps understand. At the same time, abstraction is hard to understand, to this day, I can't really appreciate the true meaning of abstraction, and the fundamental information it tries to convey.

What is think abstractly, that means you think in terms of class, in terms of reduce the tightness being too specific and group things with same criteria and make actions and computations in class and apply the result in one time but works for all terms.

One example here is $(x + y)^8$, what is the coefficient of x^2y^6? You may say this is a easy task, but you think it fast because you know all the other terms are same for some measure, that they are differ with different x^k in each instance.

Let's make it even more dramastic:

Example 2.0.1. $x^2 + x^10 = a_0 + a_1(x + 1) + \cdots + a_9(x + 1)^9 + a_10(x + 1)^10$, find a_9

In this case, if you have to think too specific, you would end up writing long sequence of the terms, while all you need to figure out is that a_9

2.1 Atomized Abstracion

Here we are treating abstraction as information, as pieces of information. atomized pieces. builded up as abilities and cares. features and

100 Examples of Abstraction

As I discuss in the distinguishability, this concept is so important and so ubiquitous, I have to do this anonying action again in this book.

Invariance under Variation

Evolution Perspective

I always find evolution has its best choice. The way brain works is actually quite flexible when we talk about its ability to solve various jobs and efficient when consider the complex situation human has to deal with.

1. we have hunter and gather.
2. we must act fast.
3. we have to adapt to various enviroments.
4. we don't need to be too smart, only smart enough to pass the genes on.
5. we want to save brain energy in order to live on limited energy available.

Following is my personal guess on evolution and brain functionality:

Let's imagine a situation that we need to survive in wild, we need to hunt and gather food.

The evolutional advantages of brain is to process the information, and so we could separate harm from goods. More and more we gain the ability of not only distinguish one thing to another. But also able to predict the future, which can learn the rules of transformation.

While this is still not that helpful, since we need to be able adapted to a large range of situations. If we understand things too specific, we can't predict the new situation that varies a little bit.

And also in the very beginning of eyes, we could only see a very low resolution of the world, that's an advantage somehow. Since we could only care a low level of certainty to be survive, and when we don't have high level functionality to process and predict. Vague means robost when you have only one technique at hand.

Gradually, brain evoles to a new level that high resolution is helpful and precise descriptions means a precise prediction. But at the same time the old part of the brain remains as a part of function to provide levels of resolutions. That means when we have an image input to our brain, it could be able to store multiple copies of the image, varies in terms of resolution. If we talk in the object oriented programming way, that means we have lots of inherence there.

What I mean here as resolution is not that in graphics, but the resolution of information, to what degree we care about the distinction of two similiar objects.

In high resolution, the details are described precisely. While in low resolution, we consider human and rock both as physical object. In terms of what concerns us, we make different views of things.

Let's take a picture for example:

This is an image with a bowl and some corns. While I see this, the image is not only the rice and corns, it's also a thing with yellow and white.

Resolution means you can distinguish stuff in level of needs, at different context, you need different contrast to solve the problems at hand.

Let us review this part:

First, evolution gives us the ability to distinguish data.

Then, we need to distinguish data with vague, which essentially class level information or abstract information.

Third, not only we predict, but also we need to learn the rule of inference, under what condition, what input will transform to what ouput.

2.1.1 Ability and Feature

2.1.2 Relationships

Examples

Here lists some examples of mathematical classifications:

Example 2.1.1. Give an instance of prime numbers

Example 2.1.2. Is 3 a prime number?

Example 2.1.3. Is $f(x) = x^3$ an odd function?

Example 2.1.4. What is the fourth elements of $a_n = n^2$

Odd number is the best illustration of abstraction.

2.2 Hierarchy of Abstraction

2.2.1 Resolution of Distinguishability

When you say something about a data. $x^2+y^2 = 1$, what you are getting is that you are saying some truth about (x, y), how to distinguish them from others, and how to use it as a way to generate the feature of this point, to use it as a weapon, to find facts that is also true for little (x, y)

2.2.2 Hierarchy of Formal Languages

2.2.3 Pattern Recognization

2.2.4 Feature Extraction

2.3 Composition with Abstraction

2.4 Transformation with Abstraction

2.4.1 Confident Extrapolate

From one instance to whole class

To determine which part is $x + y \geqslant 5$, upper or lower of $x + y = 5$, we only need to plug in a number that is not on the line, say $(0, 0)$, since $0 + 0 \ngeqslant 5$, we can deduce that the shaded area should be the upper part of the image, as shown

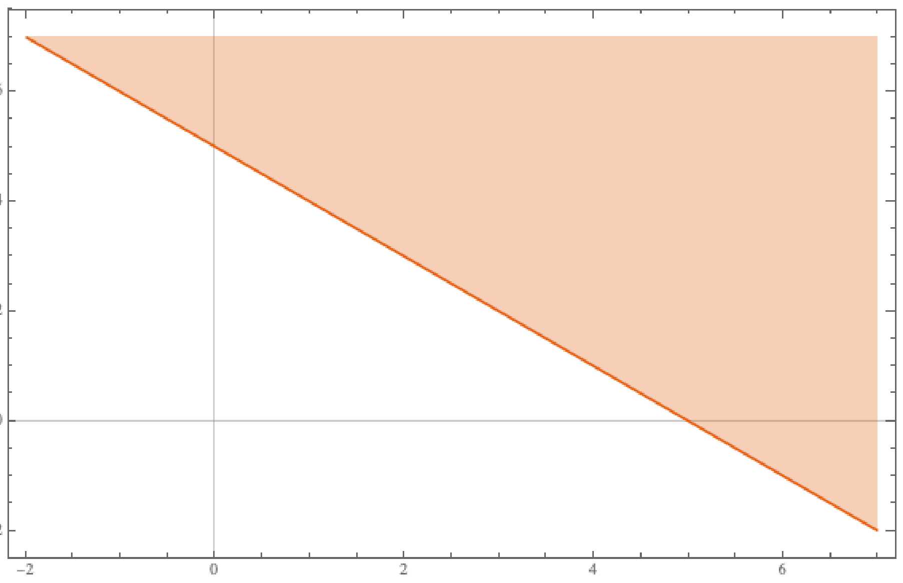


Figure 2.1: Plof of $x + y \geqslant 5$

in Figure 2.1 This single one instance is sufficient to represent the whole class, in this case we can use one to proof the all.

From several instances to whole class

2.5 Equivalence of Computation

Everytime you refactoring your codes, you are assuming the functionality of the codes with be the same for the situation (if there is no bugs).

But why you believe that is actually a fact. Why do you think any array push any object and pop it with return the same array all the time? How can you let the computer to know it? Why in mathematics we have to give some result as axioms and assume they are right without asking more. Why the proof of mathematics has to start with some assumed result, but not a proof with only definition. Shouldn't we only trust definition if we speak of truth?

The blessing of abstraction is from the rules of nature. That the world is not chaotic, it's capable of reproduce the regularity and the rule is quite reliable whenever and wherever in the world. You can design a product and reproduce it in any part of the world. You can write a website on your slow and old Macbook, and the whole world would visit it with the same rendering result on their own browsers. As I grow up, I realize the science works, the plane can fly, and when you are in the NYC, and make a phone call to China in the afternoon, your friends won't lie to you that they're in the midnight. The stars follow Newton's Law.

Equivalence of computation is everywhere in computer science. But why? Here is an approach to explain this.

As I stated earlier, the abstraction is based in physics, that you can reliable reproduce things if the setup is the same. The reason of equivalence is lying in the definition of computation. What do you mean by computing? And a more important thing is that what you mean by saying:

It's true for any array. What is ANY ARRAY? While as I spend so more inks on distinguishability and indistinguishability, it's their time to on the show. When I say any array, I mean it does care the length of the array and the each item of the array.

Let me give you an example of ANY. Imagine we have four floors, you are at the first floor, your instruction is that go to the third floor. Well, here is the trick, you don't care what's in the second floor, anything will work as long as the floor is there for you to distinguish the third floor. And you just assume an instance of the problem. It will work. Give an example, you will prove the whole class.

Another example is that, when stack pointer is at RAM location (%eax), and the next instruction is to fetch the (%eax)+2, you don't care any content local in (%eax)+1. You just don't care in the instruction. When you don't care the entire array elements, you have a proof for the equivalence of computation. Now let's refresh what array.push(x) do, it keep ignore the contents of the entire array and reach the buttom the array and add an elements. So any array will represent the whole class. Now, if we want to pop the last element, since we add the element at the end of the sequence, it doesn't touch or care or rely on the content of the array, that means the whole array is not disrupted. Then that means any instance of an array will represent the whole class. If the computer want to prove it, it simply shows that one instance works, so the whole class works. The program can add some flags, that indicate, does the instruction is changing the flag or not? if at the end the flags are remain unchanged, say it doesn't care the length of an array and the content of it. An instance is sufficient enough to show the class level facts.

While in reality, at least to me, it happens all the time, when I try to solve a problem, or when I work out some result I can see that I try to find the invariant part in the process, does it care its content, should I release the requirements, how can I generalize the thing with less constraints. I work on the instance and try to find pattern, try to find what is required, does the instruction really cares about the length or content. The answer is in the definition, not in the axioms. If you are using axioms, you are giving a new definition not a new proof, not really explaining, but hard rote the result like antient people believes some God's angry brings the anormaty in the sky.

The whole book is about there things, distinguishability, event-triggered transformation(under what condition do what) and abstraction. Abstraction is basically a way to say how restricted we want to say two things are the same or different. So the first and the third are the same thing, that means the whole book is all about computation and abstraction. And abstraction is way more important that computation in terms of solving problems, it's faster and smarter, it's real intelligence, where computer is merely a way to replace hard labour.

The indistinguishability is implicitly defined in the definition of a specific transformation. The flags are there too. The things we have to do is to track to flags along the way to find out whether the transformation is distruptive or not. When you say for all array, you mean it still has to be an array, the definitive features has to be distinguishable, but other features are discarded, or make it indistinguishable. When you say all men has to die, you mean that the features of man is remains to distinguish, but the individual features are indistinguishable. When you say for all numbers a + b = b + a, you mean the counting procedure is distinguishable but the length of the objects is not.

2.5.1 One Instance to Represent All

The reason you use examples is that because of theorems are te same for the whole calss, os it as to be true for any instance, so you can use one instance to represent the class level facts.

SPECIAL and SPECIFIC. One is for distinguishability, the other is for classification. I believe these two are linked together.

The title if I was writing a thesis would be *On distinguishability and equivalence check, the first step to abandon the axiomatic foundation of mathematics.*

Here comes one of the greatest eureka of my life.

You can give an object mulitiple abilities, and you can define classes, which are basically a Type 4 definition, with an output of True.

And let's say what we are talking about is natural numbers, we can proof that for all the natural numbers certain rule holds. And when we can have several class level abilities.

First, we must be able to proof and show the class level abilities.

Second, we examine the process of instance transformation, along the way, we notice that for each step of transformation or aruguments, we are not touching its special ability that belong to it, e.g. 2 + 1 = 3 where a + 1 don't have such an ability, if we find this thing is actually happening, then we can make sure that one instant can show a whole class.

One instant to proof all instance with the same ability.

This is not the same as axiomic approach, where we only have limited ability to begin with, we can define infinite ability as we wish and we can explore what the combination of these abilities can go.

This is exactly what human do, as I believe, because as we discussed before, that we know that human thinks in terms of tightness of abstraction, our nature is to distinguish and find pattern, so this huge pattern of ability is fundamental to think abstractly.

For x in $x + 2$, comparing to x, 2 has more talents, it contains more information it has more power to transform.

Where x has every traits of transformation, as other real number but it can't add to high-resolution guys, it has limited abilities.

If a has some ability, then a is an A. If a is A, a has the ability.

The hard problem here is that how to proof that a class have a property that is true for all of the instance of its. This is so fundamental, it lays the foundation of mathematics. axims works like that the basic ability we take for granted and deduce all facts or ability from these. But actually we can prood these axiomic abilities too and find more abilities from it.

2.5.2 50 Examples of Equivalence of Computation

Let's see an example:

Example 2.5.1. $\sin 20° \cos 10° + \cos 20° \sin 10°$

This is a computation problem. Which means in principle you could calculate it out if you have to assign it a number. The result would be approximately:

$$0.3420201433256687 \times 0.984807753012208 + 0.9396926207859084 \times 0.17364817766693033$$

and the result is:

$$0.49999999999999993220744841205137\,2$$

But also you can transform it into $\sin(20° + 10°)$, using a recognizer and transformer similiar like:

$$\sin x \cos y + \cos x \sin y = \sin(x + y)$$

And then the result becomes $\sin 30°$, which is 0.5, an exact result. This type of phenomenon is everywhere in mathematics. This fundamentally distinguishes the computing and transformation. And this is the core talking in this section.

But why $\sin x \cos y + \cos x \sin y = \sin(x + y)$? Why when you compute in two different ways can give you the same result for every x and y? In mathematics, you could find lots of identities. This type of identities is the equivalence of computation. And in solving the mathematical problems, you have to make this kind of transformation a lot, which is also called symbolical manipulation.

Another example, what is the period of:

$$f(x) = \sin x \cos x + \cos^2 x$$

This is an extremely hard or even unsolvale problem if you don't reduce this problem to another. You can't enumerate all the possible integers, and you can't even to enumerate all famous irrational numbers, like e^2, $2e$, 2^e. The only possible way to solve this problem is to reduce this problem to a solvable problem. Which in this case:

$$f(x) = \frac{\sqrt{2}}{2} \sin\left(2x - \frac{\pi}{4}\right) + \frac{1}{2}$$

Then, the answer is now obvious to solve, π. Let's think it again. Can you solve the problem without transformations? That's a problem with ultimate complexity and maybe we could design a problem which could be inherently unsolvable.

This is also a way to see the Godel's incompleteness theoroms. You can't use one tool to rule them all. The tool can only penetration a small portion of well defined problems.

Example 2.5.2.

$$\begin{aligned} x &= 1 \cdot x \\ x &= x + 0 \\ 1 &= \frac{x}{x} \end{aligned}$$

2.5.3 Analogy

2.6 Computation with Abstraction

2.6.1 Symbolic Computations

Using the symbols to denote an instance with special properties.

Example 2.6.1. If the addition of the imaginary and real part of $z = a + 2ai$ is 6, what is a?

Example 2.6.2. If $\log_2 x = 3$, what is x?

Example 2.6.3. If the left focus of $\frac{x^2}{4} + \frac{y^2}{b^2} = 1$ is $(1, 0)$, what is b?

Example 2.6.4. $\int_1^a x^2 \, dx = 9$, what is a?

Example 2.6.5. If the average of $a, 2, 3, 1, 6$ is 4, what is a?

Following examples require setting up more equations.

Example 2.6.6. For arithmetic sequence a_n, if $a_3 = 3$, $S_4 = 10$, what is a_n

Example 2.6.7. If a circle passes points $(1, 2), (3, 4), (5, 6)$, what is the equation of the circle?

2.6.2 Computational Universe

Chapter 3

Problems

3.1 Problems

To define a problem is defining a class at the same time.

To understand a problem is to know how to check whether one solution is correct or not.

3.1.1 Verifying versus Searching

3.2 Types of Problems

3.2.1 Computational Problems

3.2.2 Learning Problems

3.2.3 Generic Problems

3.3 Techniques

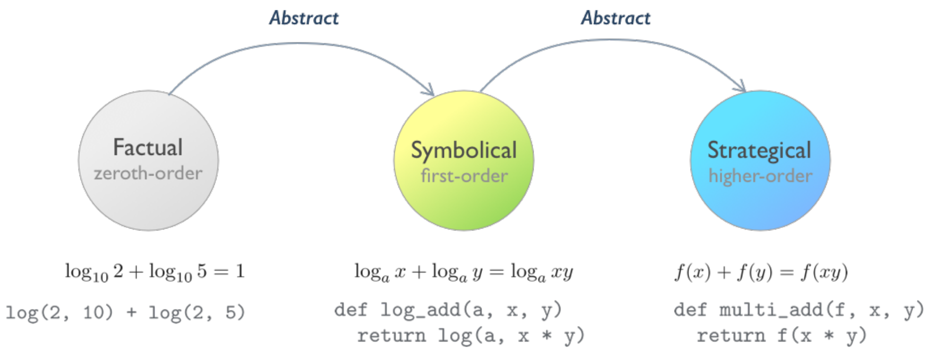
The Classification of Techniques is a core concept in daily life.

3.3.1 Robustness of Techniques

3.3.2 Frequency of Usage

3.3.3 Ad-Hoc versus Generic

One phenominon that I experience a lot is the way people don't distinguish the difference between a ad-hoc technique and a generic technique. When a teacher or a guy present a way to solve a specific problem, few people will appreciate the robustness of this technique, that how well can you transfer your ability on this learning to another situations. But because of their inability to distinguish hardness and robustness, that leads to a dangerous place, that you have to learn



```
log(2, 10) + log(2, 5)
```

```
def log_add(a, x, y)
  return log(a, x * y)
```

```
def multi_add(f, x, y)
  return f(x * y)
```

Figure 3.1: Abstraction Levels

too much techniques to cope with every problems, but you don't have enough time for learning these. And the teacher may think by doing an ad-hoc problem will lead to a better understanding for a more generic solving ability, which is vague. You don't learn too much on ad-hoc problems. You only know how to solve it in a nearby situations. The overall hint on a more generic solving strategies are often weak to recognize.

People don't know how to appreciate the robustness of techniques. Only can they find the dramatic changes of events. Even you simply press the button, you believe the underlining changes belongs to your smartness, well the main reason is the engineering behind the scene to smooth the user experience.

3.3.4 Intensity of Signal

3.3.5 Trainability of Techniques

Both the solving radius and the applicability.

3.4 Types of Techniques

3.4.1 Factual Techniques

3.4.2 Algorithmic Techniques

3.4.3 Strategic Techniques

3.4.4 Generic Techniques

3.4.5 Expansion Techniques

Chapter 4

Complexity

4.1 Computational Cost

4.1.1 Computational Cost in Time

4.1.2 Computational Cost in Space

4.1.3 Computational Cost as Human Pains

4.1.4 Sensitivity of Computation Cost

The opportunity cost is everywhere in life. You can't do everything as perfect as you want and you have to make the tradeoff like all the time. But human I think have the ability of sensing this computation cost.

4.2 Reduction of Computation

There are two types of reduction. 1. Reduction by String Replacement. 2. Recognize as a class object and reduction by Replacement.

4.2.1 Reduction by Shortcuts

4.2.2 Reduction by Theoroms

4.3 Computational Irreducibility

4.3.1 Principle of Least Computation

4.3.2 Cecullar Automata

4.3.3 Gödel's Incompleteness Theorems

4.4 Depth of Complexity

4.4.1 Learnable Regularity

4.4.2 Explorable Complexity

4.4.3 Meaningless Chaos

4.4.4 Lucky Configurations

Back to the past, you could be able to buy bitcoin at year 2010, that means you can proactively find bitcoin and buy a large amount of it. But that only means things have already been rolled out.

4.4.5 Scale of Science

4.5 Randomness and Probability

4.5.1 Incomputable Situations

4.5.2 Lack of Information

4.5.3 Distinguishability in Probability

Chapter 5

Learning

Both Machine and Human Learning

5.1 Parse and Compile

TO much time we believe beauty and smartness is in the concise way.

That's only because it's easy to parse, use less data, less to compute, less to distinguish, the beauty is the lazyness, the beauty is the thrift on the limited resource we save for living. Thinking is so demanding and dangerous, pursue beauty, pursue a easier parser.

5.1.1 Hardness of Distinguishability

The strength of neuron connections is for event causation, not features. The features are how distinguishable a representation can be.

5.1.2 Local Environment

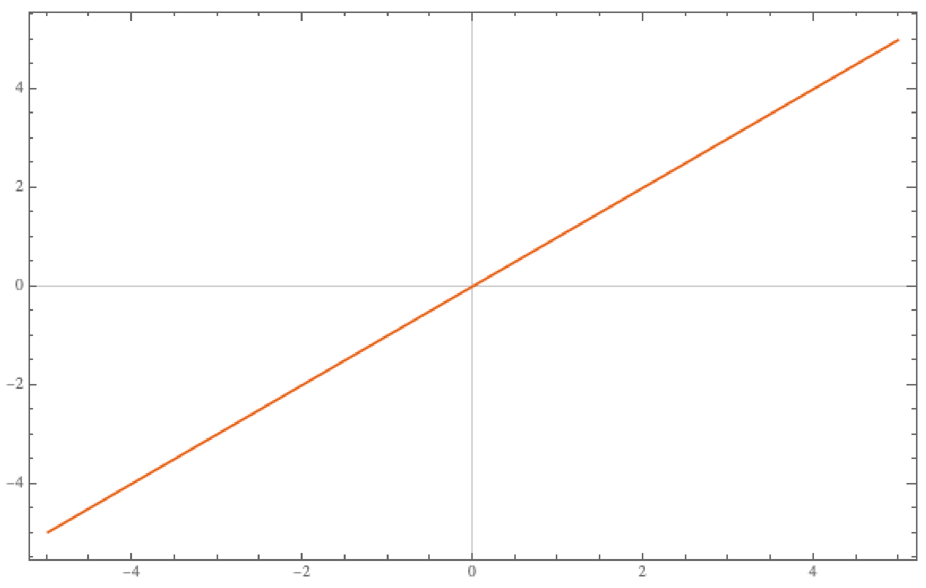
5.1.3 Change of Representation

We can display a line as following ways:

$$y = x$$

$$\theta = \frac{\pi}{4}$$

$$\begin{cases} x = t \\ y = t \end{cases}$$



5.2 Correlation and Fitting

5.3 Causation and Inference

5.4 Mapping and Modeling

People without sufficient training tend to use a simplified model for the world.

The if $f(a) = positive_result$, people not only believe $f(A) = positive_result$, but also $f(B) = positive_result$. This reduction is so bad that people always blame for the wrong direction. But that also gives engineers a harder problem to solve, you have to solve an over-kill product.

5.5 Graph and Linking

5.6 Smartness of a Human Learner

5.6.1 Short-Term Memory

5.6.2 Long-Term Memory

5.6.3 Sensitivity for Feature Extraction

5.6.4 Autocorrection for Flawed Information

5.6.5 Intensity of Learning Desire

The desire is generated when problem is at hand, this is my personal experience, I don't know the meaning of learning several subjects. The reason I take courses is because I have to to get my degree, while when I start to solve the real world problem I started to understand that the knowledge and the skill that I learn is to let myself be able to solve the real world problems to let my desire or imagination becomes to reality.

5.7 Training the Human Learners

From vague, inconfident, chaotic, erroneous, strange to clear, confident, accurate, fluent, familiar.

5.7.1 Conceptual Leaps

5.7.2 Training the Robust Techniques

Most of the time, the resaon you solve a problem that you consider it as hard is because you have practiced it very hard.

Practise gives you lots of pleasure and also a data to induct a rule from.

5.7.3 The Necessity of Repetition

The retention is the mother of repetition. And people have conflict feelings about training the robust techniques. At one hand, people encourage people to practice, especially delibrate practice. On the other hand, you find people think creativity trumphs recipe type procedure.

5.7.4 Training the Generic Techniques

It seems like the education institution wants to teach pupils about real thinkings instead of rote memorization. And the reality is that it's just the generic techniques with weak robustness, the teachability seems low, since there are differences between human individuals, some learners can perform way better than the others.

5.7.5 Learnability for Human Learners

If the students are capable to do something without the help of the teachers, the Necessity of education is self-learning for the students for the rightful accessability for this learner. But for the other students what they need is a way of learning with proper level of accessability not others.

5.7.6 Accessability of Training Tools

Even with the worst training tool, the brightest kid can learn fast.

The learning materials like roads that some are smooth and some are murdy, and the learner is like the car with different engines and tires. Some are very powerful and some are not.

The progress of human education is essential a way to smooth the road. That once bumpy sand road change to cement one. One great example in the multiplication of roman numbers, Roman has the right of pride, but its number is truly not efficient to calculate. The arrival of the Arabic number, along side the invention of logrithm, makes normal people able to learn more, to break the barrier once unnecessarily thick and trainned more and be able to make more and contribute more to the progress of the human society.

Even in our ages, I don't understand lots of the training neccessaty for the time wasting competitive purpose. That we have to curve students so some of them have the right of the limited resources. The misleading part of this is that lots of people can't even be trainned sufficient to their intelligence.

We are always amazed how human can beat computers, but only in a category of stuffs. And we are also troubled by educating the real intelligence. That means sometimes you can't explain with programming, you can't explain to people.

So I found this interesting is that the teachability and computability. People are good at pattern recognization, which as discussed earlier is the result of the ability to find distinctions and determine the degree of distinction.

What is the differency

5.7.7 Limitations of Training

The robustness of a technique, and the solving radius of it, should be a center topic of education and solving algorithms.

Normally, as I personally experienced, that more robust a technique appears, the solving radius is smaller. Or maybe because we consider it as small not essentially small. Once you invented a new technique, people could use it to solve problems, but also they could find the situations where the technique is not suitable, you need a new trick or most of the time, a higher level of strategy with bigger solving radius, but this is vague. That means you have to master a skill that you only practice once, or you have to paint a picture by seeing it for a few seconds, the signal are so weak, the possible directions are so many, you couldn't choose where to go and you can't give a suitable instance to satisfy your need.

Here we can give you a solid example in solid geometry:

How to find the dihedral angle of given two plains. The technique to solve is straightforward, you just find two surface normals, and the angle of these two is nearly the answer.

But what tricky is how can you find the two surface

You can't train too much abstract techniques, especially generic ones. But there are difference of intelligence among learners which set us apart.

5.8 Testing the Performance

Definitions

1. Able to recognize the definition in the context
2. Able to replace with definition to produce a new state
3. Able to verify whether one instant belongs to a class
4. Able to provide instants of a class

Axiom-Like Conclusions

1. Understand these are the derivative, not truth by default
2. Able to remember the result
3. Able to recognize and apply in the context

Algorithmically

1. Able to recognize the occurance of the problem
2. Able to remember the procedure fluently
3. Able to apply each step of computation reliablely

Sometimes, we can use this chapter to denote the work type in real life.

Some jobs solve the problem near algorithmically, like bus drivers. The taxi drivers have to deal with more abudant situations than bus drivers.

The job of a cook is also most of time algorithmical, producing the food according to recipes reliablely everyday to satisfy the needs of customers.

Symbolically

1. Able to use symbols to denote otherwise concrete concepts
2. Able to deduce and work comfortably using symbols
3. Able to use theoroms and equations to get the solutions

Chapter 6

Solving

6.1 Overview

The main source of seraching is similarity and familiarity.

Give me a point on $f(x) = \frac{3}{2}x + 1$, the initial responce would be the point $(2, 4)$, the reason is not defined in the strict instruction, but in the process of human likeness of similarity.

6.1.1 Core Concepts in Solving a Problem

This is more convinient to say about solving a typical problem. To say it as typica l, I am neither refer it as a knee-jerk reaction, nor as a tough math problem requiring hours to solve, but maybe take a few minutes no longer than half an hour to solve.

Here are some important notes:

1. **Parse** the problem string into an array of **signals**.
2. **Signals** contains the conditions and the purposes of the problem.
3. **Signals** are processed by **rules of transformations**.
4. **Transformations** has many kinds, notably, **Mechanical Transformation**, **On-Purpose Transformation** and **Heuristics Transformation**.
5. **Mechanical** one is the process of seeing A, transform something to B.
6. **On-Purpose Transformation** is the procee of to see A, transform something to B.
7. **Heuristics Transformation** is in the middle of the above twos. It always involves nondeterministic path choosing. The principle of heuristics is that by trying one particular avenue, it will save much computation comparing to other options.

8. The sequences of the transformation will lead you from conditions as the starting point to purposes.

6.1.2 Examples to Find a Path

Example 6.1.1. Simplify: $4\cos 50^\circ - \tan 40^\circ$

Possible Moves Revisit

Don't revisit, simply in the short and long range search. How intimidating it is for a newcomer to a new city.

6.2 Principle of Heuristics

6.2.1 Computational Cost Saving

6.3 The Analogy of City Exploration

Since to solve is to search, searching is basically exploring your searchable area.

You start with a newly assigned treasure hunt game. You start you search by going through the known routes, following the familiar signposts, try the nearby stores to save your energy, you find a famous gathering place like a museuem or a park.

You start to find the easter egg by visiting the hinted places, and you get there by some known and familiar routes, you can search nearby the hinted places, since you know how to get there and assume you can find there will be considered as being lucky. Sometimes you have to find your own routes by noticing the signposts. And since you might visit the city multiple times, you will gradually remember the notable objects. When you on a road, you will automatically complete the routes without thinking or even chatting with others, until a puzzled crossroad is encountered.

Let's compare the solving route as how much roads you have to travel. And the fluency of your technique mastering is a value of resistance, low value with well trained, and high for poor practice. That might be the case in our brain that the smooth of a chunk is basically the electric current with higher value. And the brain choose load through those low resistance route to save computation energy.

6.4 Local Expansion

6.4.1 Definitions

Let's revisit and see all kinds of definition all over again.

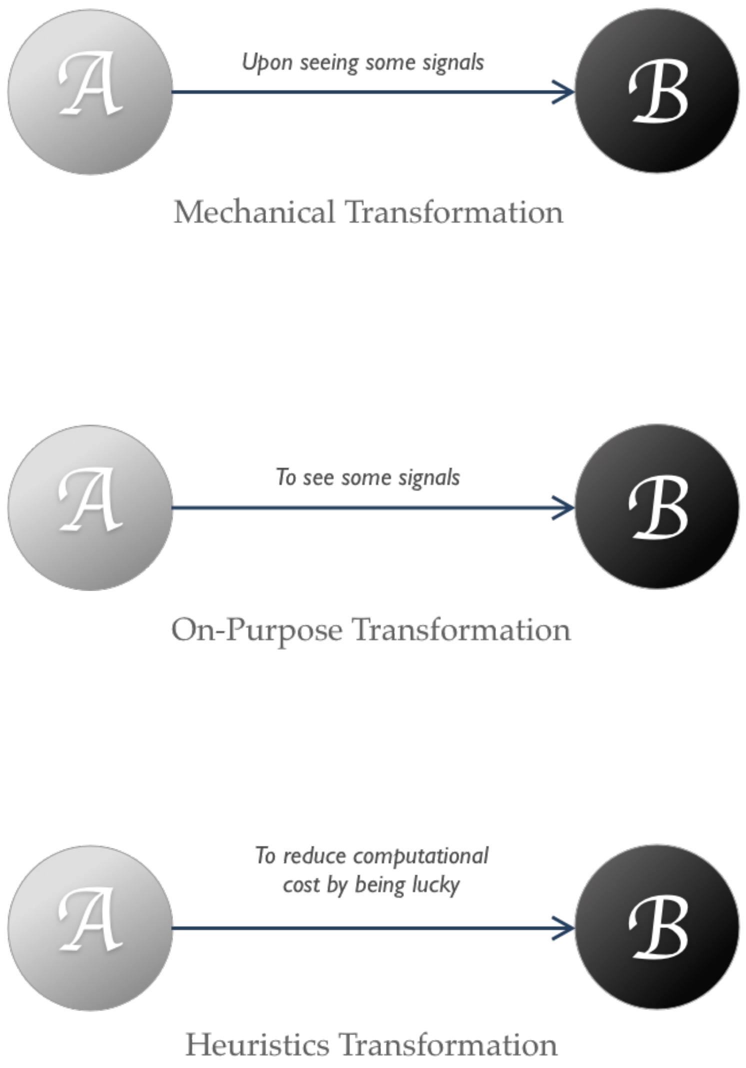

A
Upon seeing some signals
B
Mechanical Transformation
A
To see some signals
B
On-Purpose Transformation
A
To reduce computational cost by being lucky
B
Heuristics Transformation

Figure 6.1: Types of Transformation

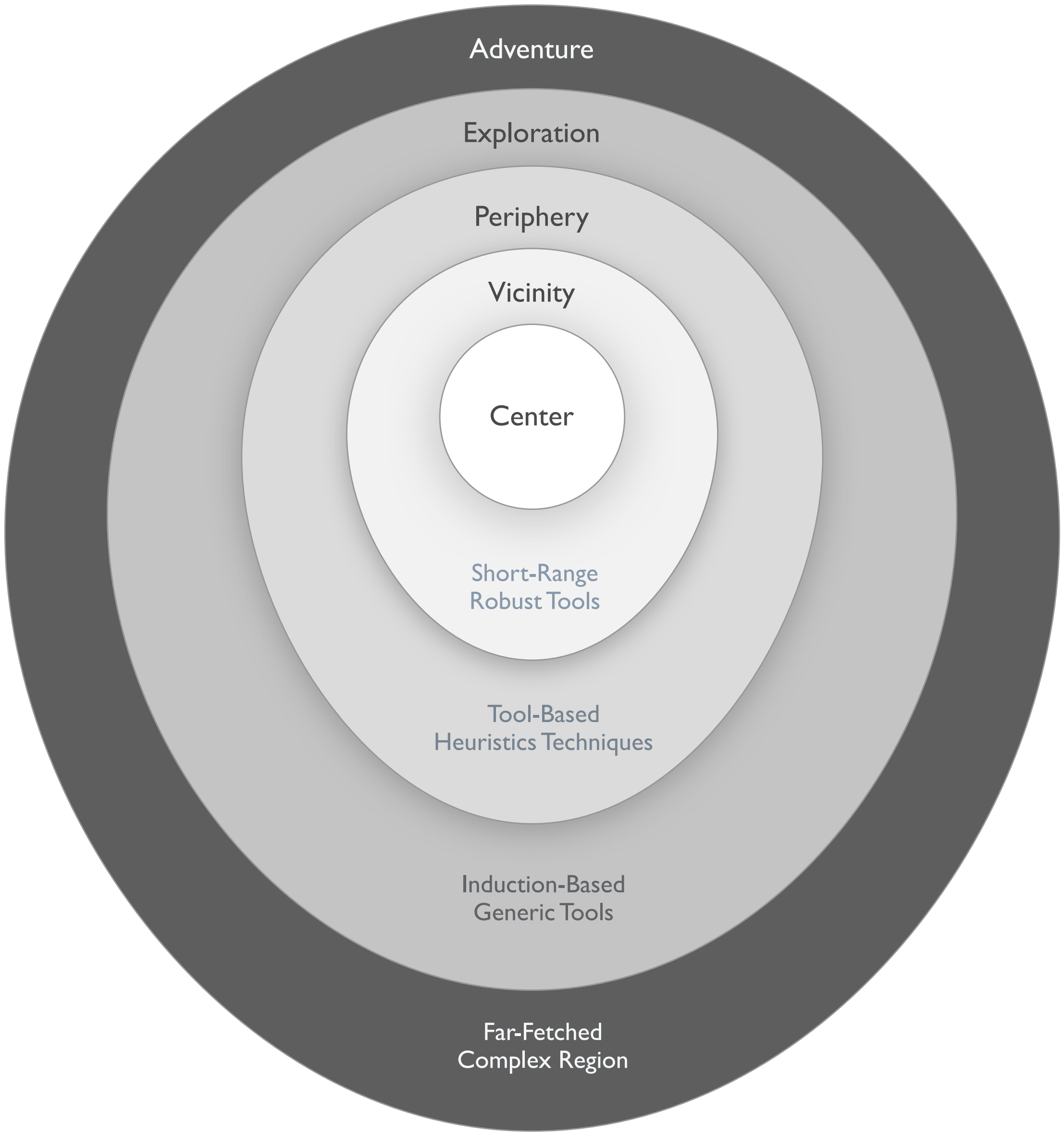

Adventure
Exploration
Periphery
Vicinity
Center
Short-Range Robust Tools
Tool-Based Heuristics Techniques
Induction-Based Generic Tools
Far-Fetched Complex Region

Figure 6.2: Types of Transformation

Replaceable Definitions

As we use them all the time in programming activities:

Example 6.4.1. $b = 2 \times 4$

Example 6.4.2. $f(x) = e^x$

Example 6.4.3. $S_n \equiv a_1 + a_2 + a_3 + \cdots + a_n$

Example 6.4.4. $n! \equiv 1 \cdot 2 \cdot 3 \cdots n$

Example 6.4.5. $C_n^k \equiv \dfrac{n!}{(n-k)!k!}$

Example 6.4.6. The eccentricity of $\dfrac{x^2}{a^2} + \dfrac{y^2}{b^2} = 1$ is $\dfrac{c}{a}$

Classification Definitions

Define things as in set theory:

Example 6.4.7. A set of points on a circle: $x^2 + y^2 = 1$

Example 6.4.8. $i^2 = -1$

Example 6.4.9. $\log_2 3$ is a number x where $2^x = 3$

Example 6.4.10. Odd function: $f(x) = f(-x)$

Here are some possible problems involve definitions.

Example 6.4.11. Give an instance of prime numbers

Example 6.4.12. Is 3 a prime number?

Example 6.4.13. Is $f(x) = x^3$ an odd function?

Example 6.4.14. What is the fourth elements of $a_n = n^2$

6.4.2 Properties

Properties are short and frequently used theoroms. Normally, we treat the properties as if they are axioms.

Example 6.4.15. Simplify: a^3a^2

Example 6.4.16. Simplify: $\log_{10} 5 + \log_{10} 2$

Example 6.4.17. Simplify: $\sin 20^\circ \cos 10^\circ + \cos 20^\circ \sin 10^\circ$

Example 6.4.18. Find the vertex of the parabola $y = -2(x+3)^2 + 4$

Example 6.4.19. $\sin^2 x + \cos^2 x = 1$

Example 6.4.20. $a^2 = b^2 + c^2 - 2bc\cos A$

Example 6.4.21. $\log_a(xy) = \log_a x + \log_a y$

Example 6.4.22. $\binom{n}{k} = \binom{n}{n-k}$

Example 6.4.23. $(a+b)^2 = a^2 + 2ab + b^2$

Example 6.4.24. $(e^x)' = e^x, (\ln x)' = \dfrac{1}{x}$

6.4.3 Alternative Forms

6.4.4 Translation

In lots of situation, the proof of one result requires long sequences of derivations.

$$u \Rightarrow u_1 \Rightarrow u_2 \Rightarrow \cdots \Rightarrow u_k \Rightarrow v$$

But because of the frequency of usage, we tend to remember it as a shortcut.

$$u \overset{*}{\Rightarrow} v$$

We can call these derivations as translation, since it seems exotic and requires recognization and transform to another string. And for translation, we could name the list of translations as vocabulary. To distinguish them by the frequency of usage, we could name them as, **basic vocabulary**, **essential vocabulary**, **advanced vocabulary**, **extended vocabulary** etc.

Here lists some examples of mathematical vocabulary:

Example 6.4.25. In ΔABC, D is the middle point of BC, and $BC = 2AD \Rightarrow \angle A = 90^\circ$

Example 6.4.26. In ΔABC, point M satifies $\overrightarrow{MA} + \overrightarrow{MB} + \overrightarrow{MC} = \mathbf{0} \Rightarrow M$ is the geometric center of ΔABC.

Example 6.4.27. In ΔABC, $AB = AC \Rightarrow \angle B = \angle C$

Example 6.4.28. Point $P(x_0, y_0)$ is on curve $y = f(x) \Rightarrow y_0 = f(x_0)$

Example 6.4.29. If points A, B, C is on the same line $\Rightarrow k_{AB} = k_{BC}$

Example 6.4.30. $\overrightarrow{PF_1} \cdot \overrightarrow{PF_2} = 0 \Rightarrow P$ is on the circle whose diameter is F_1F_2

Example 6.4.31. Points A, B is on $\dfrac{x^2}{a^2} + \dfrac{y^2}{b^2} = 1$ and M is the middle point of $AB \Rightarrow k_{AB} \cdot k_{OM} = -\dfrac{b^2}{a^2}$

Example 6.4.32. $\sin\dfrac{\pi}{12} \Rightarrow \dfrac{\sqrt{6}-\sqrt{2}}{4}$

6.4.5 Decoding

Sometimes, we don't always know the association between two propositions. We need to find it out by ourselves.

Example 6.4.33. Given $f(x) = |\log x|$ and $f(a) = f(b)$

6.4.6 Symbolic Techniques

Let's start with a simple example:

Example 6.4.34. Simplify $\frac{10}{2-i}$

While if you only know the definition as $i^2 = -1$, it will still be a highly difficult problem to solve. And you would find it hard to remove i in denominator. But if you learn a technique, this problem would be easy. What you do would be confidently compute without thinking.

Here we would discuss the importancy of robost techniques, especially these symbolic techniques.

Example 6.4.35. Solve $\begin{cases} x + y = 10 \\ x - y = 2 \end{cases}$

Example 6.4.36. For $a_n = n2^n$, what is its sum S_n?

Example 6.4.37. Solve $2x^2 - x - 1 = 0$

Example 6.4.38. If $x + 2y = 3$, what is the minimum of $\frac{2}{x} + \frac{1}{y}$?

Example 6.4.39. In ΔABC, $A = \frac{\pi}{3}, a = 2$, what is the range of its circumference?

6.4.7 Combination

Example 6.4.40. In ΔABC, $\angle C = 90°$, $AB = 4$, $AC = 1$, find $\cos B$

Example 6.4.41. Line l goes through the center of $x^2 + y^2 = 2x$, and slope is 1, what is the equation of l?

Example 6.4.42. If the fourth element of sequence $a_n = kn^2$ is 4, what is a_5?

Example 6.4.43. If the derivative of $f(x) = ax^3$ is $f(x) = x^3$, what is $f(1)$?

Example 6.4.44. If $x^2 + 2mx + 1 = 0 (m > 0)$ has exactly one real root, what is $\log_a m$?

6.4.8 Solve Like a Fox

Don't try to generalize too early, and don't try to be perfect early on. The reason is simple, the cost is too much and the resource is limited. We can explore everything and make perfect choice at each step, the only way we could do is moving along. The wisdom is not even doing your best but to do the thing you can do, making progress instead of waiting for the right time or making the perfect match to your problem.

You can come up an idea and try it, then modify it later, and even you are improving it by visiting it multiple times. This strategy is simple but maybe powerful. It released the tension of limited resources. And it guesses along or maybe even believe its own ability that the result is at the reach and improve its understanding (I should define this word more precisely) along the way.

Solving a problem with ad hoc techniques may give the first hint of generalizations. But this route might be easier using the brutal force method rather the abst ract way thinking. We'd better revist the problem after solving, which will be a separate topic in the future.

The way of heuristics works in this book is that: in the indistinguishable path choosing situation, by assuming something may work,

6.4.9 Increase the Similiarity

Example 6.4.45. If $2a + 2^a = 5, 2b + 2\log_2(b-1) = 5$, what is $a + b$?

You can solve a, which is around 1.28315, and you can also get b, approximately 2.21685. And the addition of these two numbers is exactly 3.5. The important part of the transformation is to change the first part to $2a+2\cdot 2^{a-1} = 5$, so the two expression can have a symmetric look.

6.5 Tool-Based Adaption

6.5.1 Creation of Desired Vision

An Example

Let's start with an example:

Example 6.5.1. If $x + 2y = 3$, what's the minimum of $\dfrac{2}{x} + \dfrac{1}{y}$?

Let's make us understand the problem. The condition says that x, y satisfies:

```
def some_class(x, y)
  return x + y == 3
```

Then, what is the minimum value comes out from:

```
def some_computation(x, y)
  return 2 / x + 1 / y
```

Let's combine these two:

```
def find_some_value()
  all_x, all_y = all_real_numbers()
  all_satisfied_instances = []
  for x in all_x and for y in all_y
    if (x + y == 3)
      all_satisfied_instances.append( 2 / x + 1 / y )
  return find_maximum_value_of(all_satisfied_instances)
```

This is insolvable numerically, it can only be solved symbolically in terms of class level of manipulation.

Here we go. First the *find_some_value* function is equivalent to:

```
def find_some_value()
  all_x, all_y = all_real_numbers()
  all_satisfied_instances = []
  for x in all_x and for y in all_y
    # ======== CHANGED ======== #
    if (1 / 3 * (x + y) == 1)
    # ======== CHANGED ======== #
      all_satisfied_instances.append( 2 / x + 1 / y )
  return find_maximum_value_of(all_satisfied_instances)
```

Then,

```
def find_some_value()
  all_x, all_y = all_real_numbers()
  all_satisfied_instances = []
  for x in all_x and for y in all_y
    if (1 / 3 * (x + y) == 1)
      # ======== CHANGED ======== #
      all_satisfied_instances.append( ( 2 / x + 1 / y ) * 1 )
      # ======== CHANGED ======== #
  return find_maximum_value_of(all_satisfied_instances)
```

Then,

```
def find_some_value()
  all_x, all_y = all_real_numbers()
  all_satisfied_instances = []
  for x in all_x and for y in all_y
    if (1 / 3 * (x + y) == 1)
      # ======== CHANGED ======== #
      all_satisfied_instances.append( ( 2 / x + 1 / y ) * 1 / 3 * (x + y) )
      # ======== CHANGED ======== #
  return find_maximum_value_of(all_satisfied_instances)
```

Then,

```
def find_some_value()
  all_x, all_y = all_real_numbers()
  all_satisfied_instances = []
  for x in all_x and for y in all_y
    if (1 / 3 * (x + y) == 1)
      # ======== CHANGED ======== #
      all_satisfied_instances.append( 1 + 1 / 3 * ( 2 * y / x + x / y ))
      # ======== CHANGED ======== #
  return find_maximum_value_of(all_satisfied_instances)
```

Then,

```
def find_some_value()
  all_x, all_y = all_real_numbers()
  all_satisfied_instances = []
  for x in all_x and for y in all_y
    if (1 / 3 * (x + y) == 1)
      # ======== CHANGED ======== #
      all_satisfied_instances.append( 2 * y / x + x / y )
  return 1 + 1 / 3 * find_maximum_value_of(all_satisfied_instances)
  # ======== CHANGED ======== #
```

Then,

```
def find_some_value()
  all_x, all_y = all_real_numbers()
  all_satisfied_instances = []
  for x in all_x and for y in all_y
    # ======== CHANGED ======== #
    # ======== CHANGED ======== #
    all_satisfied_instances.append( 2 * y / x + x / y )
  return 1 + 1 / 3 * find_maximum_value_of(all_satisfied_instances)
```

Then,

```
  def find_some_value()
    maimum_of_known_techniques = find_maximum_value_by_x_y('2 * y / x + x / y')
    return 1 + 1 / 3 * maimum_of_known_techniques
```

You may find it confusing, but this is neccessary if you think deep and try to make it accurate.

As you can see in this example, it's quite complicated to speak in terms of computing. And human brains would have compute this subconstiously or vaguely. If you would like to proof each equivalency, it will take lots of time and that's not feasible in computing. But maybe that's why human brains are so awesome.

And the reason that this won't work is we have to use more distinguishable way to reduce the calculation or in terms of function we need.

This transformation of data is quite unpleasing and troublesome. It simply so hard to even follow and to reasoning. While human might deduce with error and check it after the trial. We may have imitate that in our A.I. but make this A.I. more

6.5.2 Derivatives

When you want to know the result of $\overrightarrow{AC} - \overrightarrow{BC}$, you know it's either $\overrightarrow{AB}$ or $\overrightarrow{BA}$, but how do you get the truth?

Well, you would remember it through a remembered rule, like first to second, so it's $\overrightarrow{AB}$. But you can also get through

Analogy from Startups

When something phenomenal happens, people will start ask what ideas can also work by knowing the success of such improvements.

The success of Facebook gives people the imagination of how to solve problems by working in the dorm room.

The success of Uber and Airbnb, let people think about other opportunities by online sharing.

6.5.3 Declaritive Derivations

Let's consider two properties of an arithmetic sequence:

1. $S_{2n-1} = (2n-1)a_n$, for $n > 1$
2. $a_m + a_n = a_p + a_q$, if $m + n = p + q$

The condition of the followin problems is:

$$S_7 = 7$$

The direct use of the first tool:

$$\text{Find: } a_4$$

The combination of two tools:

$$\text{Find: } a_2 + a_4 + a_6$$

Here comes a little bit hard problem:

$$\text{Find: } a_1 + a_5 + a_6$$

In which case, you have to transform the first two to a_2+a_4, then the problem is reduced to the second one. But how can you think of this? In front of you is unlimited possible paths to choose. Even the reasonable ones will also be a lot. But why should we choose change the first two operands? Why we have to increase the first and decrease the second? It's solveable because of the small amount of the search area, but how can we increase the searching efficiency?

Here is the reason: We want to use the first properties since we know $S_7 = 7$, we get $a_4 = 1$ at hand. So we would like to expose the appearance of a_4 to explore more. With this in mind, we can also change the latter two operands to $a_4 + a_7$, through which we can also solve the problem at hand.

Now, let's consider a more difficult one:

$$\text{Find: } 3a_3 + a_7$$

We can also introduce a_4 by changing it to $a_3+a_4+a_2+a_7$, now the problem reduces to $a_3 + a_2 + a_7$, which is $a_1 + a_4 + a_7$.

After a few trial, we can induce that if the addition of all subscripts is divisible by 4, then the problem is solvable. The strategy would be keeping introducing the known tool of a_4.

6.6 Variable-Based Strategies

6.6.1 Completeness of Information

6.7 Trial-Based Induction

6.8 Multi-Layer Exploration

6.9 The Generic Solving Algorithm

6.9.1 Frequecy of Usage

6.9.2 Comfort of Familiarness

6.9.3 Toolbox Hierarchy

Analogy of a memory hierarchy.

6.9.4 Automatic Exploration

6.10 Reviewing the Solution

6.10.1 Hardness of a Problem

6.10.2 Evaluation of the Solution

6.10.3 Improvements of the Solution

6.10.4 Improvements of the Solving Ability

6.11 Smartness of a Human Solver

6.11.1 Illusion of Competence and Incompetence

When you read a popular book and watch a tutorial on YouTube, you seem like you understand the concepts quickly, and the subjects are easy and you can also remake it easily. But you are wrong for sometimes.

First, you can't reproduce the procedure reliablely, you just follow each step, rather walking the path yourself. When you solve the problem independently, you will find there are many crossroads and not sure which direction to go.

Second, you are able to understang the local procedure and remember it, but soon your memory will evapor and you have to recall and repeatly read or play.

Third, to invent is to choose, not to follow. There are so many combinations to work something out, but few are working and easy to communite. The masterpiece requires many choices normal people can't decide.

But sometimes you feel stupid when you study some subjects. But it's because the materials aren't good enough.

First, it might be too abstract too early on. Too few examples make you feel overwhelming and you can't use your feature extraction ability the evolution have provide. You are not a computer, you are more capable to do things current computer can't do, that's to find the pattern through data, find links between objects.

Second, if you can understand what a teacher teachs, but can't workout the similiar problems in the exercise, most of the time is that the teacher teaches you a problem that's easy to follow, but hard to solve, cause the teacher have the illusion of competence and he or she has the opportunity to read solutions and explain it to you, and he or she has more experience to choose from and maybe has more advanced strategies he or she might not even noticing using and can't teach you that. But the blame is on you, cause you can't solve the problem.

6.11.2 Basic Requirements for Being Intelligent

You have to be smart to understand the meaning of smartness. It's hard to distinguish from lucky and smart.

The ability of abstraction is in subconscious, and that is hard to touch and understand. And I guess this is also untrainable somehow. Subconscious is a blackbox, you can't really know too much about it.

6.11.3 Mysterious Human Aesthetic

6.11.4 Sensibility of Human Aesthetic

6.11.5 Imagining the Alternative Reality

6.11.6 Solvability for Human Solvers

You have to be healthy and relaxed to solve hard problems.

6.12 Designing the Solver

6.12.1 Benefits of the Solver

6.12.2 Revolution on the Foundation of Mathematics

6.12.3 Universal Algorithm of Distinguishability

6.12.4 Reinvent the Wheel

6.12.5 The End of Road

The only problem we need to solve to solve all problems is the problem of problem solving.

So, why not solve it?