

Isomorphism

– Mathematics of Programming



Xinyu LIU ¹

July 2, 2021

¹liuxinyu95@gmail.com, Version: 0.6180339887498949

Preface

Martin Gardner gives an interesting story in his popular book *aha! Insight*. In a country fair, there was a game called “Fifteen” on the carnival midway. Mr. Carny, the carnival operator explained to people the rules: “We just take turns putting down coins on a line of numbers from 1 to 9. It doesn’t matter who goes first. You put on nickles, I put on silver dollars. Whoever is the first to cover three different numbers that add to 15 gets all money on the table.”

1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---

A lady joined this game. She went first by putting a nickle on 7. Because 7 was covered, it couldn’t be covered again by either player. And it’s the same for other numbers. Mr. Carny then put a dollar on 8.

1	2	3	4	5	6	7	8	9
						nickle	dollar	

The lady next put a nickle on 2, so that one more nickle on 6 would make 15 and win the game for her. But the man blocked her with a dollar on 6. Now he could win by covering 1 on his next turn.

1	2	3	4	5	6	7	8	9
nickle				dollar	nickle	dollar		

Seeing this threat, the lady put a nickle on 1 to block his win.

1	2	3	4	5	6	7	8	9
nickle	nickle			dollar	nickle	dollar		

The carnival man then put a dollar on 4. He would win by covering 5 next. The lady had to block him again. She put a nickle on 5.

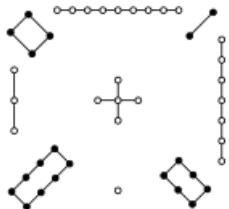
1	2	3	4	5	6	7	8	9
nickle	nickle		dollar	nickle	dollar	nickle	dollar	

But the carnival man placed a dollar on 3. He won because $3 + 4 + 8 = 15$. The poor lady lost all nickles.

1	2	3	4	5	6	7	8	9
nickle	nickle	dollar	dollar	nickle	dollar	nickle	dollar	

Many people joined to play the game. The town's Mayor was fascinated by the game, too. After watching it for a long time, he decided that the carnival man had a secret that made him never to lose the game except he wanted to. The Mayor was awake all night trying to figure out the answer.

The key to the secret can be traced back to 650BC. There was a legend about *Lo Shu* in ancient China around the time of huge flood. A turtle emerged from the river with a curious pattern on its shell: a 3×3 grid in which circular dots of numbers were arranged.



(a) Lo Shu Square

4	9	2
3	5	7
8	1	6

(b) Magic square of order 3

This is known as Lo Shu Square, a magic square of order 3. The sum of the numbers in each row, column and diagonal is the same: 15. For example, the sum of the first row $4 + 9 + 2 = 15$; and the sum of the third column $2 + 7 + 6 = 15$; the sum of the diagonal from left up to right bottom $4 + 5 + 6 = 15$. The insight to the carnival fifteen game is exactly the magic square. All three numbers sum to 15, form the rows, columns, and diagonals in that square. If the carnival man hold a secret Lo Shu map, he is essentially playing the tick-tack-toe game with other people.

4	9	2
3	5	7
8	1	6

(a) magic square

\times			\circ
\times	\times		
\circ	\times	\circ	

(b) tick-tack-toe game

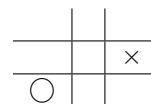
The game between the lady and Mr. Carny is equivalent to such a tick-tack-toe game. The man trapped the lady in step three. He could line up both a column and a diagonal. If the lady puts on 3, then the man could win the game by playing on 5. If you know a bit about game theory or programming, one will never lose the tick-tack-toe game if plays carefully. The carnival man with the secret Lo Shu square map does have the advantage over other people. As the fifteen game proceeds, the carnival operator mentally plays a corresponding tick-tack-toe game on his secret map. This makes it easy for the operator to set up traps of winning position.

This interesting story reflects an important mathematical idea, isomorphism. A difficult problem can be transformed to an isomorphic one, which is mathematical equivalent and easy to solve. A line of 9 numbers corresponds to a 3×3 grids; the sum target of fifteen corresponds to one of the rows, columns, and diagonals; Lo Shu pattern corresponds to magic square of order 3. This is what this book intents to tell: programming is isomorphic to mathematics. Just like in art and music, there are interesting stories and mathematicians behind the great minds.

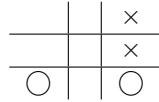
There is another further idea in this story: under the surface of the problem hides the theoretical essence, which is abstract and need to understand. With the rapid develop-

4	9	2
3	5	7
8	1	6

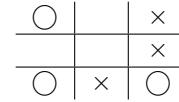
(a) magic square



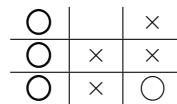
(b) Step 1, the lady puts on 7, the man puts on 8.



(c) Step 2, the lady puts on 2, the man puts on 6.



(d) Step 3, the lady puts on 1, the man puts on 4.



(e) Step 4, the lady puts on 5, the man wins on 3.

ment of artificial intelligence and machine learning, can we keep moving forward with a little cleverness and engineering practice? Are we going to open the mysterious black box to find the map to the future?

Liu Xinyu
May 2019, Beijing

Exercise 1

1. Implementing a tick-tack-toe game is a classic programming exercise. It's trivial to test if the sum of three numbers is 15. Please use this point to implement a simplified tick-tack-toe program that never loses a game¹.

The PDF book can be downloaded from <https://github.com/liuxinyu95/unplugged>. Please contact me through liuxinyu95@gmail.com if you want a hard copy.

¹The answer to all the exercises in this book can be found in the appendix.

Contents

1	Numbers	1
1.1	The history of number	1
1.2	Peano Axioms	2
1.3	Natural numbers and programming	4
1.4	Structure of natural numbers	6
1.5	Isomorphism of natural numbers	10
1.6	Form and structure	14
2	Recursion	17
2.1	Everything is number	17
2.2	The Euclidean algorithm	20
2.2.1	The Euclid's Elements	20
2.2.2	Euclidean algorithm	21
2.2.3	Extended Euclidean algorithm	23
2.2.4	Influence of Euclidean algorithm	27
2.3	The λ calculus	30
2.3.1	Expression reduction	31
2.3.2	λ abstraction	32
2.3.3	λ conversion rules	33
2.4	Definition of recursion	36
2.4.1	Y combinator	37
2.5	The impact of λ calculus	37
2.6	More recursive structures	39
2.7	The recursive pattern and structure	40
2.8	Further Reading	42
2.9	Appendix: Example program for 2 water jars puzzle	43
3	Symmetry	45
3.1	Group	48
3.1.1	Group	53
3.1.2	Monoid and semi-group	55
3.1.3	Properties of group	58
3.1.4	Permutation group	63
3.1.5	Groups and symmetry	65
3.1.6	Cyclic group	67
3.1.7	Subgroup	69
3.1.8	Lagrange's theorem	74
3.2	Ring and Field	86
3.2.1	Ring	88
3.2.2	Division ring and field	90

3.3	Galois theory	91
3.3.1	Field extension	92
3.3.2	Automorphism and Galois group	93
3.3.3	Fundamental theorem of Galois theory	95
3.3.4	Solvability	97
3.4	Further reading	101
4	Category	103
4.1	Category	105
4.1.1	Examples of categories	108
4.1.2	Arrow \neq function	112
4.2	Functors	112
4.2.1	Definition of functor	113
4.2.2	Functor examples	113
4.3	Products and coproducts	118
4.3.1	Definition of product and coproduct	121
4.3.2	The properties of product and coproduct	123
4.3.3	Functors from products and coproducts	125
4.4	Natural transformation	128
4.4.1	Examples of natural transformation	129
4.4.2	Natural isomorphism	131
4.5	Data types	132
4.5.1	Initial object and terminal object	132
4.5.2	Exponentials	137
4.5.3	Cartesian closed and object arithmetic	141
	0th Power	141
	Powers of 1	142
	First power	142
	Exponentials of sums	142
	Exponentials of exponentials	143
	Exponentials over products	143
4.5.4	Polynomial functors	143
4.5.5	F-algebra	144
	Recursion and fixed point	148
	Initial algebra and catamorphism	149
	Algebraic data type	153
4.6	Summary	156
4.7	Further reading	158
4.8	Appendix - example programs	158
5	Fusion	161
5.1	foldr/build fusion	162
5.1.1	Folding from right for list	163
5.1.2	foldr/build fusion law	164
5.1.3	build forms for list	165
5.1.4	Reduction with the fusion law	166
5.1.5	Type constraint	168
5.1.6	Fusion law in category theory	168
5.2	Make a Century	171
5.2.1	Exhaustive search	171
5.2.2	Improvement	173
5.3	Further reading	175

5.4	Appendix - example source code	175
6	Infinity	177
6.1	The infinity concept	179
6.1.1	Potential infinity and actual infinity	182
6.1.2	Method of exhaustion and calculus	184
6.2	Potential infinity and programming	188
6.2.1	Coalgebra and infinite stream★	190
6.3	Explore the actual infinity	193
6.3.1	Paradise of infinite kingdom	194
6.3.2	One-to-one correspondence and infinite set	197
	Cantor and Dedekind	198
	Fibonacci numbers and Hamming numbers	201
6.3.3	Countable and uncountable infinity	203
6.3.4	Dedekind cut	206
6.3.5	Transfinite numbers and continuum hypothesis	208
	Transfinite numbers	208
	Continuum hypothesis	211
6.4	Infinity in art and music	212
6.5	Appendix - Example programs	217
6.6	Appendix - Proof of Cantor's theorem	218
6.7	Appendix - Canon per tonos, The Music Offering by J.S. Bach	219
7	Paradox	221
7.1	Boundary of computation	224
7.2	Russel's paradox	225
7.2.1	Impact of Russell's paradox	228
7.3	Philosophy of mathematics	228
7.3.1	Logicism	228
7.3.2	Intuitionism	230
7.3.3	Formalism	231
7.3.4	Axiomatic set theory	233
7.4	Gödel's incompleteness theorems	236
7.5	Proof sketch for the first incompleteness theorem	238
7.5.1	Formalize the system	238
	Axioms and reasoning rules	239
	Incompleteness of TNT	240
7.5.2	Gödel numbering	241
7.5.3	Construct the self-reference	242
7.6	Universal program and diagonal argument	243
7.7	Epilogue	244
Appendices		
Answers		247
.1	Preface	247
.2	Natural Numbers	249
.3	Recursion	256
.4	Symmetry	262
.5	categories	274
.6	Fusion	284
.7	Infinity	288

.8 Paradox	291
Proof of commutative law of addition	293
Uniqueness of product and coproduct	295
Cartesian product and disjoint union of sets form product and coproduct	297
Reference	299
GNU Free Documentation License	307
1. APPLICABILITY AND DEFINITIONS	307
2. VERBATIM COPYING	308
3. COPYING IN QUANTITY	309
4. MODIFICATIONS	309
5. COMBINING DOCUMENTS	311
6. COLLECTIONS OF DOCUMENTS	311
7. AGGREGATION WITH INDEPENDENT WORKS	311
8. TRANSLATION	312
9. TERMINATION	312
10. FUTURE REVISIONS OF THIS LICENSE	312
11. RELICENSING	313
ADDENDUM: How to use this License for your documents	313

Chapter 1

Numbers

Numbers are the highest degree of knowledge. It is knowledge itself.

—Plato

1.1 The history of number

The number emerged with human evolution. Some people believe that language was inspired by numbers. Our ancestors learned the numbers from the gathering and hunting activities. People needed to count the gathered fruits. As trading developed, people needed numeral tools to handle bigger numbers than previously encountered.

We found in the regions of Iran, people made clay tokens for record keeping around 4000 BC. They created two round tokens with '+' sign baked to represent "two sheep". Each token represented a sheep. Representing a hundred sheep with many tokens would be impractical, so they invented different clay tokens to represent ten sheep, twenty sheep and so on. In order to avoid the record being altered, people invented a clay envelope in which tokens were placed, sealed, and baked. If anybody disputed the number, they could break open the envelope and do a recount. They also pressed the signs outside the envelope before it was baked, these signs on the outside became the first written language for numbers[2]. Figure 1.1 shows the ancient clay tokens and envelopes found in Uruk period.

As the number increasing, the clay tokens and envelopes were gradually replaced by more powerful numerals. About 3500 BC, the Sumerians in Mesopotamia used round stylus in flat clay tablets to carve pictographs representing various tokens. Each sign represented both commodity being counted and the quantity of that commodity.

The next big step happened around 3100 BC. The abstract numbers dissociated from the thing being counted. We found from the clay tablets, the things being counted were indicated by pictographs carved with a sharp stylus next to round-stylus numerals. These abstracted numerals later evolved to Babylonian cuneiform characters.

The abstract number emerged from intelligent mind. People realized the abstract

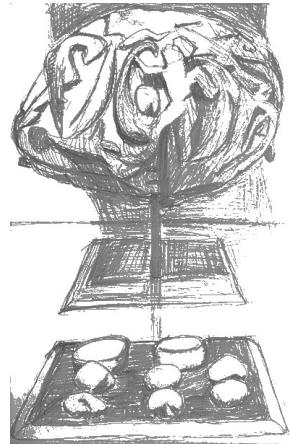
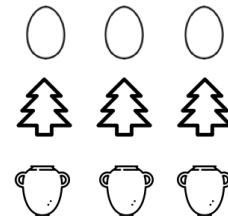


Figure 1.1: The envelop of tokens in Uruk period from Susa. Louvre Museum

1	Y	11	YY	21	YY	31	YY	41	YY	51	YY
2	YY	12	YY	22	YY	32	YY	42	YY	52	YY
3	YY	13	YY	23	YY	33	YY	43	YY	53	YY
4	YY	14	YY	24	YY	34	YY	44	YY	54	YY
5	YY	15	YY	25	YY	35	YY	45	YY	55	YY
6	YY	16	YY	26	YY	36	YY	46	YY	56	YY
7	YY	17	YY	27	YY	37	YY	47	YY	57	YY
8	YY	18	YY	28	YY	38	YY	48	YY	58	YY
9	YY	19	YY	29	YY	39	YY	49	YY	59	YY
10	Y	20	YY	30	YY	40	YY	50	YY		

(a) Babylonian numerals[3]

3



(b) The abstract three

three could represent three eggs, three trees, and three jars. It's a powerful tool. People can manipulate the pure numbers and apply the result to the concrete things. When increase the abstract three by one to get four, we know gathering another egg after three eggs gives four eggs; we also know baking another jar after three jars gives four jars. We resolve a whole kind of problems instead of one by one.

Starting from the numbers, people developed add, subtraction, then the more powerful methods of multiplication and division. When measure the length, angles, areas, and volumes, we connected the number to the geometry quantity. People from different places found the inner relationships and laws for the numbers and shapes. Ancient Egyptian, Greece, and Chinese people found the Pythagoras theorem independently, and applied it to the amazing works like to build the great pyramid. Trace back from the modern civilization, we find the natural number is the source of math and science. German mathematician Kronecker said 'God made the integers; all else is the work of man.'¹

1.2 Peano Axioms

Euclid's *Element* is the first work introduced the axiomatic methods. From five axioms and postulates, Euclid developed the laws one by one elaborately. Every result is only based on the axioms and the theorems proved before. With this approach, he built the great building of geometry. However, there was no axiomatic formal system for natural numbers for long time. People considered natural numbers were straightforward and the related facts were obvious. The axioms of natural number was setup by Italian mathematician Peano till 1889. known as Peano Axioms nowadays. It's interesting that there are also five axioms.

1. 0 is a natural number. Expressed as $\exists 0 \in N$;
2. For every natural number, there is a successor natural number. Expressed as $\forall n \in N, \exists n' = \text{succ}(n) \in N$;

It seems that we can define the infinite natural numbers only with these two axioms. From 0, the next is 1, then the next is 2, then 3, ..., then n , and then $n + 1$, ... However, there is a counter example. Consider a set with only two numbers $\{0, 1\}$. Where the successor of 1 is defined as 0, while the successor of 0 is defined as 1. It satisfies the above two axioms well although they are not as we expected. In order to avoid this situation, we need the third Peano axiom.

3. 0 isn't the successor of any natural number. Expressed as $\forall n \in N : n' \neq 0$;

¹Natural number is different from integer. We'll come back to the story of Kronecker in the chapter of infinity.

Are these three axioms enough? We can still find another counter example. Consider the set of $\{0, 1, 2\}$. Define the successor of 0 is 1, the successor of 1 is 2, and the successor of 2 is 2 again. It satisfies all the three axioms so far. We therefore need the fourth Peano axiom.

4. Different natural numbers have different successors. In other words, if two natural numbers have the same successor, then they are same. It can be formally expressed as $\forall n, m \in N : n' = m' \Rightarrow n = m$;

However, it is still not enough. We can still find another example. For set $\{0, 0.5, 1, 1.5, 2, 2.5, \dots\}$. Define 1 is the successor of 0, 2 is the successor of 1, ...; 1.5 is the successor of 0.5, 2.5 is the successor of 1.5, ...; But 0.5 is not the successor of any other numbers. In order to exclude such ‘unreachable’ elements, we need the last Peano axiom.

5. If some subset of natural numbers contains 0, and every element in it has a successor, then this subset is same as the whole natural numbers. It can be expressed as $\forall S \subset N : (0 \in S \wedge \forall n \in S \Rightarrow n' \in S) \Rightarrow S = N$.

Why does the fifth axiom exclude the above counter example? For $\{0, 0.5, 1, 1.5, 2, 2.5, \dots\}$, consider the subset of $\{0, 1, 2, \dots\}$. 0 belongs to it, and every element has a successor. But it is not identical to the original set. As 1.5, 2.5, ... are not in this subset, it does not satisfy the fifth Peano axiom. This last axiom as also known as ‘Axiom of induction’. It can be equally stated as the following.

5. For any proposition of natural numbers, if it holds for 0, and when assume it holds for some number n , we can prove it also holds for n' , then the proposition holds for all natural numbers. (This axiom ensure the correctness of mathematical induction.)

This the complete statement of the five Peano axioms. They can build the first-order arithmetic, also known as Peano arithmetic².



Figure 1.3: Giuseppe Peano (1858-1932)

Giuseppe Peano was an Italian mathematician, logician, and linguist. Peano was born and raised on a farm at Spinetta, a hamlet now belonging to Cuneo, Italy. He enrolled at the University of Turin in 1876, graduating in 1880 with high honors, after which the University employed him to teach calculus course. In 1887, Peano married Carola Crosio. In 1886, he began teaching concurrently at the Royal Military Academy. From

²Some people use 1, but not 0 as the first natural number. The order is different from the original works published by Peano, where the fifth axiom of induction was list as the third one.

1880s, Peano started to study mathematical logic. He published the Peano axioms, a formal foundation for the natural numbers. Peano started the Formulario Project. It was to be an “Encyclopedia of Mathematics”, containing all known formulae and theorems of mathematical science. In 1900, the Second International Congress of Mathematicians was held in Paris. At the conference Peano met Bertrand Russell and gave him a copy of Formulario. Russell was so struck by Peano’s innovative logical symbols that he left the conference and returned home to study Peano’s text[4].

When Russell and Whitehead wrote *Principia Mathematica*, they were deeply influenced by Peano. Peano played a key role in the axiomatization of mathematics and was a leading pioneer in the development of mathematical logic and set theory. As part of this effort, he made key contributions to the modern rigorous and systematic treatment of the method of mathematical induction. He spent most of his career teaching mathematics at the University of Turin. He also wrote an international auxiliary language, Latino sine flexione (“Latin without inflections”, later called Interlingua), which is a simplified version of Classical Latin. Most of his books and papers are in Latino sine flexione. Although Peano put a lot of effort to rewrite his works in the new language, few people read it. On the other hand, his early works in French influenced many mathematicians, especially to the Bourbaki group, which came out many top mathematicians like André Weil, Jean Dieudonné, Henri Cartan, Schwartz, Serre, Grothendieck and so on.

Giuseppe Peano died on April, 20th, 1932 when he suffered a fatal heart attack.

1.3 Natural numbers and programming

People make amazing achievement with the modern computer systems. We didn’t establish the axioms of computer programming before developing these results. After the great success of computer application, then the foundation of computer science is gradually developed to be strict, formal, and mathematized. The similar thing happens several times in our history. Calculus was developed by Newton and Leibniz independently in the 17th century, then applied to a wide range of areas, including fluid dynamics, astronomy and so on. However, it was not formalized until Weierstrass and Cauchy developed the foundation in the 19th century[4].

We’ll emulate it. From the Peano axioms, define the natural numbers with computer programs. In a computer system without familiar numbers like 0, 1, 2, ..., we can define the natural numbers as below³:

```
data Nat = zero | succ Nat
```

A natural number is either zero, or the successor of another natural number. Symbol ‘|’ is mutual exclusive, it implicates the axiom that zero is not the successor of any natural number. We can further define the addition for natural numbers.

```
a + zero = a
a + (succ b) = succ (a + b)
```

There are two rules for addition. First, any natural number adds zero gives that number itself; second, a natural number adds to a successor of some natural number equals to the successor of the sum of the two. In mathematic expression:

$$\begin{aligned} a + 0 &= a \\ a + b' &= (a + b)' \end{aligned} \tag{1.1}$$

Let’s use 2+3 as the example. natural number 2 is succ(succ zero), and 3 is succ(succ(succ zero)). According to the definition of addition:

³We use a virtual, ideal programming language in this book. Some real programs are given at the end of each chapter for reference.

```

succ(succ zero) + succ(succ(succ zero))
= succ(succ(succ zero) + succ(succ zero))
= succ(succ(succ(succ zero) + succ zero))
= succ(succ(succ(succ(succ zero) + zero))))
= succ(succ(succ(succ(succ zero))))

```

The result is the 5th successor of zero. It's not practical to apply succeed function again and again for big numbers like 100. Let's introduce a simplified notation for natural number n .

$$n = \text{foldn}(\text{zero}, \text{succ}, n) \quad (1.2)$$

It applies *succ* function to zero n times. Function *foldn* can be realized as the following.

$$\begin{aligned} \text{foldn}(z, f, 0) &= z \\ \text{foldn}(z, f, n') &= f(\text{foldn}(z, f, n)) \end{aligned} \quad (1.3)$$

Function *foldn* defines some operation on natural number. When z is *zero*, and f is the *succ* function, then it can apply the succeed operation multiple times to get a specific natural number. We can verify it with the first several numbers.

```

foldn(zero, succ, 0) = zero
foldn(zero, succ, 1) = succ(foldn(zero, succ, 0)) = succ zero
foldn(zero, succ, 2) = succ(foldn(zero, succ, 1)) = succ(succ zero)
...

```

Multiplication for natural number can be defined on top of addition.

```

a . zero = zero
a . (succ b) = a . b + a

```

The multiplication can be expressed in mathematic symbols as below.

$$\begin{aligned} a \cdot 0 &= 0 \\ a \cdot b' &= a \cdot b + a \end{aligned} \quad (1.4)$$

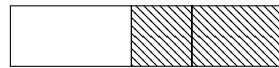
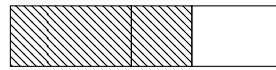


Figure 1.4: Association of addition in geometry. The areas of the above and bottom are same.

It turns out that the associative and commutative laws for addition and multiplication are neither axioms nor postulations. They all can be proved by Peano axioms and the definitions. Let's prove the associative law for addition as an example. This law states $(a + b) + c = a + (b + c)$. We firstly prove it holds when $c = 0$. According to the first rule in the add definition:

$$\begin{aligned} (a + b) + 0 &= a + b \\ &= a + (b + 0) \end{aligned}$$

Then for induction, assume $(a + b) + c = a + (b + c)$ hold, we want to prove $(a + b) + c' = a + (b + c')$.

$$\begin{aligned}
 (a + b) + c' &= (a + b + c)' && \text{2nd equation defining } +, \text{ (backward)} \\
 &= (a + (b + c))' && \text{induction assumption} \\
 &= a + (b + c)' && \text{2nd equation defining } + \\
 &= a + (b + c') && \text{2nd equation defining } +, \text{ (backward)}
 \end{aligned}$$

Hence proves the associative law for addition. However, it is a bit complex to prove the commutative law. We give it in the Appendix of the book.

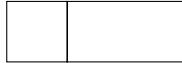


Figure 1.5: Commutative law of addition in geometry. Turn upside down or mirror the upper figure.

Exercise 1.2

1. Define 1 as the successor of 0, prove $a \cdot 1 = a$ holds for all natural numbers;
2. Prove the distributive law for multiplication;
3. Prove the associative and commutative laws for multiplication.
4. How to verify $3 + 147 = 150$ with Peano axioms?
5. Give the geometric explanation for distributive, associative, and commutative laws of multiplication.

1.4 Structure of natural numbers

We can define more complex operations on top of addition and multiplication. One example is summation: $0 + 1 + 2 + \dots$

$$\begin{aligned}
 \text{sum}(0) &= 0 \\
 \text{sum}(n + 1) &= (n + 1) + \text{sum}(n)
 \end{aligned} \tag{1.5}$$

Another example is the factorial $n!$

$$\begin{aligned}
 \text{fact}(0) &= 1 \\
 \text{fact}(n + 1) &= (n + 1) \cdot \text{fact}(n)
 \end{aligned} \tag{1.6}$$

They are similar to each other. Although artificial intelligence achieves incredible result today, the machine can't jump out of the system, as intelligent mind, to abstract in a higher level. This is one of the most complex, powerful, and mysterious part in human brain[5].

Corresponding to zero in natural number, both summation and factorial have a start value. Summation starts from zero, factorial starts from one. For recursion, they both apply some operation to a number and its successor. For summation, it's $n' + \text{sum}(n)$,

for factorial, it's $n' \cdot \text{fact}(n)$. If we abstract the start value as c , the recursive operation as h , then we can use the same form for both.

$$\begin{aligned} f(0) &= c \\ f(n+1) &= h(f(n)) \end{aligned} \tag{1.7}$$

This scheme is called as *structural* recursion over the natural numbers. Below examples show how it behaves over the first several numbers.

n	$f(n)$
0	c
1	$f(1) = h(f(0)) = h(c)$
2	$f(2) = h(f(1)) = h(h(c))$
3	$f(3) = h(f(2)) = h(h(h(c)))$
...	...
n	$f(n) = h^n(c)$

Where $h^n(c)$ means applying operation h over c for n times. It's an instance of the more general *primitive* recursion([6], p5). Further, we can find it is related to the *foldn* defined in (1.3).

$$f = \text{foldn}(c, h) \tag{1.8}$$

There are three variables in the original *foldn* definition, why are there only two appeared? We can actually write it as $f(n) = \text{foldn}(c, h, n)$. When we bind the first two variables to *foldn*, it turns to be a new function accepts one argument. We can consider it as $\text{foldn}(c, h)(n)$.

We call *foldn* the *fold* operation on natural numbers. When c is *zero* and h is *succ*, we get a sequence of natural numbers:

$$\text{zero}, \text{succ}(\text{zero}), \text{succ}(\text{succ}(\text{zero})), \dots \text{succ}^n(\text{zero}), \dots$$

When c and h are other things than *zero* or *succ*, then $\text{foldn}(c, h)$ describes some isomorphism⁴ to natural numbers. Here are some examples.

$$(+m) = \text{foldn}(m, \text{succ})$$

This is the operation to increase a number by m . When applying to the natural numbers, it generates an isomorphic sequence of $m, m+1, m+2, \dots, n+m, \dots$

$$(\cdot m) = \text{foldn}(0, (+m))$$

This is the operation to multiply a number by m . When applying to the natural numbers, it generates an isomorphic sequence of $0, m, 2m, 3m, \dots, nm, \dots$

$$m^{\wedge} = \text{foldn}(1, (\cdot m))$$

This is the operation to take the power for a number m . When applying to natural numbers, it generates an isomorphic sequence of $1, m, m^2, m^3, \dots, m^n, \dots$

Can we use the abstract tool *foldn* to define the summation and factorial? Observe the below table.

⁴The formal definition of isomorphism will be given in later chapter. Different from the mathematic definition, here it means the similarity of the form.

n	0	1	2	3	...	n'
$sum(n)$	0	$1 + 0 = 1$	$2 + 1 = 3$	$3 + 3 = 6$...	$n' + sum(n)$
$n!$	1	$1 \times 1 = 1$	$2 \times 1 = 2$	$3 \times 2 = 6$...	$n' \cdot (n!)$

We know that h need to be a binary operation as it manipulates n' and $f(n)$. To solve it, we define c as a pair (a, b) ⁵. Then define some kind of ‘succ’ operation on the pair. We also need functions to extract a and b from the pair.

$$\begin{aligned} 1st(a, b) &= a \\ 2nd(a, b) &= b \end{aligned} \tag{1.9}$$

With these setup, we can define summation:

$$\begin{aligned} c &= (0, 0) && \text{Starting pair} \\ h(m, n) &= (m', m' + n) && \text{Succeed the 1st; Add the successor and the 2nd} \\ sum &= 2nd \cdot foldn(c, h) \end{aligned}$$

Starting from $(0, 0)$, below table gives the steps for summation.

(a, b)	$(a', b') = h(a, b)$	b'
$(0, 0)$	$(0 + 1 = 1, 1 + 0 = 1) = (1, 1)$	1
$(1, 1)$	$(1 + 1 = 2, 2 + 1 = 3) = (2, 3)$	3
$(2, 3)$	$(2 + 1 = 3, 3 + 3 = 6) = (3, 6)$	6
...
$(m, sum(m))$	$(m + 1, m + 1 + sum(m))$	$sum(m + 1)$

Similarly, we can define factorial with $foldn$.

$$\begin{aligned} c &= (0, 1) && \text{Starting pair for factorial} \\ h(m, n) &= (m', m'n) && \text{Iteration for factorial} \\ fact &= 2nd \cdot foldn(c, h) \end{aligned}$$

Here we use the symbol ‘ \cdot ’ to ‘connect’ the $2nd$ function and the $foldn(c, h)$ function. We call it *function composition*. $f \cdot g$ means firstly apply g to the variable, then apply f on top of the result. That is $(f \cdot g)(x) = f(g(x))$.

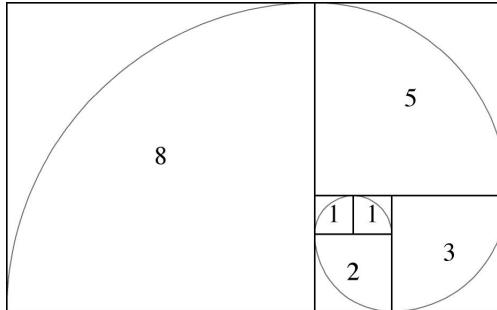
Let’s see another example powered by this abstract tool, Fibonacci sequence. It’s named after the medieval mathematician Leonardo Pisano. Fibonacci originates from ‘*filius Bonacci*’ in Latin. It means the son of (the) Bonacci. Fibonacci’s father was a wealthy Italian merchant often did trading around North Africa and the Mediterranean coast. Fibonacci traveled with him as a young boy. It was in Bugia (now Béjaïa, Algeria) that he learned about the Hindu–Arabic numeral system. Fibonacci realized many advantage of this numeral system. He introduced it to Europe through his book, the *Liber Abaci* (Book of Abacus or Book of Calculation, 1202). European people were using Roman numeral system before that. We can still see Roman numbers in clock plat today. The Roman number for year 2018 is MMXVIII. Where M stands for 1000, so two M letters mean 2000; X represents 10, V stands for 5, the three I mean 3. Sum them up, we get 2018. The Hindu–Arabic numeral system introduced by Fibonacci is a positional decimal numeral system. We are using it almost everywhere today. It uses the zero invented by Indian mathematicians. Numbers at different position mean different value. This advanced numeral system were widely used in business, for example converting different currencies, calculating profit and interest, which were important to the growing banking industry. It influenced the mathematics in Europe greatly.

⁵Also known as *tuple* in computer programs

Fibonacci numbers is well known as a problem described in the *Liber Abaci*, although it can be traced back to 200 BC in India. Assuming a newly born pair of rabbits, one male, one female, are put in a field; rabbits are able to mate at the age of one month so that at the end of its second month a female can produce another pair of rabbits; rabbits never die and a mating pair always produces one new pair (one male, one female) every month from the second month on. Then how many pairs will there be in one year?

When start, there is a pair in the first month. In the second month, there is a new born pair. In total there are two pairs. In the third month, the matured pair produces another pair, while the new born in the previous month are still young. In total, there are $2 + 1 = 3$ pairs. In the fourth month, the two pairs of matured rabbits produce another two pairs of baby. Plus the three pairs in the third month, there are total $3 + 2 = 5$ pairs. Repeating it gives a sequence of numbers.

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...



(a) The length of the squares give a Fibonacci sequence



(b) Leonardo Pisano, Fibonacci (1175-1250)

It's easy to find the pattern of this sequence. From the third number, every number is the sum of the previous two. We can understand the reason behind it like this. Let there be m pairs of rabbits in the previous month, and n pairs in this month. As the new additional $n - m$ pairs are all new born, the rest m pairs are mature. In the next month, the $n - m$ pairs grow mature; while the m pairs of big rabbits produce another m pairs of baby rabbits. The total pairs in the next month is the sum of big and baby rabbits, which is $(n - m) + m + m = n + m$. With this deduction, we can give the recursive definition of Fibonacci numbers.

$$\begin{aligned} F_0 &= 0 \\ F_1 &= 1 \\ F_{n+2} &= F_n + F_{n+1} \end{aligned} \tag{1.10}$$

The starting numbers are defined as 0 and 1 by convention⁶. As Fibonacci numbers start from a *pair* of natural numbers, and the recursive relation also uses a pair of elements, we can use our abstract tool *foldn* to define Fibonacci sequence⁷.

$$\begin{aligned} F &= 1st \cdot foldn((0, 1), h) \\ h(m, n) &= (n, m + n) \end{aligned} \tag{1.11}$$

Can this definition be realized in the computer programs in real world? Is it too

⁶If start from 1 and 3, it produces the Lucas sequence 1, 3, 4, 7, 11, 18, ...

⁷We'll give another different definition of Fibonacci numbers in the chapter about infinity.

idealistic? Below is a real piece of Haskell program implements Fibonacci numbers⁸. Run command `fib 10` outputs the 10th Fibonacci number, 55⁹.

```
foldn z _ 0 = z
foldn z f (n + 1) = f (foldn z f n)

fib = fst . foldn (0, 1) h where
  h (m, n) = (n, m + n)
```

Exercise 1.3

1. Define square for natural number $()^2$ with `foldn`;
2. Define $()^m$ with `foldn`, which gives the m -power of a natural number;
3. Define sum of odd numbers with `foldn`, what sequence does it produce?
4. There is a line of holes (infinite many) in the forest. A fox hides in a hole. It moves to the next hole every day. If we can only check one hole a day, is there a way to catch the fox? Prove this method works. What if the fox moves more than one hole a day[7]?

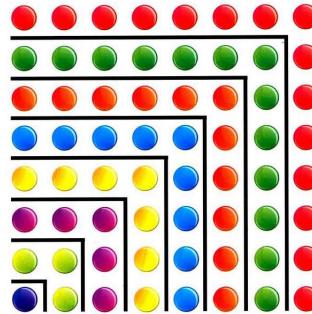


Figure 1.7: Cover of PWW (Proof Without Words), part

1.5 Isomorphism of natural numbers

We've seen the examples that natural numbers can be isomorphic to its subsets, like odd and even numbers, squares, and Fibonacci numbers. Natural numbers can also be isomorphic to other things. One interesting example is the list in the computer programs. Here is the definition of list.

```
data List A = nil | cons(A, List A)
```

As a data structure, a list of type A is either empty, represented as `nil`; or contains two parts: one node with data of type A, and the rest sub-list. Function `cons` links an element of type A to another list¹⁰. Figure 1.8 shows a list of 6 nodes.

⁸After 2010, the $n+k$ pattern matching is not supported any more in Haskell. We can modify it as:
`foldn z f n = f (foldn z f (n-1))`

⁹One line code to produce the first 100 Fibonacci numbers:

```
take 100 $ map fst $ iterate (λ(m, n) -> (n, m + n)) (0, 1)
```

¹⁰The name `cons` comes from the Lisp naming convention.

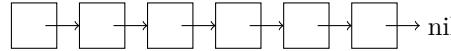


Figure 1.8: Linked-list

Because every node links to the next one or nil, list is also called as ‘linked-list’. In the tradition of computer programs, linked-list is often defined through the record data structure¹¹, for example:

```

Node of A:
  key: A
  next: Node of A
  
```

We can also understand the list as isomorphism of natural numbers. According to the first Peano axiom, nil is corresponding to zero; Based on the second Peano axiom, for any list, we can apply **cons**, to link a new element of type A to the left. We can treat **cons** corresponding to **succ** to the natural numbers. There are two different things. First, list is augmented with elements of type A. List **cons**(1, **cons**(2, **cons**(3, nil))) and **cons**(2, **cons**(1, **cons**(3, nil))); List **cons**(1, **cons**(4, **cons**(9, nil))) and **cons**('a', **cons**('b', **cons**('c', nil))) are all different lists. Second, new element is not added to the right at tail, but is added to the left on head. Different from the intuition, the list grows to the left but not to the right.

It’s not convenient to represent long list with nested **cons**. We simplify **cons**(1, **cons**(2, **cons**(3, nil))) to [1, 2, 3], and use symbol ‘:’ for **cons**. This list can also be written as 1:[2, 3] or 1:(2:(3:nil)). When type A is character, we use string in quote to represent this special type of list. For example, “hello” is the simplified form for [‘h’, ‘e’, ‘l’, ‘l’, ‘o’].

Similar to add defined for natural numbers, we can define the concatenation for lists as the following.

$$\begin{aligned} \text{nil} \# y &= y \\ \text{cons}(a, x) \# y &= \text{cons}(a, x \# y) \end{aligned} \tag{1.12}$$

There are two rules for list concatenation. First, empty list concatenates any list produces the same list; second, when concatenate the ‘successor’ of a list to another one, it equals to firstly concatenate the two lists, then take the successor. Compare to the add for natural numbers, the definition of list concatenation is mirrored symmetric.

$$\begin{array}{c} \text{nil} \# y = y \quad | \quad a + 0 = a \\ \text{cons}(a, x) \# y = \text{cons}(a, x \# y) \quad | \quad a + \text{succ}(b) = \text{succ}(a + b) \end{array}$$

Figure 1.9: The list concatenation and natural number adding are mirrored symmetric.

With the hint of symmetry, we can prove the associative law for list concatenation through the induction axiom. To prove $(x \# y) \# z = x \# (y \# z)$, we first prove it holds for $x = \text{nil}$.

$$\begin{aligned} (\text{nil} \# y) \# z &= y \# z && \text{1st equation of } \# \\ &= \text{nil} \# (y \# z) && \text{1st equation of } \# \text{ (backward)} \end{aligned}$$

For the induction case, assume $(x \# y) \# z = x \# (y \# z)$ holds. We want to prove that $((a : x) \# y) \# z = (a : x) \# (y \# z)$.

¹¹In most cases, the data stored in list have the same type. However, there is also heterogeneous list, like the list in Lisp for example.

$$\begin{aligned}
 ((a : x) \# y) \# z &= (a : (x + y)) \# z && \text{2nd equation of } \# \\
 &= a : ((x + y) \# z) && \text{2nd equation of } \# \\
 &= a : (x \# (y + z)) && \text{induction assumption} \\
 &= (a : x) \# (y \# z) && \text{2nd equation of } \# \text{ (backward)}
 \end{aligned}$$

With this, we proved the list concatenation is associative. Different from the natural numbers however, list concatenation is not commutative¹². For example $[2, 3, 5] \# [7, 11] = [2, 3, 5, 7, 11]$, but when change the order, the result is $[7, 11] \# [2, 3, 5] = [7, 11, 2, 3, 5]$.

Consider the similarity to the natural numbers, we can also define the abstract folding operation for lists. Corresponding to the abstract start value c and the abstract binary operation h , we define the recursive scheme as below.

$$\begin{aligned}
 f(nil) &= c \\
 f(cons(a, x)) &= h(a, f(x))
 \end{aligned} \tag{1.13}$$

As the next step, let $f = foldr(c, h)$, then we can abstract the list folding. We name it as *foldr* to call out the folding starts from right to left.

$$\begin{aligned}
 foldr(c, h, nil) &= c \\
 foldr(c, h, cons(a, x)) &= h(a, foldr(c, h, x))
 \end{aligned} \tag{1.14}$$

We can define varies of list manipulations with *foldr*. The followings are to sum and multiply all the elements in list.

$$\begin{aligned}
 sum &= foldr(0, +) \\
 product &= foldr(1, \times)
 \end{aligned} \tag{1.15}$$

We can understand how *sum* behaves with examples. First is about empty list, $sum([]) = foldr(0, +, nil) = 0$. Then the list with some elements:

$$\begin{aligned}
 sum([1, 3, 5, 7]) &= foldr(0, +, 1 : [3, 5, 7]) \\
 &= 1 + foldr(0, +, 3 : [5, 7]) \\
 &= 1 + (3 + foldr(0, +, 5 : [7])) \\
 &= 1 + (3 + (5 + foldr(0, +, cons(7, nil)))) \\
 &= 1 + (3 + (5 + (7 + foldr(0, +, nil)))) \\
 &= 1 + (3 + (5 + (7 + 0))) \\
 &= 16
 \end{aligned}$$

We can measure the length of the list with *sum*. It essentially maps a list to a natural number.

$$\begin{aligned}
 one(a) &= 1 \\
 length &= sum \cdot foldr(0, one)
 \end{aligned} \tag{1.16}$$

Where function *one* is called as *constant* function. It always returns 1 for whatever variables. We can use $|x| = length(x)$ to represent the length of a given list. The next example shows we can use *foldr* to define list concatenation.

$$(+y) = foldr(y, cons) \tag{1.17}$$

It is corresponding to the $(+m)$ operation for natural number. Further, similar to the multiplication of natural numbers, we can define the ‘multiplication’ for lists, concatenate all sub-lists in a list.

$$concat = foldr(nil, +) \tag{1.18}$$

¹²This is the reason why we avoid using symbol $+$ for concatenation. But many programming languages use the $+$ sign. It causes potential issues in practice.

When apply $concat([[1, 1], [2, 3, 5], [8]])$, the result is $[1, 1, 2, 3, 5, 8]$. At the end of this section, we'll define two important list operations with $foldr$, filtering and mapping¹³. Filter is to compose a new list from the elements that satisfy a given predication. In order to realize filtering, we need introduce the conditional expression¹⁴. It's written as $(p \mapsto f, g)$. When give variable x , if the predication $p(x)$ holds, then the result is $f(x)$, else it's $g(x)$. We also use if $p(x)$ then $f(x)$ else $g(x)$ for conditional expression.

$$filter(p) = foldr(nil, (p \cdot 1st \mapsto cons, 2nd)) \quad (1.19)$$

Let's use an example to understand how this definition works. We want to select all even numbers from a list $filter(even, [1, 4, 9, 16, 25])$. Like expansion process in sum , the filtering expands to $h(1, h(4, h(9, \dots)))$ till the right end $cons(25, nil)$. According to the definition of $foldr$, the result is c when the list is nil . So the next step is to compute $h(25, nil)$, where h is the conditional expression. When apply $even \cdot 1st$ to the pair $(25, nil)$, function $1st$ picks 25, as it's odd, the predication $even$ does not hold. Based on the conditional expression, $2nd$ is evaluated and gives the result nil . Then we enter the upper level to compute $h(16, nil)$. Function $1st$ extracts the number 16, as 16 is even, the predicate $even$ holds, so the conditional expression sends to $cons(16, nil)$, which produces the list $[16]$. Then we enter one more upper level to compute $h(9, [16])$, the conditional expression sends to $2nd$, which again produces $[16]$. The computation enters to $h(4, [16])$ next. The conditional expression sends to $cons(4, [16])$, which produces the list $[4, 16]$. The computation finally reach to the top level $h(1, [4, 16])$. The conditional expression sends to $2nd$, which produces the final result $[4, 16]$.

The concept of mapping is to transform every element in one list to another value through a function f , and form a new list. That is $map(f, \{x_1, x_2, \dots, x_n\}) = \{f(x_1), f(x_2), \dots, f(x_n)\}$. It can be defined with $foldr$ as below.

$$\begin{aligned} map(f) &= foldr(nil, h) \\ h(x, c) &= cons(f(x), c) \end{aligned} \quad (1.20)$$

We call the operation that applies a function to the first value in a pair as '*first*', that is $first(f, (x, y)) = (f(x), y)$. We'll come back to it in the chapter of category theory. With *first*, *map* can be defined as $map(f) = foldr(nil, cons \cdot first(f))$.

Exercise 1.4

1. What does the expression $foldr(nil, cons)$ define?
2. Read in a sequence of digits (string of digit numbers), convert it to decimal with $foldr$. How to handle hexadecimal digit and number? How to handle the decimal point?
3. Jon Bentley gives the maximum sum of sub-vector puzzle in *Programming Pearls*. For integer list $\{x_1, x_2, \dots, x_n\}$, find the range i, j , that maximizes the sum of $x_i + x_{i+1} + \dots + x_j$. Solve it with $foldr$.
4. The longest sub-string without repeated characters. Given a string, find the longest sub-string without any repeated characters in it. For example, the answer for string "abcabcb" is "abc". Solve it with $foldr$.

¹³Different from one to one mapping, the map defined here is one direction only. For example, the map from a string to its length is one direction. The reverse map does not exist.

¹⁴Also known as McCarthy conditional form, or McCarthy formalism. It was introduced by the computer scientist, the inventor of Lisp, John McCarthy in 1960.

1.6 Form and structure



Figure 1.10: Raphael, School of Athens (part)

We illustrated several conclusions with geometry figures in this chapter, like the associative law, the commutative law, and the Fibonacci spiral. We want to express the beauty of isomorphism. Aristotle said “The chief forms of beauty are order and symmetry and definiteness, which the mathematical sciences demonstrate in a special degree.” (*Metaphysic*) Like geometry, with Peano’s work, natural numbers can also be built on top of the axioms. We use the similarity of natural numbers and lists to demonstrate the beauty of symmetry. When the Italian Renaissance artist Raphael created the world famous fresco *School of Athens*, he used the same approach of isomorphism by inventing a system of iconography. Many ancient Greece philosophers were illustrated with the figures in Raphael’s time of Renaissance. The center figures are Plato and his student Aristotle. Of which Plato is depicted in the image of Leonardo da Vinci; Aristotle is in the image of Giuliano da Sangallo. They were all great artists in Renaissance. The elder Plato is walking alongside Aristotle with his right hand figure point up, while Aristotle is stretching his hand forward. It is popularly thought that their gestures indicate central aspects of their philosophies, for Plato, his Theory of Forms, and for Aristotle, his empiricist views, with an emphasis on concrete particulars. Plato argues a sense of timelessness while Aristotle looks into the physicality of life and the present realm. Below the steps in the middle, the great philosopher Heraclitus leans the box and meditates. He is famous for the thoughts about simple dialectics and materialism. The image of Heraclitus is another great artist Michelangelo in Renaissance. The front left is centered on the great mathematician Pythagoras. He is writing something. On the right side of Pythagoras is a blond young man in a white cloak, considered to be Francisco Maria della Rovere. He was Ulbino’s future Grand Duke. The center of the bottom right is the great mathematician Euclid with a compass in hand (or Archimedes in other opinion), he is surrounded by Ptolemy, the great astronomer with the celestial sphere in hand. The opposite is the painter Raphael’s fellow villager, the architect Bramante. The one who wore a white hat on the right is the painter Sodom, the young man next to him with half head and a hat on his head, is the painter, Raphael himself. This reminds us the great musician Bach, who wrote his name B-A-C-H in his work ‘The Art of Fugue’ (Die Kunst der Fuge in German). School of Athens reflects the ancient Greece art and philosophy with the figures in the time of Renaissance. It is the multiple levels of isomorphism of

form and content, structure and thoughts. It is seen as “Raphael’s masterpiece and the perfect embodiment of the classical spirit of the Renaissance”.

Exercise 1.5

1. In the fold definition of Fibonacci numbers, the successor is computed as $(m', n') = (n, m + n)$. It is essentially matrix multiplication:

$$\begin{pmatrix} m' \\ n' \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} m \\ n \end{pmatrix}$$

Where it starts from $(0, 1)^T$. Then the Fibonacci numbers is isomorphic to natural numbers under the matrix multiplication:

$$\begin{pmatrix} F_n \\ F_{n+1} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Write a program to compute the power of 2-order square matrix, and use it to give the n -th Fibonacci number.

Chapter 2

Recursion

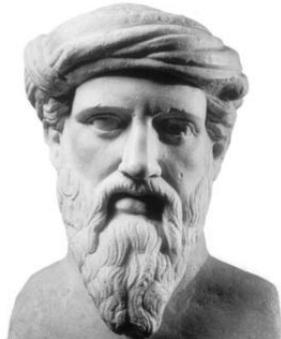
GNU means **GNU's Not Unix**

-Richard Stallman

People learn our world with numbers. In previous chapter, we introduced Peano axioms, and things that are isomorphic to natural numbers, like the list data structure in programming. Natural number is a fundamental tool, however, our building still need some corner stones. We accept the recursive definition without proof of its correctness, like the factorial for example.

$$\begin{aligned} \text{fact}(0) &= 1 \\ \text{fact}(n + 1) &= (n + 1)\text{fact}(n) \end{aligned}$$

Why does recursion work? What is the theory of recursion? Could we formally express recursion? We'll explore these questions in this chapter.



Pythagoras (about 570BC - 490BC)

2.1 Everything is number

Pythagoras was the first mathematician and philosopher who studied the universe with numbers. He is famous all over the world in the theory named after him. He was born in the island of Samos, off the coast of modern Turkey. Pythagoras might have learnt from Thales of Miletus. With Thales suggestion, he went to oriental to learn about mathematics. He spent 13 years (22 years in other sayings) in Egypt. After the Persian Empire conquered Egypt, Pythagoras went eastward to Babylon with the army. He learned mathematics and astronomy from the Babylonians. Pythagoras might also arrive in India. Wherever he went, Pythagoras learned from the local scholars to enrich his knowledge. He did not only study hard, but also thought deeply. After long time of study, Pythagoras formed his own thoughts[8].

Pythagoras returned his hometown after long journey abroad and began to give lectures. Around 520BC, he left Samos, possibly because he disagreed with the local tyranny. He arrived in the Greek colony of Croton (southern Italy today). At Croton, he won trust and admiration of people, and founded the philosophical school of Pythagoreanism. Many prominent members of his school were women. The school was devoted to study astronomy, geometry, number theory, and music. They are called quadrivium, affected more than 2000 years of European education[10]. Quadrivium reflects the Pythagoreans'

philosophy, that everything is number. The planetary motion corresponds to geometry, while geometry is built on top of numbers. Numbers are also connected with music. The so-called Pythagoreans, who were the first to take up mathematics, not only advanced this subject, but saturated with it, they fancied that the principles of mathematics were the principles of all things. said Aristotle in Metaphysics. Pythagoras is the first one discovered the pattern of octave in mathematics. Pythagoras was revered as the founder of mathematics and music¹.

The Pythagoreans believed all things were made of numbers. They studied the numbers and the their connection to nature. They developed the early number theory, one of the most important area in mathematics. The pythagreans classified the natural numbers, defined many important concepts including even and odd numbers, prime and composite numbers and so on. They found some numbers equal to the sum of their proper positive divisors², and named them as perfect numbers. For example $6 = 1 + 2 + 3$, while 1, 2, 3 are the all three divisors of 6. Pythagoreans found the first two perfect numbers³. The smallest one is 6, the next is 28 ($28 = 1 + 2 + 4 + 7 + 14$). The Pythagorean also found a class of figurate numbers⁴, when they formed geometry figure with stones.

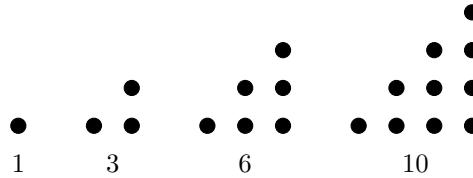


Figure 2.2: Triangular number

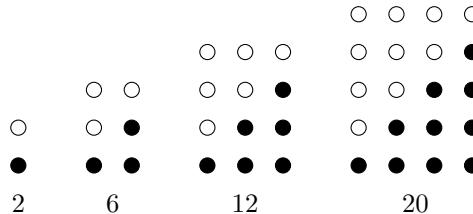


Figure 2.3: Oblong number (number of rectangle)

Figure 2.2 and 2.3 demonstrate the triangular numbers and oblong numbers (rectangle numbers). It's easy to figure out that the oblong number is two times of the corresponding triangle number. While the triangle number is the sum of the first n positive integers. By this way, the Pythagoreans found the formula to calculate the sum of postive integers.

$$1 + 2 + 3 + \dots + n = \frac{1}{2}n(n + 1)$$

¹There are different sayings about Pythagoras' death. His teachings of dedication and asceticism are credited with aiding in Croton's victory over the neighboring colony. After the victory, a democratic constitution was proposed, but the Pythagoreans rejected it. The supporters of democracy roused the populace against them. An attack was made upon them in some meeting-place. The building was set on fire, and many of the members perished; Different sources disagree regarding whether Pythagoras was killed, or if he managed to flee to Metapontum, where he lived out the rest of his life.

²The proer postive divisors are those positive divisors less than the number

³Also known as complete numbers or ideal numbers. Euclid proved a formation rule (Euclid's Element, Book IX, proposition 35) whereby $q(q+1)/2$ is an even perfect number whenever q is a prime of the form $2^p - 1$ for prime p —what is now called a Mersenne prime. Much later, Euler proved that all even perfect numbers are of this form. This is known as the Euclid–Euler theorem.

⁴The Pythagorean studied mathematics by making figures with small stones. The English word calculus comes from the Greek word stone[8].

The Pythagoreans also found the odd number could be represented in gnomon⁵ shape as shown in figure 2.4. And the first n gnomon shapes form a square, as in figure 2.5. By this way, they found the formula to calculate the sum of n -odd numbers.

$$1 + 3 + 5 + \dots + (2n - 1) = n^2$$

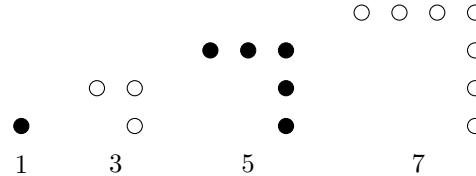


Figure 2.4: Gnomon number

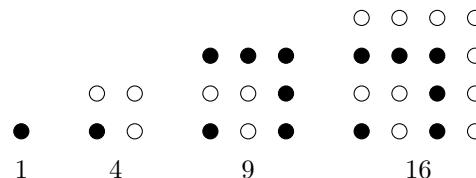


Figure 2.5: Square number and gnomon numbers

This is the answer to the exercise problem in chapter 1. With these facts, the Pythagoreans found there were many things could be explained in numbers. Given two strings under the same tension, it's said Pythagoras found the tune was harmonic if the ratio of their lengths is an integer. He developed the earliest music theory based on this. It seemed that music and mathematics were totally different things, while finally Pythagoras concluded that music was mathematics. Such unexpected relationship impacted Pythagoras greatly. He guessed that all things could be explained with integers or the ratio of integers. The Pythagoreans started to find more and more things connected to numbers, they believed the meaning of the whole universe was the harmonic of numbers, and developed the philosophy based on number. This led to the attempt to build the geometry also on top of the numbers, so the overall mathematics is based on integers.

The Pythagoreans' most famous achievement is the Pythagoras theorem. However, we'll see later, this theorem is a double-edged sword. It led to a recursive circle, and revealed the loophole of the idea that everything is number. In order to understand this, we need introduce the concept of commensurable and the Euclidean algorithm. To build the geometry on top of the numbers, The Pythagoreans defined how to measure a line segment with another, if segment A can be represented by duplicating segment V finite times, we say V measures A. It means the length of one segment is the integer times of the other. There can be varies of measurements for a given line. When one can use the same segment to measure different lines, it has to be the common measure. That is to say if and only if segment V can measure both A and B, it is the common measure of them. The Pythagoreans believed for any two segments, there must be a common measure. If this was true, then the whole geometry can be built on top of numbers.

⁵The word “gnomon” originally in Babylonia it probably meant and upright stick whose shadow was used to tell time. In Pythagoras’ time it meant a carpenter’s square. It also meant what was left over from a square when a smaller square was cut out of one corner. Later Euclid extended from square to parallelogram([9], p31).

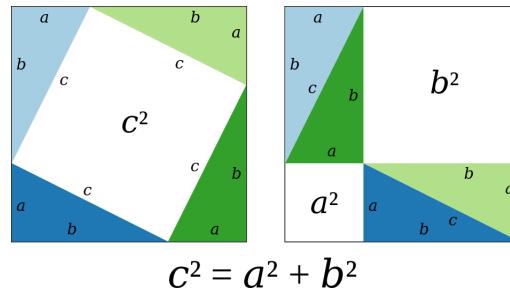


Figure 2.6: One of the methods to prove the Pythagoras theorem. The areas in white are same.

2.2 The Euclidean algorithm

As there can be multiple common measures, we define the biggest one as the greatest common measure. Formally speaking, if segment V is the common measure of A and B, and V is greater than any other common measures, we say V is the greatest common measure of A and B. Given two segments, how to find the greatest common measure? There is a famous ancient recursive method, called the Euclidean algorithm can solve this problem. It is named after the great ancient Greek mathematician Euclid⁶. This algorithm is defined as proposition 3, in book X⁷ of Euclid's Elements[11].

2.2.1 The Euclid's Elements

Euclid of Alexandria is the most prominent ancient Greek mathematician, often referred as “father of geometry”. His *Elements* is one of the most influential works in the history. However, little is known of Euclid's life except that he taught at Alexandria in Egypt. The year and place of his birth and death are unknown. Proclus, the last major Greek philosopher who lived around 450AD introduced Euclid briefly in his *Commentary on the Elements*. He mentioned an interesting story about Euclid. When Ptolemy I of Alexandria (king of Egypt 323BC - 283BC) grew frustrated at the degree of effort required to master geometry via Euclid's *Elements*, he asked if there was a shorter path, Euclid replied there is no royal road to geometry. This becomes the learning maxim of eternal. Another story told by Stobaeus said someone who had begun to learn geometry with Euclid, when he had learnt the first theorem, asked Euclid “What shall I get by learning these things?” Euclid said “Give him three pence since he must make out of what he learns”. Euclid disagreed with the narrow practical



Euclid, About 300BC

⁶The Euclidean algorithm was also developed independently in ancient India and China. The Indian mathematician Aryabhata used this method to solve the Diophantine equation around the end of the 5th century. The Euclidean algorithm was treated as a special case of the Chinese remainder theorem in *Sunzi Suanjing*. Qin Jiushao gave the detailed algorithm in his *Mathematical Treatise in Nine Sections* (数书九章) in 1247.

⁷The same algorithm for integers is also defined as proposition 1, book VII. However, the algorithm for segments covers the integer case.

view of learning[11].

From ancient time to the late 19th century, people considered the *Elements* as a perfect example of correct reasoning. Although many of the results in *Elements* originated with earlier mathematicians, one of Euclid's accomplishments was to present them in a single, logically coherent framework, making it easy to use and easy to reference, including a system of rigorous mathematical proofs that remains the basis of mathematics 23 centuries later. More than a thousand editions have been published, making it one of the most popular books after the Bible. Even today, *Elements* is still widely taught in school⁸ as one of the basic way to train logic reasoning[8].

2.2.2 Euclidean algorithm

Proposition 2.2.1 (Euclid's Elements, Book X, Proposition 3). *To find the greatest common measure of two given commensurable magnitudes.*

The solution Euclid gave only uses recursion and subtraction. It means the greatest common measure can be solved only with ruler and compass essentially. This algorithm can be formalized as the following⁹.

$$\text{gcm}(a, b) = \begin{cases} a = b & : a \\ b < a & : \text{gcm}(a - b, b) \\ a < b & : \text{gcm}(a, b - a) \end{cases} \quad (2.1)$$

Suppose segment a and b are comensurable. If they are equal, then either one is the greatest common measure, we can return a as the result. If a is longer than b , we can use compass to intercept b from a repeatedly (through recursion), then find the greatest common measure for the intercepted segment a' and b ; otherwise if b is longer than a , we intercept a from b repeatedly, and find the greatest common measure for the segment a and b' . Figure 2.8 illustrated the steps when processing two segments. We can also use this algorithm to process two integers 42 and 30. The detailed steps are list in the following table.

$\text{gcm}(a, b)$	a	b
$\text{gcm}(42, 30)$	42	30
$\text{gcm}(12, 30)$	12	30
$\text{gcm}(12, 18)$	12	18
$\text{gcm}(12, 6)$	12	6
$\text{gcm}(6, 6)$	6	6

Repeatedly subtracting b from a to get a' , it's exactly the definition of division with remainder: $a' = a - \lfloor a/b \rfloor b$, or denoted as $a' = a \bmod b$. We can use the division with remainder to replace the repeated subtraction in the origin Euclidean algorithm. Besides that, when one magnitude is an integer multiple of the other, for example $b \leq a$ and a can be divided by b , we know the greatest common measure is b . As the remainder $a \bmod b = 0$, we define $\text{gcm}(0, b) = \text{gcm}(b, 0) = b$. We can compare a and b first, then exchange them if $a < b$. As we know $a \bmod b$ must be less than b , when recursively compute the next time, we can directly exchange them as $\text{gcm}(b, a \bmod b)$. This gives the improved Euclidean algorithm.

$$\text{gcm}(a, b) = \begin{cases} b = 0 & : a \\ \text{otherwise} & : \text{gcm}(b, a \bmod b) \end{cases} \quad (2.2)$$

⁸The most popular version is edited by the French mathematician Lagrange (1736 - 1813).

⁹Term 'gcm' is the abbreviation for the greatest common measure. When a and b are integers, we often use 'gcd' as the abbreviation for the greatest common divisor.

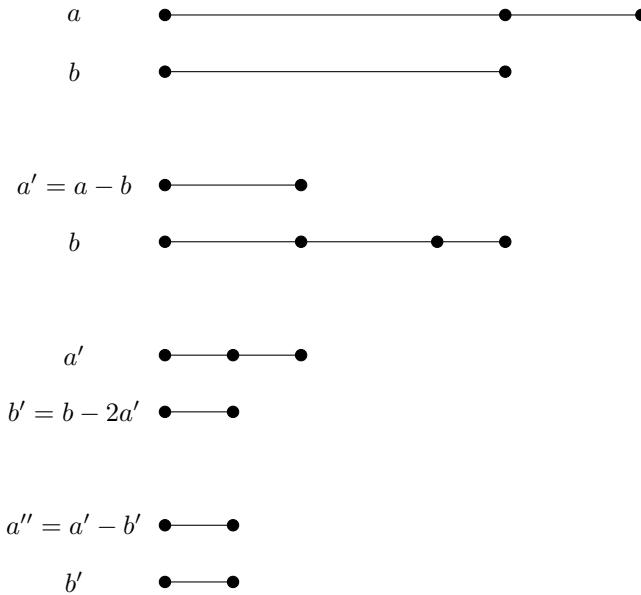


Figure 2.8: Euclidean algorithm example steps.

Why does this algorithm give the greatest common measure? We prove it with two steps. At step one, we prove this algorithm gives the common measure. Suppose $b \leq a$, let the integer q_0 be the quotient, r_0 be the remainder. We have $a = bq_0 + r_0$. As we get the common measure if r_0 is zero, let's focus on the case that r_0 is not zero. We can next express b as $b = r_0q_1 + r_1$, and list the similar equations unless the remainder is not zero.

$$\begin{aligned}
 a &= bq_0 + r_0 \\
 b &= r_0q_1 + r_1 \\
 r_0 &= r_1q_2 + r_2 \\
 r_1 &= r_2q_3 + r_3 \\
 &\dots
 \end{aligned}$$

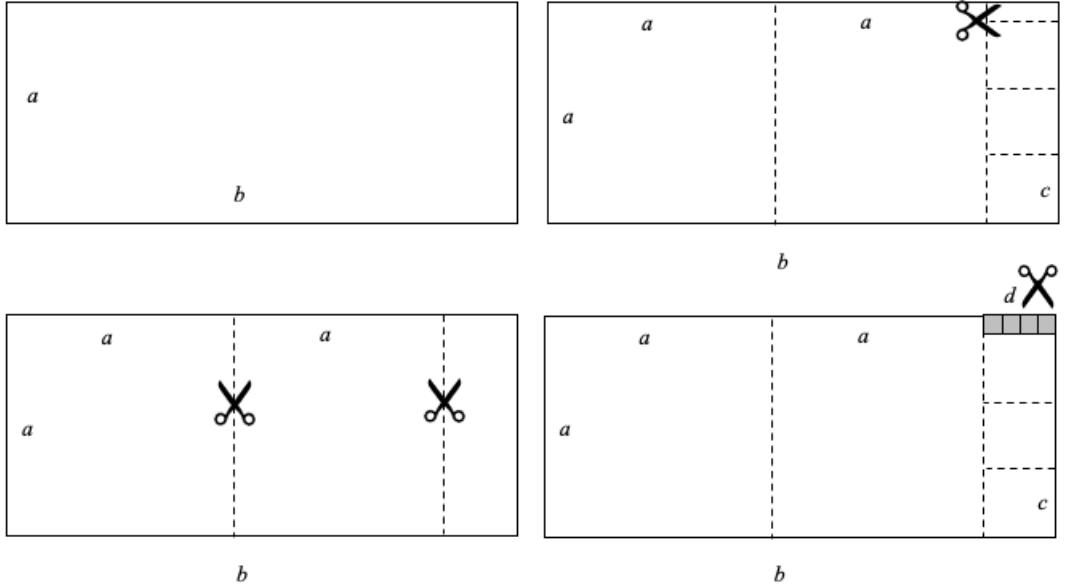
This list grows as long as a and b are commensurable. But it is not infinite. This is because every time, we use compass to intercept integer times. The quotients are integers. And all the remainders are less than the relative divisors. We have $b > r_0 > r_1 > r_2 > \dots > 0$. As the remainder can not be less than zero, and the initial magnitude is finite, we must reach to $r_{n-2} = r_{n-1}q_n$ within finite steps.

Next is to prove r_{n-1} can measure both a and b . Obviously r_{n-1} measures r_{n-2} by definition. Consider the last second equation $r_{n-3} = r_{n-2}q_{n-1} + r_{n-1}$, since r_{n-1} measures r_{n-2} , r_{n-1} also measures $r_{n-2}q_{n-1}$. So it measures $r_{n-2}q_{n-1} + r_{n-1}$, which equals to r_{n-3} . Similarly, we can prove r_{n-1} measures the left hand of every equations step by step upwards to b and a . Thus proves the answer found by Euclidean algorithm, r_{n-1} is the common measure of a and b . Suppose the greatest common measure is g , we have $r_{n-1} \leq g$.

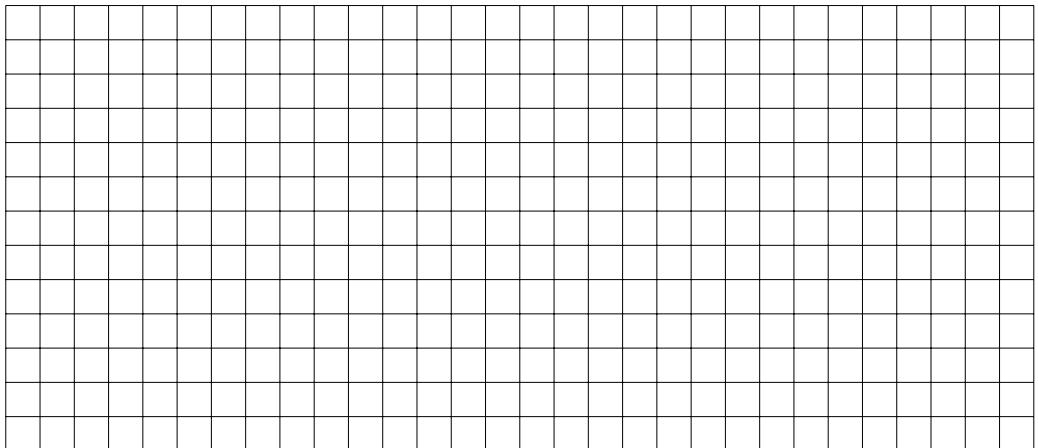
The next step is to prove, for any common measure c for a and b , it measures r_{n-1} . As c is the common measure, both a and b can be expressed with it, let $a = mc$, $b = nc$, where m and n are some integers. Then the first equation $a = bq_0 + r_0$ can be denoted as $mc = ncq_0 + r_0$, as we know $r_0 = (m - nq_0)c$, it means c measures r_0 . Similarly, we can prove c measures r_1, r_2, \dots, r_{n-1} one by one. Thus we proved any common measure also measures r_{n-1} , so the greatest one g also measures r_{n-1} . It means $g \leq r_{n-1}$.

Combine the results from step one and two, that $r_{n-1} \leq g$ and $g \leq r_{n-1}$, we deduce the greatest common measure $g = r_{n-1}$. It does not only prove the correctness of Euclidean algorithm, but also tells us g is the greatest common measure for every pair of magnitudes:

$$g = \text{gcm}(a, b) = \text{gcm}(b, r_0) = \dots = \text{gcm}(r_{n-2}, r_{n-1}) = r_{n-1} \quad (2.3)$$



(a) Recursively cut off squares



(b) Tile the small squares

A geometric description of Euclidean algorithm

2.2.3 Extended Euclidean algorithm

The extended Euclidean algorithm is an extension to the Euclidean algorithm. For given magnitude a and b , in addition to compute their greatest common measure g , it can also find two integers x and y , that satisfy Bézout's identity $ax + by = g$. Why does Bézout's

identity¹⁰ always hold? Here is a proof. We can construct a set, consists of all the positive linear combinations of a and b .

$$S = \{ax + by \mid x, y \in \mathbb{Z} \text{ and } ax + by > 0\}$$



(a) Étienne Bézout, 1730 - 1783



(b) Claude Gaspard Bachet de Méziriac, 1581–1638, who first discovered and proved Bézout's identity for integers.

For line segments, S must not be empty, as it contains at least a (where $x = 1, y = 0$) and b (where $x = 0, y = 1$). Since all the elements in S are positive, there must exist the smallest one. We denote the smallest element as $g = as + bt$. We'll next show that g is the greatest common measure of a and b . Let's express a as the quotient and remainder of g .

$$a = qg + r \tag{2.4}$$

Where the remainder $0 \leq r < g$. It is either zero or belongs to set S , this is because:

$$\begin{aligned} r &= a - qg && \text{From (2.4)} \\ &= a - q(as + bt) && \text{Definition of } g \\ &= a(1 - qs) - bqt && \text{Change to combination of } a \text{ and } b \end{aligned}$$

It means r can be expressed as the linear combination of a and b , therefore, if it's not zero, it must belong to set S . However, this is not possible because we previously defined g as the least positive element in S , while r is less than g . To avoid this contradiction, we know that r has to be zero. From equation (2.4), g measures a . With the same method, we can prove g also measures b . Therefore g is the common measure of them. We'll next prove that g is the greatest one. Consider an arbitrary common measure c for a and b , according to the definition, there exists integers m and n , that $a = mc$ and $b = nc$. Then g can be expressed as:

$$\begin{aligned} g &= as + bt && \text{The definition} \\ &= mcs + nct && c \text{ is common measure of } a \text{ and } b \\ &= c(ms + nt) && g \text{ is multiple of } c \end{aligned}$$

¹⁰Bézout's identity, or Bézout's theorem was first found and proven by French mathematician Méziriac (Claude Gaspard Bachet de Méziriac, 1581–1638) for integers. Bézout proved it hold for polynomials. Bézout identity can be extended to any Euclidean domain and Principle Idean Domain (PID).

It means c measure g , so $c \leq g$. It gives that g is the greatest common measure. Summarize the above, we complete the proof of Bézout's identity. There exists integers, that $ax + by = g$ holds. Besides that, we know the greatest common measure is the minimum positive values among all the linear combinations.

We can deduce the extended Euclid algorithm with Bézout's idendity.

$$\begin{aligned}
 ax + by &= \text{gcm}(a, b) && \text{Bézout's idendity} \\
 &= \text{gcm}(b, r_0) && \text{Euclid algorithm (2.3)} \\
 &= bx' + r_0y' && \text{Use Bézout's idendity for } b \text{ and } r_0 \\
 &= bx' + (a - bq_0)y' && \text{By } a = bq_0 + r_0 \\
 &= ay' + b(x' - y'q_0) && \text{As linear combination of } a \text{ and } b \\
 &= ay' + b(x' - y'\lfloor a/b \rfloor) && q_0 \text{ as the quotient of } a \text{ and } b
 \end{aligned}$$

This gives the recursive case:

$$\begin{cases} x = y' \\ y = x' - y'\lfloor a/b \rfloor \end{cases}$$

The edge case happens when $b = 0$, we have $\text{gcm}(a, 0) = 1a + 0b$. Combine it with the recursive case, we obtain the extended Euclidean algorithm.

$$\text{gcm}_{ex}(a, b) = \begin{cases} b = 0 & : (a, 1, 0) \\ \text{otherwise} & : (g, y', x' - y'\lfloor a/b \rfloor) \text{ 其中 } (g, x', y') = \text{gcm}_{ex}(b, a \bmod b) \end{cases} \quad (2.5)$$

Here is a puzzle can be solved with the extended Euclidean algorithm([14], p50). Given two jars with capacity of 9 and 4 gallons, how to get 4 gallons of water from the river?

There are some variances of this puzzle. The capacities can be other nunmbers. It's said the French mathematician Simèon Denis Poisson solved this puzzle when he was a child.

There are total six operations between two jars. Denote the big one as A with capacity of a ; denote the small one as B with capacity of b :

- Fill the big jar A ;
- Fill the small jar B ;
- Empty the big jar A ;
- Empty the small jar B ;
- Pour the water from jar A to jar B ;
- Pour the warer from jar B to jar A .

The last two operations stop when either jar is empty or full. Below example shows a list of operations (suppose $b < a < 2b$ hold).

No matter what sequence of operations, the water in the jars can always be expressed as $ax + by$ where x and y are integers. It means the water we get is the linear combination of a and b . From the proof of Bézout's identity, we know the smallest positive number of this linear combination is exactly the greatest common measure g . We can tell if it is possible to get c gallons of water if and only if c can be measured by g ¹¹. We assume c is not greater than the capacity of the bigger jar.

¹¹If and only if c can be devided by the greatest common divisor g for integer capacities.

A	B	Operation
0	0	start
0	b	fill B
b	0	pour from B to A
b	b	fill B
a	$2b - a$	pour from B to A
0	$2b - a$	empty A
$2b - a$	0	pour from B to A
$2b - a$	b	fill B
a	$3b - 2a$	pour from B to A
...

Table 2.1: The water in the jars and the operations.

For example, we can't get 5 gallons of water with two jars of 4 gallons and 6 gallons. This is because the greatest common divisor of 4 and 6 is 2. Which can't divide 5. (In other words, we can't get odd gallons of water with two jars of even gallons capacities.) If a and b are coprime, i.e. $\gcd(a, b) = 1$, then we are sure to be able to get any natural number c gallons of water.

Although we can tell that the puzzle is solvable when g measures c , we still don't know the detailed steps to pour water. Actually, the steps can be decided as far as we can find two integers x and y , satisfying $ax + by = c$. If $x > 0, y < 0$, it means we need fill jar A x times, and empty jar B y times; Else if $x < 0, y > 0$, we need empty jar A x times, and fill jar B y times.

For instance, let the capacity of the big jar $a = 5$ gallons, the small jar $b = 3$ gallons. We want to get $c = 4$ gallons of water. As $4 = 3 \times 3 - 5$, so $x = -1, y = 3$. We can arrange the steps as below table.

A	B	operation
0	0	start
0	3	fill B
3	0	pour from B to A
3	3	fill B
5	1	pour from B to A
0	1	empty A
1	0	pour from B to A
1	3	fill B
4	0	pour from B to A

Table 2.2: Steps to get 4 gallons of water.

We can see from these steps, jar B is filled 3 times, jar A is emptied 1 time. The next question is how to find x and y that satisfies $ax + by = c$. With the extended Euclid algorithm, we can find a solution to Bézout's identity $ax_0 + by_0 = g$. Since c is m times of the greatest common measure g , we can make a solution by multiplying x_0 and y_0 m times.

$$\begin{cases} x_1 = x_0 \frac{c}{g} \\ y_1 = y_0 \frac{c}{g} \end{cases}$$

From this solution, we can generate all the integer solutions to the Diophantine equa-

tion ¹².

$$\begin{cases} x = x_1 - k \frac{b}{g} \\ y = y_1 + k \frac{a}{g} \end{cases} \quad (2.6)$$

Where k is an integer. Thus we get all the integer solutions to the water jar puzzle. Further, we can find a special k , that minimizes $|x| + |y|$, it gives the fast pouring steps¹³. Below is the example Haskell program solves this puzzle.

```
import Data.List
import Data.Function (on)

— Extended Euclidean Algorithm
gcmex a 0 = (a, 1, 0)
gcmex a b = (g, y', x' - y' * (a `div` b)) where
  (g, x', y') = gcmex b (a `mod` b)

— Solve the linear Diophantine equation ax + by = c
solve a b c | c `mod` g ≠ 0 = (0, 0, 0, 0) — no solution
             | otherwise = (x1, u, y1, v)
  where
    (g, x0, y0) = gcmex a b
    (x1, y1) = (x0 * c `div` g, y0 * c `div` g)
    (u, v) = (b `div` g, a `div` g)

— Optimal by minimize |x| + |y|
jars a b c = (x, y) where
  (x1, u, y1, v) = solve a b c
  x = x1 - k * u
  y = y1 + k * v
  k = minimumBy (compare `on` (λi → abs (x1 - i * u) +
                                abs (y1 + i * v))) [-m..m]
  m = max (abs x1 `div` u) (abs y1 `div` v)
```

After figure out x and y , we can populate the steps as in Appendix of this chapter.

2.2.4 Influence of Euclidean algorithm

Euclidean algorithm was developed to find the the greatest common divisor for two integers in spirit of all things are made of numbers. However, Euclid applied it to the abstract geometric magnitudes. We see the separation of geometry from numbers¹⁴. Geometry was not built on top of numbers, but developed independently to solve the generic problems not limit to numbers. The ancient Greek formed such a tradition, that even for any conclusion about number, one need to give proof in terms of geometry. It kept influencing people till the 16th Century. For example, the Italian mathematician Gerolamo Cardano still used geometric cubic filling method in his book *Ars Magna* when solving the cubic and four-order equations in 1545[12].

The Euclidean algorithm is the most famous recursive algorithm. German mathematician, Dirichlet, the founder of analytic number theory, commented in his *Lectures*

¹²Naming after the acient Greek mathematician, Diophantus of Alexandira (about 200 - 284AD). In his book *Arithmetica*, he made important advances in mathematical notation, becoming the first person known to use algebraic notation and symbolism. Diophantus is often called “the father of algebra” because he contributed greatly to number theory, mathematical notation, and because *Arithmetica* contains the earliest known use of syncopated notation[12].

¹³One way to get this special k is to represent the solution as two lines in Cartesian plain. When taking absoluted value, it flips the lower part to the x-axis. Then we can find the k minimizes $|x| + |y|$ from the figure.

¹⁴This is the reason why we name Euclidean algorithm gcm, but not gcd.

on Number Theory¹⁵, The structure of the whole number theory is based on the same foundation, which is the greatest common divisor algorithm. The modern RSA cryptosystem¹⁶ utilizes the extended Euclidean algorithm directly. We demonstrated how to figure out the integer solutions for binary linear Diophantine equation $ax + by = c$. Find the greatest common measure g . There's no integer solution if g does not divide c . Otherwise, for the x_0, y_0 satisfying Bézout identity, duplicate them c/g times to get a special solution x_1, y_1 , then define the common solution of $x = x_1 - kb/g$, and $y = y_1 + ka/g$.

The Euclid algorithm is a double-edged sword. The powerful recursive method can be applied to attack the corner stone of the concept that all things are made of numbers. The Pythagoreans believed any two numbers must have common measurement, because all things and phenomenon are essentially ratio of integer to integer. About 470AC, Hippasus, a student in Pythagorean school attempted to find the common measure for the side and diagonal of a square. However, no matter how small magnitude being used, he could not measure them. It surprised the people, and led to a crisis for the foundation of Pythagoreans. There is also saying that, Hippasus was inspired by the mysterious pentagram logo of Pythagorean school. The Pythagoreans use pentagram as the school's badge and liaison symbol. There was a story about a school member met difficulty in a foreign land, poor and sick. The landlord helped to take care of him. He drew a pentagram on the door before dead. A few years later, someone in Pythagorean school saw the sign. He asked about the past, paid the landlord a lot of money then left[8]. In the Walt Disney's film *Donald in Mathmagic Land* in 1959, Duck Donald met Pythagoras and his friends, they discovered the principles of music scale together. After shaking hands with Pythagoras, who then vanishes, Donald found on his hand a pentagram, the symbol of the secret Pythagorean society. As shown in figure 2.12, there is another story said that Hippasus also found segment AC and AG couldn't be commensurable.



Hippasus of Metapontum, about 5th Century, BC.

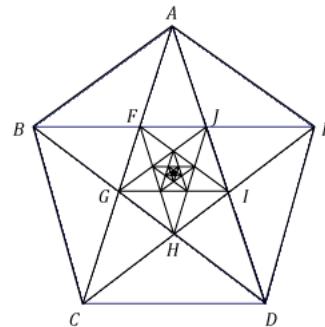


Figure 2.12: The recursive pentagram

The Scottish mathematician George Chrystal reconstructed Hippasus's proof in the 19th Century. Using reduction to absurdity, suppose there exists a segment c that measures both the side and diagonal of the square. From the definition of the common

¹⁵Lejeune Dirichlet, J.P.G.; Richard Dedekind (1863). *Vorlesungen über Zahlentheorie*. F. Vieweg und sohn.

¹⁶RSA is the world first public-key asymmetric crypto algorithm developed by Ron Rivest, Adi Shamir, and Leonard Adelman in MIT, 1977. The acronym RSA is made of the initial letters of their surnames.

measurement, let the side be mc , and the diagonal be nc , where both m, n are integers. As shown in figure 2.13, taking the side as radius, we draw an arc with A as the center, which intersects the diagonal AC at point E . Then draw a line from E that is perpendicular to the diagonal, and intersects side BC at point F .

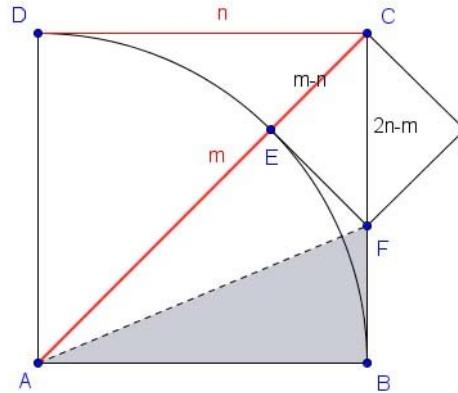


Figure 2.13: The side and the diagonal of the square.

As it is an arc, the length of AE equals to the square side. Thus the length of segment AE is $(m - n)c$. Because EF is perpendicular to AC , while angle $\angle ECF$ spans 45° , therefore the triangle ECF is the isosceles right triangle. Since the isosceles triangle has two sides of equal length, we have $|EC| = |EF|$ holds. Observe two right triangles $\triangle AEF$ and $\triangle ABF$. Side AE equals AB , and AF is the shared side, they congruent. Then we have $|EF| = |FB|$. As the result, the three segments $|EC| = |EF| = |FB|$. Therefore the length of FB is also $(m - n)c$, while segment CF can be get by deducing FB from CB , which is $nc - (m - n)c = (2n - m)c$. We list all the results as below.

$$\left\{ \begin{array}{l} |AC| = mc \\ |AB| = nc \end{array} \right. \quad \left| \quad \left\{ \begin{array}{l} |CF| = (2n - m)c \\ |CE| = (m - n)c \end{array} \right. \right.$$

As both m, n are integers, it's obvious that c measures both the diagonal CF and side CE of the small square. Using the same method as above, we can draw another even smaller square. Repeating it leads to smaller and smaller infinite squares. But c measures both diagonal and side for every square. Since m, n are finite integers, we can't endlessly duplicate this process, which leads to absurdity. Therefore, our assumption is not true, there is no common measure for the diagonal and side of a square.

It is a loophole in Pythagorean theory about all things are made of numbers, there exists segments that can't be represented by ratio of integers. It was said Hippasus was murdered due to this finding. The Pythagoreans didn't want that secret be disclosed, they drowned Hippasus at sea. The discovery of irrational number greatly boost mathematics. The ancient Greek philosophers and mathematicians considered this problem seriously. After the work of Odoxos, Aristotle, and Euclid, they finally strictly defined the incommensurable magnitudes, and incorporate it into the ancient Greek mathematics through geometry.

Proposition 2.2.2 (Euclid's Elements, Book X, Proposition 2). *If, when the less of two*

unequal magnitudes is continually subtracted in turn from the greater that which is left never measures the one before it, then the two magnitudes are incommensurable.

It's interesting that the incommensurable is defined by checking whether the Euclidean algorithm terminates or not. Because Euclidean algorithm is recursive, it means the condition is essentially whether recursion terminates or not. It brings our attention to the nature of recursion, what's recursion? How to represent recursion in a formal way?

Exercise 2.1

1. The Euclidean algorithm described in this section is in recursive manner. Try to eliminate recursion, implement it and the extended Euclidean algorithm only with loop.
2. Most programming environments require integers for modular operation. However, the length of segment isn't necessarily integer. Implement a modular operation that manipulates segments. What's about its efficiency?
3. In the proof of Euclidean algorithm, we mentioned "Remainders are always less than the divisor. We have $b > r_0 > r_1 > r_2 > \dots > 0$. As the remainder can not be less than zero, and the initial magnitude is finite, the algorithm must terminate." Can r_n infinitely approximate zero, but not be zero? Does the algorithm always terminate? What does the precondition that a and b are commensurable ensure?
4. For the binary linear Diophantine equation $ax + by = c$, let x_1, y_1 and x_2, y_2 be two pairs of solution. Proof that the minimum of $|x_1 - x_2|$ is $b / \text{gcm}(a, b)$, and the minimum of $|y_1 - y_2|$ is $a / \text{gcm}(a, b)$
5. For the regular pentagon with side of 1, how long is the diagonal? Proof that in the pentagram shown in figure 2.12, the segment AC and AG are incommensurable. What's their ratio in real number?

2.3 The λ calculus

Recursion would not be a problem if performed by human. As the intelligent beings, we are able to enter the next level of computation when meet recursion, and return back to the upper level after that. However, it matters when instruct machine to compute. Several computation models were developed in 1930s independently. The most famous ones are Turing machine (1935 by Turing), λ -calculus (1932 - 1941 by Church, and 1935 by Stephen Kleene. The Greek letter λ is pronounced as lambda), and recursive function (1934 by Jacques Herbrand and Kurt Gödel).

Turing was an English mathematician, computer scientist, and logician. Turing was highly influential in the development of theoretical computer science, providing a formalization of the concepts of algorithm and computation with the Turing machine, which can be considered a model of a general-purpose computer. Turing is widely considered to be the father of theoretical computer science and artificial intelligence[15].

During the Second World War, Turing worked for the Government at Bletchley Park, Britain's codebreaking center that produced Ultra intelligence. He devised a number of techniques for speeding the breaking of German ciphers, including improvements to the pre-war Polish bombe method, an electromechanical machine that could find settings for the Enigma cipher machine. Turing played a pivotal role in cracking intercepted coded messages that enabled the Allies to defeat the Nazis in many crucial engagements. It has been estimated that this work shortened the war in Europe by more than two years and saved millions lives.

After the war, Turing worked on the Automatic Computing Engine (ACE), which was one of the first designs for a stored-program computer. the Pilot ACE executed its



(a) Alan Mathison Turing, 1912 - 1954



(b) Alonzo Church, 1903 - 1995

first program on 1950, and a number of later computers around the world owe much to it. The full version of ACE was built in 1958 after his death. From 1950, Turing worked in “Computing Machinery and Intelligence”. He addressed the problem of artificial intelligence, and proposed an experiment that became known as the Turing test, an attempt to define a standard for a machine to be called “intelligent”. The idea was that a computer could be said to “think” if a human interrogator could not tell it apart, through conversation, from a human being. Turing was elected a Fellow of the Royal Society (FRS) in 1951 at the age of 39. Since 1966, the Turing Award has been given annually by the Association for Computing Machinery (ACM) for technical or theoretical contributions to the computing community. It is widely considered to be the computing world’s highest honour, equivalent to the Nobel Prize.

The formalization of computation itself is called *Metamathematics*. This attempt led to a great work, the λ -calculus. There’s a interesting story about the name. When considering the computation itself, people realized we should distinguish function and its evaluation result. For example, if we say ‘if x is odd, then $x \times x$ is also odd’, we mean the evaluated value of the function; while if we say ‘ $x \times x$ is monotonic increasing’, we mean the function itself. To differentiate these two concepts, we write function as $x \mapsto x \times x$, but not only $x \times x$.

The ‘ \mapsto ’ symbol was introduced by Nicolas Bourbaki¹⁷ around 1930. Russel and Whitehead used the notation $\hat{x}(x \times x)$ in their famous book *Principia Mathematica* in 1910s. Church wanted to use a similar notation in 1930s, however, the publisher he worked with didn’t know how to print the ‘hat’ symbol on top of x . Alternatively, they printed the uppercase Greek letter Λ before x , and later changed to lowercase letter λ . That is the reason why we see it’s in the form of $\lambda x. x \times x$ today[16]. Although the $x \mapsto x \times x$ presentation is widely accepted, people tend to use Church’s notation particularly in logic and computer science, and named it as the ‘ λ -calculus’.

2.3.1 Expression reduction

We start from some simple example of λ -calculus to demonstrate how to formalize the computation and algorithm. For basic arithmetic operations of plus, minus, times, and subtraction, we treat them also as functions. For instance $1 + 2$ can be considered as

¹⁷Nicolas Bourbaki is the collective pseudonym of a group of (mainly French) mathematicians. Their aim is to reformulate mathematics on an extremely abstract and formal but self-contained basis with the goal of grounding all of mathematics on set theory. Many famous mathematician participants the Bourbaki group, like Henri Cartan, Claude Chevalley, Jean Dieudonné, André Weil, Laurent Schwartz, Jean-Pierre Serre, Alexander Grothendieck, and Serge Lang.

applying function ‘+’ to two arguments 1 and 2. Following the traditions to write function name first, this expression can be written as $(+ 1 2)$. The process of expression evaluation can be viewed as a series of reduction steps. For example:

$$\begin{aligned} & (+ (\times 2 3) (\times 4 5)) \\ \rightarrow & (+ 6 (\times 4 5)) \\ \rightarrow & (+ 6 20) \\ \rightarrow & 26 \end{aligned}$$

The arrow symbol \rightarrow read as ‘reduce to’. Note that when apply f to variable x , we don’t write it as $f(x)$, but as $f x$. For multi-variable functions, like $f(x, y)$, we don’t write as $(f (x, y))$, but use the uniformed way as $((f x) y)$. Therefore, three plus four should be written as $((+ 3) 4)$. Expression $(+ 3)$ actually represents a function, it adds 3 to any argument passed in. As a whole, this expression means ‘apply function $+$ to a variable that equals to 3, it gives a function as result. Then apply this function to another variable that equals to 4’. By this means, we treat every function only takes one argument. This method was first introduced by Schönfinkel (1889 - 1942) in 1924, then widely used by Haskell Curry from 1958. It is known as *Currying*[17].

There will be too many parentheses if written strictly in Curried form. To make it concise, we’ll omit some parentheses without causing ambiguity. For example we’ll simplify $((f ((+ 3) 4)) (g x))$ to $(f (+ 3 4) (g x))$.

We need define the meanings for basic components when perform expression reduction. For arithmetic operation, we’ve defined plus and multiplication in chapter 1 on top of Peano axioms. We can use the similar approach to define their reversed operation for minus and divide. For the numbers as operands, we can define them with zero and the successor. With these being clarified in theory, we realize the arithmetic operators and numbers built-in for performance consideration. The logic and, or, not, Boolean constant value true and false are also typically built-in realized. The conditional expression can be realized in McCarthy form like $(p \mapsto f, g)$ introduced in chapter 1, or defined as the below **if** form:

$$\begin{aligned} \text{if } true & \text{ then } e_t \text{ else } e_f \mapsto e_t \\ \text{if } false & \text{ then } e_t \text{ else } e_f \mapsto e_f \end{aligned}$$

Where both e_t , e_f are expressions. For the compound data structure defined by *cons* in chapter 1, we also need define functions to extract every part:

$$\begin{aligned} \text{head } (cons \ a \ b) & \mapsto a \\ \text{tail } (cons \ a \ b) & \mapsto b \end{aligned}$$

2.3.2 λ abstraction

We told the story about how λ symbol was introduced. λ abstraction is a method to construct function. Let’s use an example to understand every component in λ abstraction.

$$(\lambda x. + \ x \ 1)$$

A λ abstraction contains four parts. First is the λ symbol, it means we start to define a function. The next part is the variable. It’s x in this example. The variable is called formal parameter. Following the formal parameter, there is a dot. The rest part is the function body that extends to the right most. It’s $+ x 1$ in our example. We can add parentheses to avoid ambiguity about the right boundary of the body. For our example, it will be $(+ x 1)$. To make it easy for memory, we write the four parts in λ abstraction corresponding to natural language as below.

$(\lambda \quad \quad \quad x \quad \quad \cdot \quad \quad + x 1)$
 ↑ ↑ ↑ ↑
 That function of x which add x to 1

We'll also use the equivalent $x \mapsto x + 1$ form for convenience. Note that λ abstraction is not equivalent to λ expression, λ abstraction is only one type of λ expressions. λ expression also has two other types:

$\langle \text{exp} \rangle$	$=$	$\langle \text{constant} \rangle$	built-in constants, numbers, Boolean etc.
		$\langle \text{variable} \rangle$	variable names
		$\langle \text{exp} \rangle \langle \text{exp} \rangle$	applications
		$\lambda \langle \text{variable} \rangle . \langle \text{exp} \rangle$	λ abstraction

2.3.3 λ conversion rules

When evaluate the below λ expression, we need to know the value for the global variable y . On the other hand, we needn't know the value of variable x , because it appears as the formal parameter.

$$(\lambda x. + x y) 2$$

Different from y , we say x is bound to λx . When apply this λ abstraction to argument 2, we replace x by 2. On the contrary, y is not bound by λ , we say y is free. Overall, the expression value is determined by the unbound free variable values. A variable is either bound or free. Here is another example:

$$\lambda x. + ((\lambda y. + y z) 3) x$$

We can make it clear with the arrow notation:

$$x \mapsto ((y \mapsto y + z) 3) + x$$

We see that both x and y are bound, while z is free. In the more complex expression, the same variable name may be bound, and at the same time appear as free. For example:

$$+ x ((\lambda x. + x 1) 2)$$

Written in the arrow form:

$$x + ((x \mapsto x + 1) 2)$$

We see that, x is free in its first occurrence, but is bound in the second occurrence. The same name represents for different variable can cause confusion in a complex expression. To solve the name conflict, we introduce the first λ conversion rule, α -conversion. Where α is the Greek letter alpha. This rule allows us to rename a variable in λ expression to another one. For example:

$$\lambda x. + x 1 \quad \xleftarrow{\alpha} \quad \lambda y. + y 1$$

or written in the arrow form:

$$x \mapsto x + 1 \quad \xleftarrow{\alpha} \quad y \mapsto y + 1$$

We mentioned that λ abstraction is a method to construct function. How to apply the constructed function to specific parameter? In order to do that, we need the second λ conversion rule, the β -conversion. When using this rule, we replace all the free occurrence of formal parameter in function body to its value. For example:

$$(x \mapsto x + 1) \ 2$$

According to the conversion rule, applying the λ abstraction $x \mapsto x + 1$ to the free variable 2 gives $2 + 1$. $2 + 1$ is the result when replace the formal parameter x in function body $x + 1$ with 2. It can be written in arrow form as below:

$$(x \mapsto x + 1) \ 2 \xrightarrow{\beta} 2 + 1$$

We call the conversion along the direction of this arrow as β -reduction. When using it reversely, we call it β -abstraction. Let's get familiar with β -reduction with more examples. First is about multiple occurrences of the formal parameter.

$$\begin{array}{ccc} (x \mapsto x \times x) \ 2 & \xrightarrow{\beta} & 2 \times 2 \\ & \longrightarrow & 4 \end{array}$$

Here's another example that the formal parameter does not occur.

$$(x \mapsto 1) \ 2 \xrightarrow{\beta} 1$$

This is a typical example of constant projection. The next is a multiple steps reduction.

$$\begin{array}{ccc} (x \mapsto (y \mapsto y - x)) \ 2 \ 4 & \xrightarrow{\beta} & (y \mapsto y - 2) \ 4 & \text{Currying} \\ & \xrightarrow{\beta} & 4 - 2 & \text{Inner reduction} \\ & \longrightarrow & 2 & \text{Built-in arithmetic} \end{array}$$

We see that the reduction from inner to outer is a repeated Currying process. We write the multiple steps reduction in a simplified way sometimes:

$$(\lambda x.(\lambda y.E)) \Rightarrow (\lambda x.\lambda y.E)$$

Where E represents the function body. Written in the arrow form:

$$(x \mapsto (y \mapsto E)) \Rightarrow (x \mapsto y \mapsto E)$$

When applying a function with β -reduction, the parameter can be another function. For example:

$$\begin{array}{ccc} (f \mapsto f \ 5) \ (x \mapsto x + 1) & \xrightarrow{\beta} & (x \mapsto x + 1) \ 5 \\ & \xrightarrow{\beta} & 5 + 1 \\ & \longrightarrow & 6 \end{array}$$

The last conversion rule we'll introduce is the η -conversion. It's defined as the following:

$$(\lambda x.F \ x) \xleftrightarrow{\eta} F$$

or written in the arrow form:

$$x \mapsto F \ x \xleftrightarrow{\eta} F$$

Where F is a function, and x is not the free variable in F . Here is an example:

$$(\lambda x. + \ 1 \ x) \xleftrightarrow{\eta} (+ \ 1)$$

In this example, the λ -expressions in both sides of η -conversion behave same. When apply to a parameter, the effect is add 1 to it. The reason why x must not be the free variable in F in η -conversion is to avoid wrongly converting expression like $(\lambda x. + x x)$ to $(+ x)$. We can see that x is the free variable in $(+ x)$. It's also necessary to limit F to function, otherwise it could convert 1 to $(\lambda x.1 x)$, which does not make sense. We call the transform from left to right as η -reduction.

So far, we introduced the three conversion rules for λ expression. Summarized as below:

1. α -conversion to change the name for formal parameters;
2. β -reduction to realize function application;
3. η -reduction to eliminate redundant λ abstraction.

Besides these three rules, we call the built-in functions, like arithmetic operations, logic and, or, not as δ -conversion. Some materials about λ -calculus uses another simplified notation. When perform β -reduction for expression $(\lambda x.E) M$, we use M to replace x in E , written the result as $E[M/x]$. Then the three conversion rules can be simplified as below:

conversion	λ form	arrow form
α	$(\lambda x.E) \xleftarrow{\alpha} \lambda y.E[y/x]$	$x \mapsto E \xleftarrow{\alpha} y \mapsto E[y/x]$
β	$(\lambda x.E) M \xleftarrow{\beta} E[M/x]$	$(x \mapsto E) M \xleftarrow{\beta} E[M/x]$
η	$(\lambda x.E x) \xleftarrow{\eta} E$	$x \mapsto E x \xleftarrow{\eta} E$

All these conversions can be applied in both directions, left to right or reversed. It raises two questions by nature. First, will the reduction terminate? Second, do the different reduction steps lead to the same result? For the first question, the answer is not deterministic. The reduction process is not ensure to terminate¹⁸. Here is an example of endless loop: $(D D)$, where D is defined as $\lambda x.x x$. Or written in the arrow form: $x \mapsto x x$. If we attempt to simplify it, we'll get the following result:

$$\begin{aligned}
 (D D) &\rightarrow (x \mapsto x x) (x \mapsto x x) && \text{Substitute with definition of } D \\
 &\xrightarrow{\alpha} (x \mapsto x x) (y \mapsto y y) && \alpha\text{-conversion for the second } \lambda \text{ abstraction} \\
 &\xrightarrow{\beta} (y \mapsto y y) (y \mapsto y y) && \text{replace } x \text{ with the second expression} \\
 &\xrightarrow{\alpha} (x \mapsto x x) (x \mapsto x x) && \text{replace } y \text{ with } x \\
 &\rightarrow (x \mapsto x x) (x \mapsto x x) && \text{repeat the above steps} \\
 &\dots
 \end{aligned}$$

A more interesting example is $(\lambda x.1) (D D)$, if firstly reduce the $(\lambda x.1)$ part, it terminates with the result of 1. But if firstly reduce $(D D)$ part, it loops endlessly as shown above. Church and his student Rosser¹⁹ proved a pair of theorems that completely answered the second question.

¹⁸Note the answer is not 'no', but none deterministic. It's essentially as same as the Turing halting problem. There is no determined process can tell if a given reduction process terminates. We'll introduce the details in the last chapter.

¹⁹John Barkley Rosser Sr. 1907 - 1989. was an American mathematician and logician. Besides Church-Rosser confluence theory, he also found the Kleene-Rosser paradox with Kleene. In number theory, he developed what is now called "Rosser sieve" and proved Rosser theorem that the n -th prime number $p_n > n \ln n$. Rosser gave a stronger form for the Gödel's first incompleteness theorem. He improved the none deterministic proposition to 'For any proof to this proposition, there exists a shorter one for the negated one.'

Theorem 2.3.1 (Church-Rosser theorem 1). *If $E_1 \leftrightarrow E_2$, then there exists E that $E_1 \rightarrow E$ and $E_2 \rightarrow E$.*

It means, if the reduction process terminates, then the results confluence. Varies reduction steps give the same result, as shown in figure 2.15. Church and Rosser proved the second theorem on top of the first one. We need the concept of the *normal form* to understand it. The normal form, also known as β normal form, is an expression that we can't do any further β -reduction. It means all the functions have already been applied. A more strict normal form is the $\beta - \eta$ normal form, which neither β -reduction, nor η -reduction can be performed. For example, $(x \mapsto x+1) y$ is not normal form, because we can apply β -reduction to change it to $y+1$. The following defines the normal form recursively.

$\text{normal}((\lambda x.y) z)$	=	false
$\text{normal}(\lambda x.(f x))$	=	false
$\text{normal}(x y)$	=	$\text{normal}(x) \wedge \text{normal}(y)$
$\text{normal}(x)$	=	true

can do further β -reduction

can do further η -reduction

Application: both function and parameter are normal
others

Theorem 2.3.2 (Church-Rosser Theorem 2). *If $E_1 \rightarrow E_2$, and E_2 is normal form, then there exists normal order to convert from E_1 to E_2 .*

Note this theorem requires the reduction process terminates. The normal order is the order to reduce from left to right, from outer to inner.

2.4 Definition of recursion

With λ abstraction, we can define some simple functions. How to define recursive function? The factorial for example can be recursively defined as below:

$$\text{fact} = n \mapsto \text{if } n = 0 \text{ then 1 else } n \times \text{fact}(n - 1)$$

But this is not a valid λ expression. The λ abstraction can only define anonymous functions, while we don't know how to name a function. Observe the recursive factorial definition, it has the pattern like:

$$\text{fact} = n \mapsto (\dots \text{fact} \dots)$$

Reversely using the β -reduction (i.e. β -abstraction), we can get:

$$\text{fact} = (f \mapsto (n \mapsto (\dots f \dots))) \text{ fact}$$

It can be further abstract to:

$$\text{fact} = H \text{ fact} \quad (2.7)$$

where

$$H = f \mapsto (n \mapsto (\dots f \dots))$$

Note that after this conversion, H is not recursive any more. It is a normal λ expression. Observe the equation (2.7), it represents recursion. It is in equation form, which reminds us about the differential equation. For example, solving the differential equation $y' = \sin(x)$ gives $y = a - \cos(x)$. If we can solve the equation $F = H F$, then we are able to define factorial independently. Further observe this equation, it means when apply H to F , the result is still F . Such concept is called *fixed point* in mathematics. We say that F is the fixed point of H . Here's another example: the fixed points for λ expression $x \mapsto x \times x$ are 0 and 1, this is because we have $(x \mapsto x \times x) 0 = 0$ and $(x \mapsto x \times x) 1 = 1$.

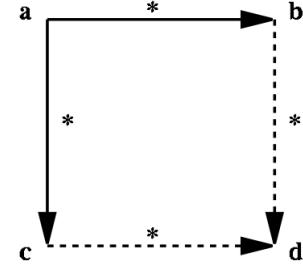


Figure 2.15: Church-Rosser confluence

2.4.1 Y combinator

We want to figure out the fixed point for H , it's obvious that the fixed point only depends on H . To do that, we introduce a function Y . It accepts a function, then returns its fixed point. Y behaves like this:

$$Y H = H (Y H) \quad (2.8)$$

Y is called fixpoint combinator. By using Y , we define the solution to equation (2.7).

$$fact = Y H \quad (2.9)$$

Such $fact$ is a non-recursive definition. We can verify this solution as the following:

$$\begin{aligned} fact &= Y H && \text{By (2.9)} \\ &= H (Y H) && \text{By (2.8)} \\ &= H fact && \text{Reverse of (2.9)} \end{aligned}$$

Y is so powerful that it can represent any recursive functions. However, it is still a black box to us. We need realize it in λ abstraction.

$$Y = \lambda h.(\lambda x.h (x x)) (\lambda x.h (x x)) \quad (2.10)$$

Written in the arrow form:

$$Y = h \mapsto (x \mapsto h (x x))(x \mapsto h (x x))$$

We are opening a magic box, let's verify if Y in λ abstraction behaves as we expected: $Y H = H (Y H)$.

Proof.

$$\begin{aligned} Y H &= (h \mapsto (x \mapsto h (x x))(x \mapsto h (x x))) H && \text{Definition of } Y \\ &\xrightarrow{\beta} (x \mapsto H (x x)) (x \mapsto h (x x)) && \beta\text{-reduction, substitute } h \text{ with } H \\ &\xrightarrow{\alpha} (y \mapsto H (y y)) (x \mapsto h (x x)) && \alpha\text{-conversion for the first half} \\ &\xrightarrow{\beta} H ((x \mapsto H (x x)) (x \mapsto h (x x))) && \beta\text{-reduction, substitute } y \text{ with the second half} \\ &\xrightarrow{\beta} H (h \mapsto (x \mapsto h (x x)) (x \mapsto h (x x))) H && \beta\text{-abstraction, extract } H \text{ as parameter} \\ &= H (Y H) && \text{Definition of } Y \end{aligned}$$

□

Finally, let us define factorial with Y :

$$Y (f \mapsto (n \mapsto \text{if } n = 0 \text{ then } 1 \text{ else } n \times f (n - 1)))$$

It is more significant in mathematics than in practice to define Y in λ abstraction. Y is often realized as a built-in function in real environment, which directly converts $Y H$ to $H (Y H)$.

2.5 The impact of λ calculus

The significant of the λ calculus is that it models the complex computation process with a set of simple rules. Consider the way of representing the Euclidean algorithm in λ expressions, then perform β -reduction to evaluate the result, it's feasible although looks complex from realization perspective. It does not limit to Euclidean algorithm,

but can represents any computable functions. People later proved that λ calculus and Turing machine are equivalent. One advantage of λ calculus is that it only uses the traditional function concept in mathematics. People used it in 1930s to formalize the metamathematics. However, Kleene and Rosser proved that the original lambda calculus was inconsistent in 1935. Subsequently, in 1936 Church isolated and published just the portion relevant to computation, what is now called the untyped lambda calculus. In 1940, he also introduced a computationally weaker, but logically consistent system, known as the simply typed lambda calculus.

We showed how to use λ calculus to define arithmetic operations, logic operations, the simple functions, and recursive functions. There is also an important thing, the composed data structure, need be covered. Actually, λ calculus can define *cons*, *head*, and *tail* as well:

$$\begin{aligned} \text{cons} &= (\lambda a. \lambda b. \lambda f. f a b) \\ \text{head} &= (\lambda c. c (\lambda a. \lambda b. a)) \\ \text{tail} &= (\lambda c. c (\lambda a. \lambda b. b)) \end{aligned}$$

Written in the arrow form:

$$\begin{aligned} \text{cons} &= a \mapsto b \mapsto f \mapsto f a b \\ \text{head} &= c \mapsto c (a \mapsto b \mapsto a) \\ \text{tail} &= c \mapsto c (a \mapsto b \mapsto b) \end{aligned}$$

Let's verify that $\text{head} (\text{cons} p q) = p$ holds.

$$\begin{aligned} \text{head} (\text{cons} p q) &= (c \mapsto c (a \mapsto b \mapsto a)) (\text{cons} p q) \\ &\xrightarrow{\beta} (\text{cons} p q) (a \mapsto b \mapsto a) \\ &= ((a \mapsto b \mapsto f \mapsto f a b) p q) (a \mapsto b \mapsto a) \\ &\xrightarrow{\beta} ((b \mapsto f \mapsto f p b) q) (a \mapsto b \mapsto a) \\ &\xrightarrow{\beta} (f \mapsto f p q) (a \mapsto b \mapsto a) \\ &\xrightarrow{\beta} (a \mapsto b \mapsto a) p q \\ &\xrightarrow{\beta} (b \mapsto p) q \\ &\xrightarrow{\beta} p \end{aligned}$$

It tells us that the composite data structure needn't be built-in realized. We can use λ to define them. The exercise of this chapter demands you to consider how to define the natural numbers in Peano Axioms, the Boolean values, and the logic operators with λ calculus.

Exercise 2.2

1. Use λ conversion rules to verify $\text{tail} (\text{cons} p q) = q$.
2. We can define numbers with λ calculus. The following definition is called Church numbers:

$$\begin{aligned} 0 &: \lambda f. \lambda x. x \\ 1 &: \lambda f. \lambda x. f x \\ 2 &: \lambda f. \lambda x. f (f x) \\ 3 &: \lambda f. \lambda x. f (f (f x)) \\ &\vdots \dots \end{aligned}$$

Define the addition and multiplication operators for the Church numbers with what we introduced in chapter 1.

3. The following defines the Church Boolean values, and the relative logic operators:

true	$\lambda x. \lambda y. x$
false	$\lambda x. \lambda y. y$
and	$\lambda p. \lambda q. p \ q \ p$
or	$\lambda p. \lambda q. p \ p \ q$
not	$\lambda p. p \ \mathbf{false} \ \mathbf{true}$

where **false** is defined as same as the Church number 0. Use the λ conversion rules to prove that: **and true false = false**. Please give the definition of if ... then ... else ... expression with the λ calculus.

2.6 More recursive structures

We've completely defined the recursive functions and the basic pair data structures on top of pure mathematics. We can next define complex data structures like the binary trees.

```
data Tree A = nil | node (Tree A, A, Tree A)
```

This definition says, a binary tree of type A is either empty, or a branch node with three parts: two sub-trees of type A, together with an element of type A. We often call the two sub-trees as the left and right sub-trees. A is the type parameter, like natural numbers for example. $node(nil, 0, node(nil, 1, nil))$ is a binary tree of natural numbers. We can define the abstract fold operation *foldt* for binary trees.

$$\begin{aligned} foldt(f, g, c, nil) &= c \\ foldt(f, g, c, node(l, x, r)) &= g(foldt(f, g, c, l), f(x), foldt(f, g, c, r)) \end{aligned} \tag{2.11}$$

If function f maps variable of type A to B, we write its type as $f : A \rightarrow B$. The Curried function $foldt(f, g, c)$ has type of $foldt(f, g, c) : Tree A \rightarrow B$, where the type of c is B ; the type of g is $g : (B \times B \times B) \rightarrow B$, written in Curried form is $g : B \rightarrow B \rightarrow B \rightarrow B$. We can define the map function *mapt* for binary trees with the *foldt* function.

$$mapt(f) = foldt(f, node, nil) \tag{2.12}$$

With the folding function, we can count the number of elements in a tree:

$$sizet = foldt(one, sum, 0) \tag{2.13}$$

Where, $one(x) = 1$ is a constant function, it always returns 1 for any parameters. sum is a ternary summation function defined as $sum(a, b, c) = a + b + c$.

Using the list defined in chapter 1, we can expand the binary trees to multi-trees as below.

```
data MTree A = nil | node (A, List (MTree A))
```

A multi-tree of type A is either empty, or a composite node, which contains an element of type A, together with multiple sub-trees. The sub-trees are hold in a list. The abstract tree folding operation recursively calls the list folding operation.

$$\begin{aligned} foldm(f, g, c, nil) &= c \\ foldm(f, g, c, node(x, ts)) &= foldr(g(f(x), c), h, ts) \\ h(t, z) &= foldm(f, g, z, t) \end{aligned} \tag{2.14}$$

Exercise 2.3

1. Define the abstract *mapt* for binary trees without of using *foldt*.
2. Define a function *depth*, which counts for the maximum depth of a binary tree.
3. Someone thought the abstract fold operation for binary tree *foldt*, should be defined as the following:

$$\begin{aligned} \text{foldt}(f, g, c, \text{nil}) &= c \\ \text{foldt}(f, g, c, \text{node}(l, x, r)) &= f(\text{foldt}(f, g, g(\text{foldt}(f, g, c, l), f(x)), r), r) \end{aligned}$$

That is to say $g : (B \times B) \rightarrow B$ is a binary operation like add. Can we use this *foldt* to define *mapt*?

4. The binary search tree (BST) is a special tree that the type A is comparable. For any none empty $\text{node}(l, k, r)$, all elements in the left sub-tree l are less than k , and all elements in the right sub-tree r are greater than k . Define function $\text{insert}(x, t) : (A \times \text{Tree } A) \rightarrow \text{Tree } A$ that inserts an element into the tree.
5. Can we define the mapping operation for multi-trees with folding? If not, how should we modify the folding operation?

2.7 The recursive pattern and structure

Recursion exists both in the ancient Euclidean algorithm and in modern computer systems. The fascinating recursive pattern and structure also appear in various of arts in human civilizations. For example in Islamic mosaic arts, shown in figure 2.16.

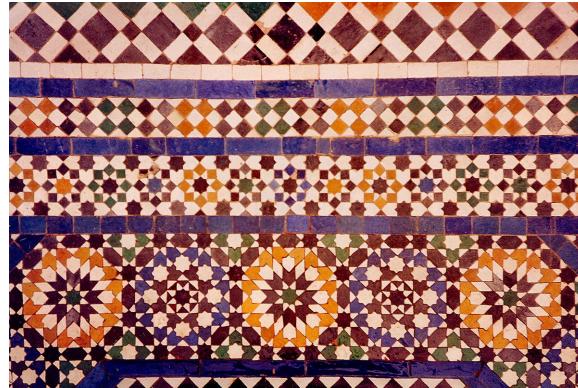
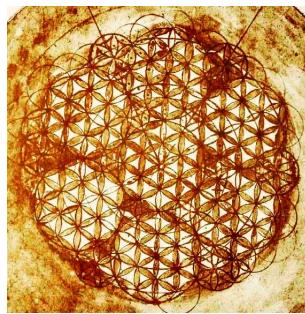


Figure 2.16: Zellige terracotta tiles in Marrakech (a city of the Kingdom of Morocco)

We can see the polygon patterns in the mosaic recursively contain smaller polygons. It demonstrates the beauty of recursive geometric patterns through the colorful tiles. The small patterns form big stripes, which brings the varies of effects. Figure 2.17 is a sketch of the famous Renaissance artist Leonardo da Vinci. It's also a recursive pattern. Using the same radius, he drew six interlaced circles along the centered one. They form a six-lobed snowflake style figure. And they recursively form the same pattern in a bigger scope. The right figure shows a pile of Chinese hand-made katydid cages. It demonstrated the similar recursive pattern. The mesh of the cage is hexagonal. While the overall shape of the cage is also hexagonal when viewed from the axial direction.

Not only in art, recursion also appears in music. For example, the polyphonic music canon and fugue can have recursive musical texture. Canon is a contrapuntal music that



(a) A sketch of Leonardo da Vinci



(b) Chinese katydid cages

Figure 2.17: The recursive pattern in art and artifact.

employs a melody with one or more imitations played after a given duration. The initial melody is called the leader, while the imitative melody, which is played in a different voice, is called the follower. The follower must imitate the leader, either as an exact replication of its rhythms and intervals or some transformation thereof. The weaving of the leader and followers, results in a continuous effect. Each part imitates the theme but contains various changes, such as raising or lowering the pitch, retrograde overlapping, faster (diminution) or slowing (augmentation), melody reflection, and so on.



Figure 2.18: Minuet of Haydn's String Quartet in D Minor, Op. 76, No. 2

A fugue is like a canon, in that it is usually based on one theme which gets played in different voices and different keys, and occasionally at different speeds or upside down or backwards. However, the notion of fugue is much less rigid than that of canon, and consequently it allows for more emotional and artistic expression. The telltale sign of a fugue is the way it begins: with a single voice singing its theme. When it is done, then a second voice enters, either five scale-notes up, or four down. Meanwhile the first voice goes on, singing the “countersubject”: a secondary theme, chosen to provide rhythmic, harmonic, and melodic contrasts to the subject. Each of the voices enters in turn, singing the theme, often to the accompaniment of the countersubject in some other voice, with the remaining voices doing whatever fanciful things entered the composer’s mind. When all the voices have “arrived”, then there are no rules. There are, to be sure, standard kinds of things to do—but not so standard that one can merely compose a fugue by formula. The two fugues in J.S. Bach’s *Musical Offering* are outstanding examples of fugues that could never have been “composed by formula”. There is something much deeper in them than

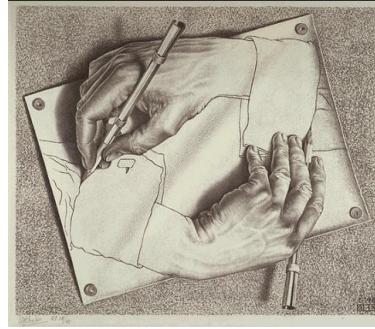


Figure 2.19: M.C. Escher, Drawing Hands, 1948

mere fugality[5].

Not only finite recursion, we can also find the infinite recursion in arts. Figure 2.19, Drawing Hands is a lithograph by the Dutch artist M. C. Escher first printed in January 1948. Two hands mutual recursively are drawing each other. It an example of infinite recursion. The upper hand is using a pencil drawing the lower hand, while the lower hand, at the same time, is drawing the upper hand. The recursion is embedded loop by loop endlessly.

The perfect combination of mathematical recursion and art is fractal. Kock snowflake is a famous fractal curve, which can be generated by infinite recursion rules: For every section, divided it into 3 equal parts, draw a equilateral triangle on top of the middle section, then erase the bottom side of the triangle. Figure 2.20 shows the Kock snowflake result after recursively applying this rule three times on a equilateral triangle. Another famous fractal pattern is called Sierpinski triangle. The generation rule is to connect all the three middle points of the side in a triangle recursively. Below figure shows the Sierpinski triangle after recursively applying the rules four times.

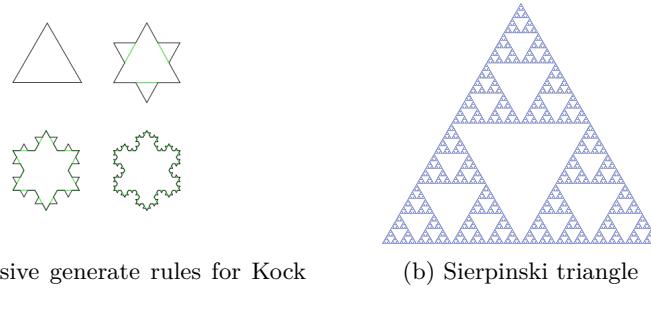
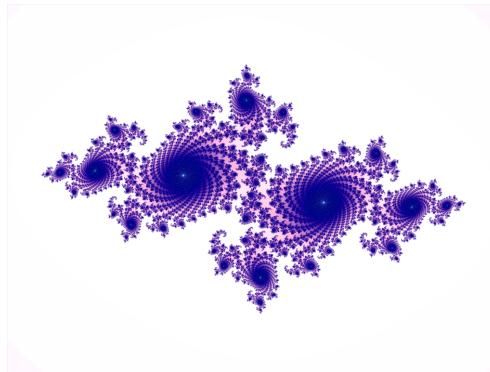


Figure 2.20: Recursive generated fractal patterns.

We show another two fractal patterns as the close of this chapter. One is the fractal in human mind, Julia set; the other is a fractal in nature.

2.8 Further Reading

Mathematics: The Loss of Certainty by Morris Kline contains good introduction about mathematics in ancient Greek. The *Elements* by Euclidean is the most famous classic book. It contains the Euclidean algorithm. *From Mathematics to Generic Programming* by Alexander Stepanov and Daniel E Ross gives varies of implementation of the



(a) Julia set fractal



(b) Broccoli fractal

Euclidean algorithm. As more and more main stream programming environments adopt lambda calculus, there are many materials about it. *The Implementation of Functional Programming Languages* by Simon Peyton Jones is a good book introduced lambda calculus in depth. *Gödel, Escher, Bach: An Eternal Golden Braid* by Douglas Hofstadter intensively presents the unbelievable ideas of recursion and self-reference. It won the Pulitzer Prize for general non-fiction and the National Book Award for Science.

2.9 Appendix: Example program for 2 water jars puzzle

After figure out the integer solutions for 2 water jars puzzle, we can generate the detailed steps, and output them like table (2.2).

```
— Populate the steps
water a b c = if x > 0 then pour a x b y
               else map swap $ pour b y a x
where
  (x, y) = jars a b c

— Pour from a to b, fill a for x times, and empty b for y times.
pour a x b y = steps x y [(0, 0)]
where
  steps 0 0 ps = reverse ps
  steps x y ps@((a', b'):_)
    | a' == 0 = steps (x - 1) y ((a, b'):ps) — fill a
    | b' == b = steps x (y + 1) ((a', 0):ps) — empty b
    | otherwise = steps x y ((max (a' + b' - b) 0,
                                min (a' + b') b):ps) — a to b
```

Run this program, enter `water 9 4 6`, the best pour water steps are output as below:

```
[(0,0),(9,0),(5,4),(5,0),(1,4),(1,0),(0,1),(9,1),(6,4),(6,0)]
```


Chapter 3

Symmetry

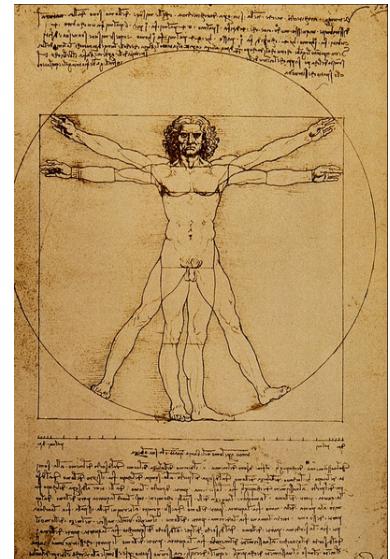
One must be able to say at all times—instead of points, straight lines, and planes—tables, chairs, and beer mugs

—David Hilbert

Symmetry appears everywhere in our world. It is often related to a sense of harmonious, order, pattern, beautiful proportion and balance. We, as human is symmetric, left and right, the sagittal plane divides our body into bilateral halves. *Vitruvian Man* by Leonardo da Vinci is often used as a representation of symmetry in the human body and by extension, the natural universe. The beautiful butterfly, the fish, and birds all represent the symmetry in biology. In our civilization, people created symmetric artifacts, arts, and buildings. The ancient clay jars are rotational symmetric; the Arabic carpets and rags are rectangular symmetric; the fine Chinese style window pane is radial symmetric. The great buildings like Taj Mahal, the forbidden city are reflectional symmetric. We are surprised about the symmetry in snowflakes, every one is unique, but all follow the same hexagon symmetric pattern. When step in the garden in spring, we see all kinds of the beautiful flowers with radial symmetry; while in the woods in autumn, the matured fruits and spikes, the colorful leaves demonstrate us an amazing canvas in the language of symmetry.

In the great world of astronomy, the galaxy rotates its symmetric arms; in the small world of particle, the symmetric crystal lattice reflects light to tell us the mystical nature. A meaningful palindrome poem leads to deep thinking, a random variation music moves our feeling. Our minds contain symmetric things, like connotation and denotation concepts, like abstraction versus material. Mathematics is full of symmetric things in geometry, in algebra, in equations, in curves. Programming is also about symmetry, push in and pop out of a stack, computation scheduling, allocation and release. What is symmetry? How to accurately define, or even measure symmetry?

Mathematician developed group to define, explain, and measure symmetry. And sur-



Leonardo da Vinci, Vitruvian Man, 1490

prisingly, some hard problems, like the solvability of equation is ultimately solved by revealing the symmetry of roots. Other famous problems, like the three classic geometric problems in ancient Greece are also solved by this. To uncover the secret of symmetry, we are going to follow the path in this chapter to the abstract algebra world of groups, rings, and fields.

Humans gradually developed the habit to sort things. We classify similar things together. The methods and properties those apply to the entire class are also valid for every thing in it. In this way, we needn't repeatedly solve the concrete individual problems one by one, but solve the abstract problem as a whole. It greatly improved our ability to understand the world.

In previous chapters, we generalized the abstract ‘folding’ operation from sum and factorial for numbers. We observed their similar structures, abstracted zero in sum, and one in factorial to unit element; then abstracted add and multiplication to binary operation. As the result, we developed the fold operation for numbers in a higher level. With this powerful tool, we then further solved a large sort of problems that are isomorphic to natural number, such as the Fibonacci numbers.

For another example, we defined the abstract *foldr* operation for list in chapter 1. With this tool, we can sum a list of numbers as $sum = foldr(0, +)$; we can also multiply them as $product = foldr(1, \times)$. In computer programming, there is a data structure called ‘binary search tree’. We introduced about it in chapter 2. Binary search tree is a special binary tree. Its elements are comparable¹. For any branch node, all elements in its left sub-trees are ahead of the element in this node; while all elements in its right sub-trees are behind it. Due to this kind of ordering, we also call it ‘sorted binary tree’. We can define insert operation for binary search tree as below:

$$\begin{aligned} insert(nil, x) &= node(nil, x, nil) \\ insert(node(l, y, r), x) &= \begin{cases} x < y : node(insert(l, x), y, r) \\ x > y : node(l, y, insert(r, x)) \end{cases} \end{aligned}$$

According to this definition, when insert element x to binary search tree, if the tree is empty, the result is $node(nil, x, nil)$; otherwise, we compare x with the element y in the branch node. If x is ahead of y (the ‘ $<$ ’ holds), then we recursively insert it to the left sub-tree, else insert to the right sub-tree. Is there any similarity among the insertion, sum, and factorial? Insertion is also a binary operation, nil can be considered as the unit element. Then we can apply the abstract fold operation to turn a list of elements into a binary search tree:

$$build = foldr(nil, insert)$$

Figure 3.2 shows the binary search tree generated when compute $build [4, 3, 1, 8, 2, 16, 10, 7, 14, 9]$.

We have similar experience when developing the concept of add. The addition operation was applied to specific things at first, like the fruits being collected, the prey being hunted. People later abstracted the addition for numbers and removed the specific meanings for things. Then we extended our understanding about numbers from integers to fractions, although the detailed addition process is quite different. We need firstly unify the denominators, then add the numerators together, and finally reduce the fraction. People generalized the two different processes to the unified addition for rational numbers. We learned to think about the essence and principles of addition. Every time when people extended the concept of numbers, there was a new definition for addition. Along this way, we defined the addition for real numbers and complex numbers. We

¹The meaning of comparable is abstract. If the elements are numbers, we can compare which one is bigger, if they are words, we can compare their lexicographical order.

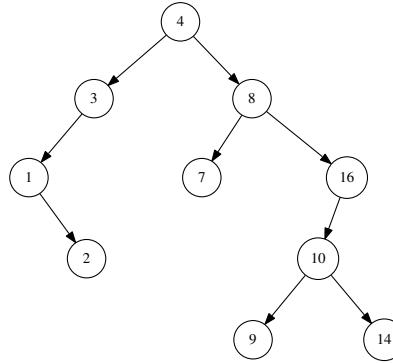


Figure 3.2: The binary search tree generated from folding.

found the method established for abstract things without specific meanings had a greater scope of application. Abstract method can solve a whole kind of problems rather than the individual ones. Similar things happened in computer engineering. People developed the object-oriented method, generic type system, and dynamic type system. All these are sorts of mechanisms to support abstraction.

We should always ask an important question when developing abstract tools or generic methods. ‘What is the applicable scope for this abstraction? When will the abstraction be invalid?’ It could lead to ridiculous result if ignore this. One example is about the sum of the infinite geometric series $1 + x + x^2 + x^3 + \dots = 1/(1 - x)$. It is so powerful that people can solved the Zeno’s paradox² with it. The mathematicians in the 17 Century substituted x with -1, then got a result of $1 - 1 + 1 - 1 + \dots = 1/2$. While, someone had a different idea that, $S = (1 - 1) + (1 - 1) + \dots = 0$. And there was another different one: $S = 1 + (-1 + 1) + (-1 + 1) + \dots = 1$. There were people supported the result should be $1/2$, because $S = 1 - (1 - 1 + 1 - 1 + \dots) = 1 - S$, solving this equation gave $S = 1/2$. The Italian mathematician Grandi (1671 - 1742) found more surprising results. By using the infinite series:

$$\frac{1}{1 + x + x^2} = 1 - x + x^3 - x^4 + x^6 - x^7 + \dots$$

$$\frac{1}{1 + x + x^2 + x^3} = 1 - x + x^4 - x^5 + x^8 - x^9 + \dots$$

...

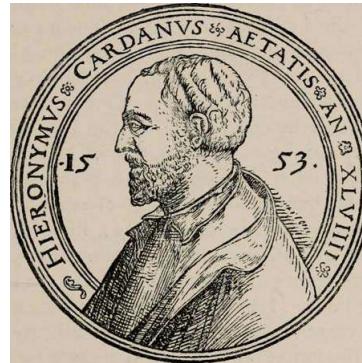
Let $x = 1$, Grandi found the sum of the infinite series $1 - 1 + 1 - 1 + 1 - 1 + \dots$ could be $1/3, 1/4, \dots$ Even the great mathematician Leibniz thought the result could be 0 or 1 with the same probability, therefore the ‘true’ value should be the average $1/2$. Grandi offered a new explanation in 1710: Two brothers inherited a priceless gem from their father, who forbade them to sell it, so they agreed that it would reside in each

²Here it is the Achilles and turtle paradox, which is one of the four famous paradoxes by Zeno, the ancient Greek philosopher. Achilles was a hero in ancient Greek. Zeno supposed Achilles wanted to catch up with the turtle ahead. When he ran to the position where the tortoise left, the tortoise had moved a short distance forward. Achilles must continue to run to that new position, but the turtle moved forward again. Repeated this process, Zeno argued that Achilles would never catch up the turtle. We’ll explain Zeno’s paradox in detail in Chapter 5.

other's museums on alternating years. If this agreement last for all eternity between the brother's descendants, then the two families would each have half possession of the gem, even though it changed hands infinitely often[8]. People kept suffering from these strange puzzles until the French mathematician Cauchy introduced the convergence concept for infinite series.

This chapter introduces the basic abstract algebra structures. They are not only abstraction to numbers, but also the abstraction to concepts, properties, and relationships. They are the most valuable things from many great thoughts and minds. Some contents challenge our limit of abstract thinking. It's quite common that you can't grasp them during the first time reading. I intended to add many stories about those great mathematicians, how they made breakthrough with unbelievable difficulties. I hope these interesting stories could encourage you keep going forward.

3.1 Group



Gerolamo Cardano, 1501-1576

The group theory is originate from the history of equations. Equation is a powerful tool developed in ancient time. From Rhind Mathematical Papyrus and Babylonian clay tablets, we know that the ancient Egyptians and Babylonians mastered the method to solve the linear and quadratic equations with one unknown. However, people didn't find the way to solve the generic cubic equations until the 16th Century. Several Italian mathematicians made great progress. Gerolamo Cardano finally published the radical solutions to generic cubic equation and quartic in his 1545 book *Ars Magna*. It was not only about to pursue higher and higher orders, but also along with the totally new understanding to numbers in the past thousand years. All the negative roots were discarded because people in that time believed they were meaningless. People also thought the coefficients must be positive numbers. The equation $x^2 - 7x + 8 = 0$ is quite common today to us, but it had to be written in form of $x^2 + 8 = 7x$ to ensure the coefficients are positive. After Cardano list 20 different types of quartic equation in *Ars Magna*, he said there were another 67 types of quartic equation could not be given because the coefficient is either negative or zero[18]. It was the French mathematician François Viète who unified different forms of equations. Although most people thought the negative square root made no sense, Cardano found an interesting thing when solve the cubic equations like $x^3 = 15x + 4$. His formula gives the intermediate result of $\sqrt[3]{2 + \sqrt{-121}} + \sqrt[3]{2 - \sqrt{-121}}$. Then it could next generate the three rational roots of $4, -2 \pm \sqrt{3}$. Such problems expand our view to the irrational number, and finally, the great German mathematician Carl Friedrich Gauss

developed the fundamental theorem of algebra³.



Gauss (1777-1855) in 10 Mark

People encountered surprisingly difficulty when seek for radical solutions for generic quintic and higher order equations in the next 300 years. The breakthrough happened in the 19th Century with unexpected result. The French young Genius Évariste Galois developed an innovative idea, he was able to determine a necessary and sufficient condition for a polynomial to be solvable by radicals while still in his teens⁴.



Galois, 1811 - 1832

It was a tragedy in Galois' short 20 years life, but his work laid the foundations of abstract algebra. Galois was born on October 25th, 1811 in Paris. His mother, the daughter of a jurist, was a fluent reader of Latin and classical literature and was responsible for her son's education for his first twelve years. In 1823, he entered a prestigious school in Paris. At the age of 14, he began to take a serious interest in mathematics. He found a copy of *Elements* adapted by Legendre which, it is said, he read "like a novel" and mastered at the first reading. At 15, he was reading the original papers of Joseph-Louis

³Gauss proved the fundamental theorem of algebra several times along his life. in 1799 at age of 22, he proved in his doctor thesis that every single-variable polynomial of degree n with real coefficient has at least one complex root, thus deduced the single-variable equation of degree n has and only has n complex roots (counted with multiplicity for the same ones). Gauss gave another two different proofs in 1815 and 1816. In 1849, to celebrate the 50th anniversaries that Gauss received his doctor degree, he published the fourth proof and extend the coefficient to complex number.

⁴Around 1770, Joseph Louis Lagrange began the groundwork that unified the method to solve equations, he introduced the new idea to permute roots in the form of Lagrange resolvents. But he didn't consider the combination among the permutations. In 1799 The Italian mathematician Paolo Ruffini marked a major improvement, developing Lagrange's work on permutation theory. However, in general, Ruffini's proof was not considered convincing, and was discovered later incomplete. In 1824, the young Norwegian mathematician Niels Henrik Abel first completed proof demonstrating the impossibility of solving the general quintic equation in radicals. It is called 'Abel-Ruffini theorem' nowadays. We know that there are radical solutions to the special quintic equation $x^5 - 1 = 0$. In what condition a polynomial is solvable in radicals? This problem was completely solved by Galois[19].

Lagrange, which likely motivated his later work on equation theory, yet his classwork remained uninspired, and his teachers accused him of affecting ambition and originality in a negative way.

In 1828, he attempted the entrance examination for the École Polytechnique, the most prestigious institution for mathematics in France at the time, without the usual preparation in mathematics, and failed for lack of explanations on the oral examination. In that same year, he entered the École Normale, a far inferior institution for mathematical studies at that time, where he found some professors sympathetic to him.

In 1829 April, Galois' first paper, on continued fractions, was published. It was around the same time that he began making fundamental discoveries in the theory of polynomial equations. He submitted two papers on this topic to the Academy of Sciences. But both were rejected due to some reasons⁵

On July 28, 1829, Galois' father, a mayor of the village, committed suicide after a bitter political dispute with the village priest[20]. On August 3, Galois made his second and last attempt to enter the Polytechnique, and failed yet again. It is undisputed that Galois was more than qualified; however, accounts differ on why he failed. More plausible accounts state that Galois made too many logical leaps and baffled the incompetent examiner, which enraged Galois. Having been denied admission to the Polytechnique, Galois took the Baccalaureate examinations in order to enter the École Normale. He passed. His examiner in mathematics reported, “This pupil is sometimes obscure in expressing his ideas, but he is intelligent and shows a remarkable spirit of research.”



Eugène Delacroix, Liberty Leading the People, 1830, Louvre, Paris

In February 1830, following Cauchy's suggestion Galois submitted his memoir on equation theory to the Academy's secretary Joseph Fourier, to be considered for the Grand Prize of the Academy. Unfortunately, Fourier died soon after, and the memoir was lost. The prize was awarded that year in June to Niels Henrik Abel⁶ posthumously and also

⁵There was saying that the paper was lost by Cauchy. Actually, Cauchy refereed these papers, but refused to accept them for publication for reasons that still remain unclear. However, in spite of many claims to the contrary, it is widely held that Cauchy recognized the importance of Galois' work, and that he merely suggested combining the two papers into one in order to enter it in the competition for the Academy's Grand Prize in Mathematics. Cauchy, an eminent mathematician of the time, though with political views that were at the opposite end from Galois', considered Galois' work to be a likely winner[20].

⁶The young Norwegian mathematician Abel died on April 6, 1829. He made pioneering contributions in

to Carl Gustav Jacob Jacobi[12].

Galois lived during a time of political turmoil in France. The July Revolution broke out in France in 1830⁷. While their counterparts at the Polytechnique were making history in the streets, Galois and all the other students at the École Normale were locked in by the school's director. Galois was incensed and wrote a blistering letter criticizing the director, which he submitted to the *Gazette des Écoles*, signing the letter with his full name. Although the *Gazette*'s editor omitted the signature for publication, Galois was expelled.

Galois quit school and joined the staunchly Republican artillery unit of the National Guard. He divided his time between his mathematical work and his political affiliations. Due to controversy surrounding the unit, soon after Galois became a member, on December 31, 1830, the artillery of the National Guard was disbanded out of fear that they might destabilize the government. He was arrested the on May 10, 1831, but was acquitted on June 15. On the Bastille Day (July 14), Galois was at the head of a protest, wearing the uniform of the disbanded artillery, and came heavily armed with several pistols, a rifle, and a dagger. He was again arrested. On October 23, he was sentenced to six months in prison for illegally wearing a uniform. Early in 1831, Siméon Poisson asked him to submit his work on the theory of equations, which he did on January 17, 1831. Around July 4, 1831, Poisson declared Galois' work "incomprehensible", declaring that "The argument is neither sufficiently clear nor sufficiently developed to allow us to judge its rigor⁸". While Poisson's report was made before Galois' July 14 arrest, it took until October to reach Galois in prison. It is unsurprising, in the light of his character and situation at the time, that Galois reacted violently to the rejection letter, and decided to abandon publishing his papers through the Academy and instead publish them privately through his friend Auguste Chevalier. Apparently, however, Galois did not ignore Poisson's advice, as he began collecting all his mathematical manuscripts while still in prison, and continued polishing his ideas until his release on April 29, 1832.

Shortly after released from prison, Galois was involved in a obscure duel because of love. On May 29, Galois was so convinced of his impending death that he stayed up all night writing letters to his friends and composing what would become his mathematical testament, the famous letter to Auguste Chevalier outlining his ideas, and three attached manuscripts. German mathematician Hermann Weyl said of this testament, "This letter, if judged by the novelty and profundity of ideas it contains, is perhaps the most substantial piece of writing in the whole literature of mankind." In these final papers, he outlined the rough edges of some work he had been doing in analysis and annotated a copy of the manuscript submitted to the Academy and other papers.

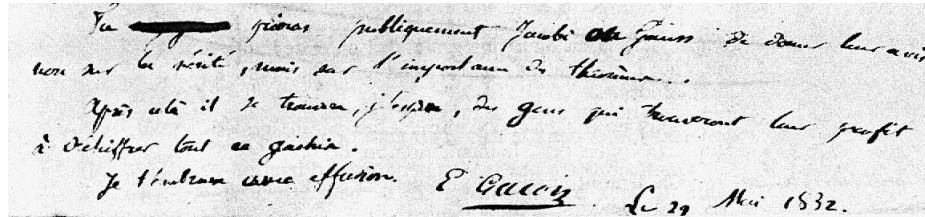
When read Galois' 7 pages testament, there are some words really sad: "these subjects are not the only ones that I have explored... But I don't have time, and my ideas are not yet well developed in this area, which is immense...it would not be too much in my interest to make mistakes so that one suspects me of having announced theorems of which I would not have a complete proof." The most impressed and saddest words are: "I don't

a variety of fields. His most famous single result is the first complete proof demonstrating the impossibility of solving the general quintic equation in radicals. He was also an innovator in the field of elliptic functions, discoverer of Abelian functions. He made his discoveries while living in poverty and died at the age of 26 from tuberculosis.

⁷The Bourbon monarch was restored after Napoleon's defeat in Waterloo. King Charles X cleaned the soldiers who had worked for Napoleon in the army and caused dissatisfaction among the people. Continues with a series of failures in politics, economy, culture, religion, and diplomacy, Charles X signed the July Ordinances on July 25, 1830. These, among other steps, suspended the liberty of the press, dissolved the newly elected Chamber of Deputies, and excluded the commercial middle-class from future elections. It triggered the armed uprising of people, overthrew the Bourbon monarch through revolution. It ended with Louis-Philippe becoming king. Known as the July Monarchy.

⁸However, the rejection report ends on an encouraging note: "We would then suggest that the author should publish the whole of his work in order to form a definitive opinion." [20]

have time" At the end of the letter, he asked his friend to "Ask Jacobi or Gauss publicly to give their opinion, not as to the truth, but as to the importance of these theorems. Later there will be, I hope, some people who will find it to their advantage to decipher all this mess."^[21]



"E. Galois, le 29 mai 1832" at the bottom of the last page in his testament.

Early in the morning of May 30, 1832, Galois was injured badly in the duel. He was shot in the abdomen. A passing farmer found him, and sent Galois to the hospital. He died the following morning at ten o'clock, after refusing the offices of a priest. His younger brother was notified, and came to the hospital. His last words to his brother Alfred were: "Don't cry, Alfred! I need all my courage to die at twenty!" We don't know the exact reason behind the duel, whether it was a love tragedy or a political murder. Whatever the reason, a great talent mathematician was killed at the age of 20. He had only been studied mathematics for 5 years. Within the only 67 pages of Galois' collected works are many important ideas that have had far-reaching consequences for nearly all branches of mathematics.

Chevalier and Galois' young brother published the testament in *Revue encyclopédique*, but it was not noticed. It might be too brief and hard, there was almost no any impact to the mathematics in that years⁹. Decades passed, in 1843 Liouville reviewed Galois manuscript and declared it sound. It was finally published in the October–November 1846 issue of the *Journal de Mathématiques Pures et Appliquées*. The most famous contribution of this manuscript was a novel proof that there is no quintic formula –that is, that fifth and higher degree equations are not generally solvable by radicals. Although Abel had already proved the impossibility of a "quintic formula" by radicals in 1824, Galois' methods led to deeper research in what is now called Galois theory. For example, one can use it to determine, for any polynomial equation, whether it has a solution by radicals. Liouville thought about this tragedy and commented in the introduction to Galois' paper: "Perhaps, his exaggerated desire for conciseness was the cause of this defect, and is something which one must endeavor to refrain from when dealing with the abstract and mysterious matters of pure Algebra. Clarity is, indeed, all the more necessary when one has intention of leading the reader away from the beaten roads into the desert... But at present all that has changed. Alas, Galois is no more! Let us cease carrying on with useless criticisms; let us leave its defects, and instead see its qualities... My zeal was soon rewarded. I experienced great pleasure the moment when, after having filled in the minor gaps, I recognized both the scope and precision of the method that Galois proved."^[22]

In 1870, French mathematician Camille Jordan wrote the book *Traité des substitutions et des équations algébriques* based on Galois' theory¹⁰. Galois' most significant contribu-

⁹The similar lessons happened to Abel as well. When he spent his own money to publish the paper about why quintic equation couldn't be solved by radicals in 1824, to save money, he tried all means to consolidate the paper in 6 pages. As the result, it's too brief and obscure for people to notice and understand it till Abel's death.

¹⁰Galois' theory was notoriously difficult for his contemporaries to understand, especially to the level where they could expand on it. For example, in his 1846 commentary, Liouville completely missed the

tion to mathematics is his development of Galois theory. He realized that the algebraic solution to a polynomial equation is related to the structure of a group of permutations associated with the roots of the polynomial, the Galois group of the polynomial. It laid the foundation of group theory and lead to the development of abstract algebra and modern mathematics. As the ironic result “Instead of the political revolution, what Galois actually triggered was the mathematics revolution[10].”

3.1.1 Group

Let us start the journey from group to understand what Galois landed for abstract algebra.

Definition 3.1.1. *A group is a set G equipped with a binary operation “ \cdot ”, which satisfied four axioms:*

1. **Closure:** For all $a, b \in G$, the result of the operation $a \cdot b \in G$;
2. **Associativity:** For all a, b, c in G , $(a \cdot b) \cdot c = a \cdot (b \cdot c)$;
3. **Identity element:** There exists an element e in G such that, for every element a in G , the equation $a \cdot e = e \cdot a = a$ holds;
4. **Inverse element:** For each element $a \in G$, there exists an element a^{-1} , such that $a \cdot a^{-1} = a^{-1} \cdot a = e$, where e is the identity element.

The binary operation is often called “multiplication”, and the “product” $a \cdot b$ is usually written as ab . e is the identity element. The elements in a group can be finite or infinite many, thus the group is called finite group or infinite group. the **order** of a finite group is the number of the elements in that group. A group contains infinite many elements is said to have infinite order.

The “multiplication” operation of the group may not be commutative like the normal multiplication for numbers. For example, all the invertible matrix with real entities, together with the matrix multiplication form a group. However, the matrix multiplication order matters, it is not commutative. Groups for which the community equation $ab = ba$ always holds are called **abelian** groups (in honor of Abel).

let us see some examples to understand the definition of groups.

1. Integers with addition. Elements are all integers, the binary operation is addition. This is one of the most familiar groups;
2. The set of remainders of all integers divided by 5, that is $\{0, 1, 2, 3, 4\}$. The binary operation is addition then divided by 5, and take the remainder. For example $3 + 4 = 7 \bmod 5 = 2$. They form a group called addition group of integers modulo 5. Denoted as Z_5 . We can consider it as partition the integers by taking the remainder¹¹. It is called **residue class** or residue modulo n ;
3. The rotations of Rubik cube form a group. The elements are all cube rotations¹², the binary operation is the composition of cube rotates, corresponding to the result of performing one cube rotate after another.

group-theoretic core of Galois’ method. Joseph Alfred Serret who attended some of Liouville’s talks, included Galois’ theory in his 1866 textbook (third edition) *Cours d’algèbre supérieure*. Jordan was Serret’s pupil. Outside France, Galois’ theory remained more obscure for a longer period. It turned a century in Britain. In Germany, it was Dedekind lectured on Galois’ theory at Göttingen in 1858, showing a very good understanding.

¹¹It is denoted as $Z/5Z$ nowadays.

¹²There are 18 Rubik cube rotations. A cube move rotates one of the 6 faces, front, back, top, bottom, left, right, 90° , 180° , or -90° . For example, rotating the left side 90° , 180° , -90° can be denoted as L , L^2 , L' [23]. Plus the identity transform, there are total 19 elements.

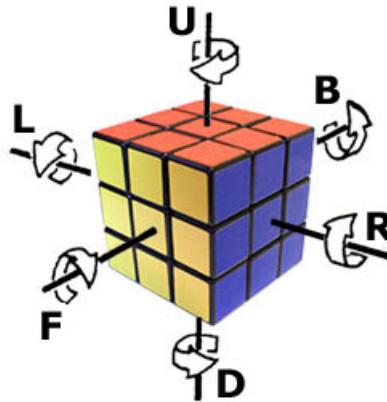


Figure 3.8: There are total 18 Rubik cube rotates for the 6 faces, plus the identity transformation. These moves together with the composition operation form a group.

It's helpful to think about symmetry through the Rubik cube group. In original state, all 6 sides have the same color for each. Let a kid play it with a series of random rotations. The player need to figure out a way, restore the Rubik cube to its original state, the all sides are back to the unified color. If we record the kid's rotations with a camera as $\{t_0, t_1, \dots, t_m\}$, and the restore process as $\{r_0, r_1, \dots, r_n\}$. Then the whole process to solve the Rubik cube puzzle means:

$$(r_n \cdot r_{n-1} \dots \cdot r_0) \cdot (t_m \cdot t_{m-1} \dots \cdot t_0) = e$$

Obviously, one solution is to reverse every rotation made by the kid, that is $r_i = t_{m-i}^{-1}$, or $r_i \cdot t_{m-i} = e$. Hence the above equation must hold. However, in practice, a seasoned player restores the Rubik cube through a set of 'formulas'. Although the above equation holds, not every r_i is the reversed rotation of some t_{m-i} . Even the number of steps to restore may not equal to the number of steps to disrupt the Rubik cube.

4. For a plane, all the rotations around a fixed point form a group. The group elements are all the degrees for rotation. The group binary operation is the composition, corresponding to the result of rotating a degree after another degree. The unit rotation is zero degree.

We say a shape has rotational symmetry if it can be rotated about a fixed point, and still looks same. A snowflake for example, keeps same by rotating 60° , 120° , 180° , 240° , and 360° . However, the rotational group in this example can not define the symmetry of snowflakes. It actually tells such a fact. If two rotations satisfy $r_\alpha \cdot r_\beta = e$, then the two degrees either negate to each other $\alpha = -\beta$, or their sum are multiples of 360° . The only symmetric shapes under this rotational group are circles at the same centre. We'll give the group that defines the snowflake symmetry in later sections.

While the following examples are not groups:

1. All the integers except 0, together with multiplication do not form a group. We can't use 1 as the identity element, otherwise there will be no inverse element for 3 for example ($1/3$ is not an integer);

2. For the same reason, remainders of modulo 5 $\{0, 1, 2, 3, 4\}$, together with multiplication modulo 5 do not form a group. However, when exclude all multiples of 5, then the set $\{1, 2, 3, 4\}$ and multiplication modulo 5 form a group. We can see this fact from the following “multiplication table” modulo 5:

	1	2	3	4
1	1	2	3	4
2	2	4	1	3
3	3	1	4	2
4	4	3	2	1

Therefore, 1 is the identity element, it is also the inverse element of itself; 2 and 3 are inverse elements for each other; the inverse element for 4 is 4 again;

3. Although all the none zero remainders modulo 5 form a group under modulo multiplication, the none zero remainders modulo 4 do not form a group. Observe the multiplication table modulo 4:

	1	2	3
1	1	2	3
2	2	0	2
3	3	2	1

Note that $(2 \times 2) \bmod 4 = 0$, which is not in set $\{1, 2, 3\}$. This negative example shows that, only the remainders that are coprime to n form a group under the multiplication modulo n . This kind of groups are called multiplicative group of integers modulo n . For any prime number p , set $\{1, 2, \dots, p-1\}$ forms a multiplicative group modulo p .

4. All the rational numbers together with multiplication do not form a group. Although all rational number with form p/q has a inverse element q/p , but there is no inverse element for 0. All the rational number exclude 0 form a group under multiplication.

Exercise 3.1

1. Do all the even numbers form a group under addition?
2. Can we find a subset of integers, that can form a group under multiplication?
3. Do all the positive real numbers form a group under multiplication?
4. Do integers form a group under subtraction?
5. Find an example of group with only two elements.
6. What is the identity element for Rubik cube group? What is the inverse element for F ?

3.1.2 Monoid and semi-group

The criteria to be a group is a bit strict. From the negative examples in previous section, we see some common algebraic structures can't satisfy all the group axioms. Sometimes we needn't inverse element. If relax the limitation, we get the **monoid** structure.

Definition 3.1.2. A **monoid** is a set S together with a binary operation ‘ \cdot ’, which satisfies two axioms:

1. **Associativity:** For any three elements in S , the equation $(a \cdot b) \cdot c = a \cdot (b \cdot c)$ holds.
2. **Identity element:** There exists an element e in S , such that for every element a in S , the equation $a \cdot e = e \cdot a = e$ holds.

The monoid definition is quite similar to group except there is no axiom about inverse element. Many negative examples for group are monoids. For example, integers under multiplication form a monoid. The identity element is 1. Monoid often appears in computer programming. We'll revisit this important algebraic structure in next chapter about category theory. Here are some more examples about monoid:

1. Given a character set, all finite strings with the concatenation operation form a monoid. The elements are strings; the binary operation is string concatenation; the identity element is the empty string.
2. Expand from string to list of type A (List A). All lists form a monoid under the concatenation operation. The monoid elements are lists; the binary operation is the list concatenation (denoted as ++); the identity element is the empty list nil .

By using monoid as the algebraic structure, we can abstract both string and list as the below example program.

```
instance Monoid (List A) where
  e = nil
  (*) = (++)
```

The folding operation to string and list, can be abstracted to the level of monoid as well¹³. The below concatenate operation for example, is defined to any monoid:

$$\text{concat} = \text{foldr } e \text{ } (*)$$

We can concatenate list with this *concat* operation like this:

concat $[[1], [2], [3], [1, 2], [1, 3], [2, 3], [1, 2, 3], [1, 3, 2]]$ gives result: $[1, 2, 3, 1, 2, 1, 3, 2, 3, 1, 2, 3, 1, 3, 2]$.

3. Heap is a common data structure in programming. If the top element in the heap is always the minimum one, it's call the min-heap; If the top element is always the maximum one, it's called max-heap. Skew heap is a type of heap realized in binary tree ([14], section 7.3).

```
data SHeap A = nil | node (SHeap A, A, SHeap A)
```

The definition is as same as binary tree except for its name. The minimum element is always located at the root for the none empty heap. We define the merge operation for two heaps as below:

$$\begin{aligned} \text{merge}(\text{nil}, h) &= h \\ \text{merge}(h, \text{nil}) &= h \\ \text{merge}(h_1, h_2) &= \begin{cases} k_1 < k_2 : & \text{node}(\text{merge}(r_1, h_2), k_1, l_1) \\ \text{otherwise} : & \text{node}(\text{merge}(h_1, r_2), k_2, l_2) \end{cases} \end{aligned}$$

¹³We'll introduce how to realize the abstract folding in next chapter about category theory

When merge two heaps, if one is empty, the result is the other one; if neither one is empty, we denote h_1, h_2 as $node(l_1, k_1, r_1)$ and $node(l_2, k_2, r_2)$ respectively. To merge them, we firstly compare their root, select the smaller one as the new root; then merge the other heap with the bigger element to one of its sub-trees. Finally, we exchange the left and right sub-trees. For example, if $k_1 < k_2$, we select k_1 as the new root. Then we can either merge h_2 to l_1 , or merge h_2 to r_1 . Without loss of generality, we merge to r_1 . Then, we exchange the left and right sub-trees to get the final result $(merge(r_1, h_2), k_1, l_1)$. Note the binary merge operation is recursive. The set of all the skew heaps, together with the binary merge operation form a monoid. The identity element is the empty heap nil.

4. Heap can also be realized in multi-trees as explained in previous chapter, for example pairing heap is defined as the following ([14] section 9.4):

```
data PHeap A = nil | node (A, List (PHeap A))
```

The definition is as same as multi-tree except for its name. It is a recursive definition. A pairing heap is either empty; or a multi-tree with a root and a list of sub-trees. The minimum element is located at the root for a none empty heap. We define the merge operation for pairing heap as below:

$$\begin{aligned} merge(nil, h) &= h \\ merge(h, nil) &= h \\ merge(h_1, h_2) &= \begin{cases} k_1 < k_2 : & node(k_1, h_2 : ts_1)) \\ \text{otherwise :} & node(k_2, h_1 : ts_2)) \end{cases} \end{aligned}$$

If either heap is empty, then the merge result is the other heap. Otherwise if neither heap is empty, we represent the two heaps h_1, h_2 as $node(k_1, ts_1)$ and $node(k_2, ts_2)$ respectively. We compare the root elements, let the bigger one be another new sub-tree of the other. The set of all the pairing heaps form a monoid under the merge operation. The identity element is the empty heap nil.

If relax the limitation one more step, to remove the requirement of identity element, then we get another algebraic structure, semigroup.

Definition 3.1.3. *A semigroup is a set together with the associative binary operation.*

The binary operation for semigroup is associative. It means for any three elements a , b , and c the equation $(ab)c = a(bc)$ holds. The constraints for semigroup is relaxed one more step from monoid. Here are some semigroup examples:

1. All the positive integers form a semigroup under addition, as well as under multiplication;
2. All the even numbers together with addition form a semigroup, so as under multiplication.

As we mentioned before, people often call the binary operation for group, monoid, and semigroup as ‘multiplication’, therefore we use the term ‘power’ to represent applying the binary operation multiple times, for example: $x \cdot x \cdot x = x^3$. Generally, the ‘power’ of group and monoid is defined as below recursively:

$$x^n = \begin{cases} n = 0 : & e \\ \text{otherwise :} & x \cdot x^{n-1} \end{cases}$$

For semigroup, since the identity element is not defined, n must be none zero positive integer:

$$x^n = \begin{cases} n = 1 : & x \\ \text{otherwise :} & x \cdot x^{n-1} \end{cases}$$

Exercise 3.2

1. The set of Boolean values {True, False} forms a monoid under the logic or operator \vee . It is called ‘Any’ logic monoid. What is the identity element for this monoid?
2. The set of Boolean values {True, False} forms a monoid under the logic and operator \wedge . It is called ‘All’ logic monoid. What is the identity element for this monoid?
3. For the comparable type, when compare two elements, there can be three different results. We abstract them as $\{<, =, >\}$ ¹⁴. For this set, we can define a binary operation to make it a monoid. What is the identity element for this monoid?
4. Prove that the power operation for group, monoid, and semigroup is commutative: $x^m x^n = x^n x^m$

3.1.3 Properties of group

One powerful idea in abstract algebra is that, we can focus on the inner pattern of the abstract structures and their relations without caring about the concrete object and its meaning. The pattern and the revealed insight are applicable to all objects by the nature of abstraction. When know the generic group properties, if the elements represent points, lines, and surfaces, then we obtain the properties of geometry; if the elements represent Rubik cube rotations, then we obtain the properties of Rubik cube transformation; if the elements represent some data structure in programming, we obtain the properties of the algorithm on top of that data structure. We introduce some important group properties in this section.

Theorem 3.1.1. *There is one and only one identity element for any group.*

Proof. Suppose there is another identity element e' , for all element a , the equation $e'a = ae' = a$ holds. Substitute a with e , we have $e = ee' = e'$. Hence proved the uniqueness of the identity element. \square

For any group, not only the identity element, but also the inverse element for every element is unique.

Theorem 3.1.2. *The unique existence of inverse element. For all element a , there is one and only one a^{-1} that satisfies $aa^{-1} = a^{-1}a = e$. We call a^{-1} the inverse element of a .*

Proof. We know the existence of inverse element from the group identity element axiom. Therefore, we only need proof the uniqueness. Suppose there exists another element b , that also satisfies $ab = ba = e$. We multiply a^{-1} to the equation from right to get:

$$\begin{aligned} aba^{-1} &= baa^{-1} = ea^{-1} \\ \Rightarrow be &= a^{-1} && \text{Apply associative law to the 2nd term} \\ \Rightarrow b &= a^{-1} && \text{Uniqueness of the inverse element} \end{aligned}$$

\square

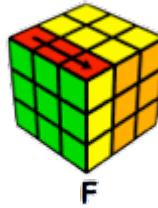


Figure 3.9: Repeatedly rotating a Rubik cube with F 4 times returns to the original state.

We defined the order of group before. For group element, we can define order as well. For element a , the minimum positive integer m that satisfies $a^m = e$ is called the order of a . If such m does not exist, we say the order of a is infinite. Using the Rubik cube group for example, if we repeat F rotation 4 times, the cube returns to its original state. Therefore, the order of F is 4. Because rotating F' twice returns the original state, the order of F' is 2. Another example is the integer multiplicative group modulo 5. For all elements except 1, the 4th power modulo 5 are 1. All their orders are 4. Actually, we have the following interesting theorem:

Theorem 3.1.3. *For any finite group, all elements have finite order.*

Proof. Denote the order of a given finite group G as n . For any element a , we can construct a set $\{a, a^2, \dots, a^{n+1}\}$. There are $n + 1$ elements in this set, however, the order of the group is n . According to the principle of pigeon hole, there are at least two equal elements. Denote such two equal elements as a^i and a^j , where $0 < i < j \leq n + 1$ without loss of generality. We have:

$$\begin{aligned} a^j a^{-i} &= a^i a^{-i} && \text{since } a^i = a^j \\ a^j a^{-i} &= e && a^i \text{ is inverse to } a^{-i} \\ a^{j-i} &= e && \text{the order of } a \text{ is } j - i \end{aligned}$$

Thus the order of $a, j - i$ is finite. □

We use the term ‘isomorphism’ in chapter 1 to describe things that have the same inner structure. It’s time to give the strict definition for homomorphism and isomorphism. Suppose there is a mapping (morphism) f from set A to set B . a and b are two elements in A , their images in B are $f(a)$ and $f(b)$ respectively. Let’s consider the element $a \cdot b$, which is result of the binary closed operation defined in A . Under the map (morphism) f , its image in B is $f(a \cdot b)$. If for the binary closed operation defined in B , the following equation always holds:

$$f(a) \cdot f(b) = f(a \cdot b)$$

We say f is a **homomorphism** from A to B . If f is a surjection, known as ‘onto’, which means every element b in B , has the corresponding a in A , that $f(a) = b$, then f is called surjective homomorphism. For example, consider a test function $odd : \mathbb{Z} \rightarrow \text{Bool}$, it accepts an integer number, if the number is odd, then it returns True, otherwise returns False. All integers form a group under addition, while the Boolean value set $\{\text{True}, \text{False}\}$ also forms a group under logic exclusive or operation. We can verify that:

¹⁴Some programming languages, such as C, C++, Java use negative number, zero, and positive number to represent these three results. In Haskell, they are GT, EQ, and LE respectively.

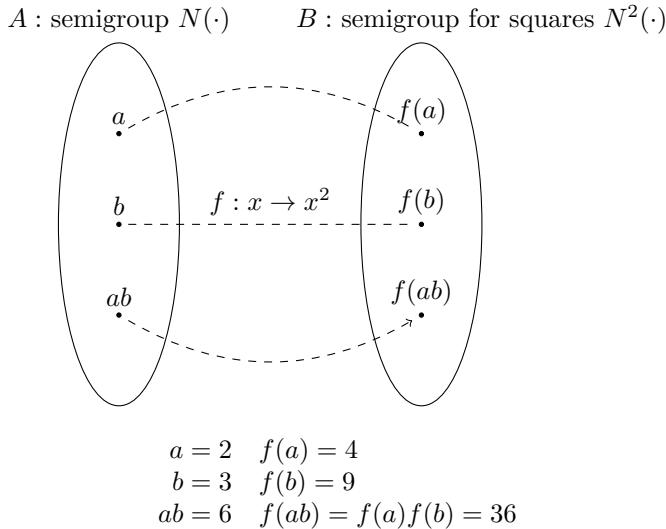


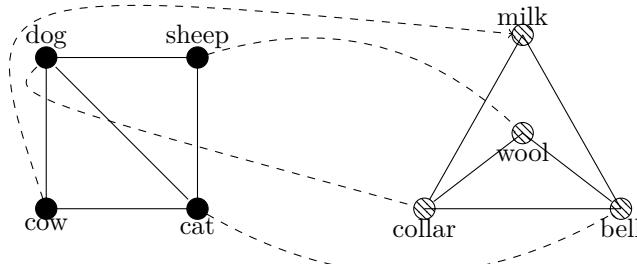
Figure 3.10: Isomorphism

1. Both a and b are odd numbers. $\text{odd}(a)$ and $\text{odd}(b)$ are both True. Their sum is even, $\text{odd}(a + b)$ is False. Equation $\text{odd}(a) \oplus \text{odd}(b) = \text{odd}(a + b)$ holds;
2. Both a and b are even numbers. $\text{odd}(a)$ and $\text{odd}(b)$ are both False. Their sum is also even, $\text{odd}(a + b)$ is False. Equation $\text{odd}(a) \oplus \text{odd}(b) = \text{odd}(a + b)$ holds;
3. a and b one is odd, the other is even; $\text{odd}(a)$ and $\text{odd}(b)$ one is True, the other is False. Their sum is odd, $\text{odd}(a + b)$ is True. Equation $\text{odd}(a) \oplus \text{odd}(b) = \text{odd}(a + b)$ holds.

If f is not only a surjection, but also a injection (there are no any two different elements map to the same image), then it's one-to-one mapping. We call f is the **isomorphism** from A to B . Isomorphism is a very powerful relationship. Besides group, isomorphism is also applicable to semigroups, monoids and other algebraic structures. As shown in figure 3.10. If A is isomorphic to B , from the abstracted view, they are essentially the same thing with different names. If there is a algebraic property in A , then there is exactly a same property in B ([24] pp.25). The special isomorphism from A to A is called **automorphism** of A . For example, the group of integers with addition has a automorphism under the negate operation.

People often use the familiar concrete, example to help understanding abstract algebraic structure. For groups, most time, we think about integers with addition. How those concepts, properties of group will be for integers? But this approach may give us illusion that the element of group is kind of entity, like numbers; and the binary operation is more like the common addition or multiplication that is commutative. We'll introduce an 'exceptional' example, the transformation group. On one hand, it is not abelian, the binary operation is not commutative; on the other hand, its group elements are not numbers, but transformations.

Transformation is a map from set A to A itself. Denoted as $\tau : A \rightarrow A$. It maps element a in A to $\tau(a)$. That is $a \rightarrow \tau(a)$. There are varies transformations for a set. The following are all the transformations for Boolean set, we denote true as T , false as F .



$$\begin{aligned} f(\text{dog}) &= \text{collar} & f(\text{cat}) &= \text{bell} \\ f(\text{sheep}) &= \text{wool} & f(\text{cow}) &= \text{milk} \end{aligned}$$

Figure 3.11: Graph isomorphism. The two different graphs have the same structure.

$$\begin{aligned} \tau_1 : & T \rightarrow T, & F \rightarrow T \\ \tau_2 : & T \rightarrow F, & F \rightarrow F \\ \tau_3 : & T \rightarrow T, & F \rightarrow F \\ \tau_4 : & T \rightarrow F, & F \rightarrow T \end{aligned}$$

Among them, τ_3 and τ_4 are one to one transformations. For a given set A , all its transformations form a new set:

$$S = \{\tau, \lambda, \mu, \dots\}$$

Let's next define a binary operation for S , and call it as multiplication. For convenient purpose, we express $\tau(a)$ as this way:

$$\tau : a \rightarrow a^\tau = \tau(a)$$

Note that a^τ does not mean the τ power of a . It is only a notation means transformation. Observe two elements τ and λ in S .

$$\tau : a \rightarrow a^\tau, \lambda : a \rightarrow a^\lambda$$

It's obvious that $a \rightarrow (a^\tau)^\lambda = \lambda(\tau(a))$ is also a transformation for A . We define it as the product of τ and λ .

$$\tau\lambda : a \rightarrow (a^\tau)^\lambda = a^{\tau\lambda}$$

Such multiplication is actually composition operation of two transformations. You can choose some Boolean transformations to verify their products. The multiplication is actually associative, since:

$$\begin{aligned} \tau(\lambda\mu) : & a \rightarrow (a^\tau)^{\lambda\mu} = ((a^\tau)^\lambda)^\mu \\ (\tau\lambda)\mu : & a \rightarrow (a^{\tau\lambda})^\mu = ((a^\tau)^\lambda)^\mu \end{aligned}$$

We benefit from the power expression. We can't say too much about a powerful notation system in the history of math. Euler, Leibniz were all masters invented many great symbols. For the above multiplication we defined, the identity element in S is the identity transformation of A , which maps every element to itself, $\epsilon : a \rightarrow a$. We can verify that:

$$\begin{aligned}\epsilon\tau : a &\rightarrow (a^\epsilon)^\tau = a^\tau \\ \tau\epsilon : a &\rightarrow (a^\tau)^\epsilon = a^\tau\end{aligned}$$

Therefore $\epsilon\tau = \tau\epsilon = \tau$. With this multiplication operation, set S almost forms a group. Although we say it ‘almost’, it eventually can not form a group. This is because for a given transformation τ , there is not necessarily an inverse element. For example the τ_1 in the Boolean set, it maps any Boolean value to true. None of the 4 transformations can change τ_1 back. Thus there is no reverse element for τ_1 .

Although S can not form a group, interestingly, its subgroup G can. In fact, as long as G only contains the one-to-one transformations of A , then it form a group under the multiplication operation.

For set A , the set of one to one transformations of A together with the composite operation (defined above as multiplication) form a **transformation group** of A . We have the below important theorem:

Theorem 3.1.4. *All the one to one transformations of set A form a transformation group G .*

Transformation groups are not necessarily abelian. It’s easy to find negative examples. Consider the shift transformation τ_1 which moves the origin point from $(0, 0)$ to $(1, 0)$ in the plane, and the rotation transformation τ_2 which rotate around the origin by $\pi/2$. We have:

$$\begin{aligned}\tau_1\tau_2 : (0, 0) &\rightarrow (0, 1) \\ \tau_2\tau_1 : (0, 0) &\rightarrow (1, 0)\end{aligned}$$

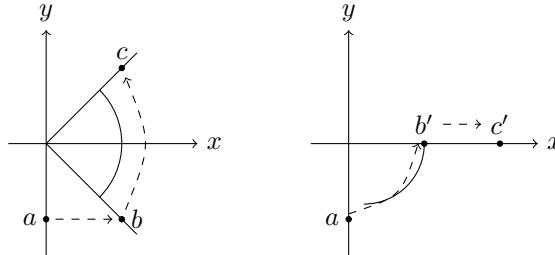


Figure 3.12: The transform order matters, and cause different results.

Therefore, this transformation group is not abelian. Transform groups are important and have wide applications. We have a strong fact:

Theorem 3.1.5. *Every group is isomorphic to a transformation group.*

We skip the proof. This theorem tells us, for any abstract group, we can find a concrete instance as a transformation group. In other words, we needn’t concern about finding an abstract group in the future, which is totally a castle in the air([24] pp49).

Exercise 3.3

1. Is the odd-even test function homomorphic between the integer addition group $(\mathbb{Z} +)$ and the Boolean logic-and group $(Bool, \wedge)$? What about the group of integers without zero under multiplication?
2. Suppose two groups G and G' are homomorphic. In G , the element $a \rightarrow a'$. Is the order of a same as the order of a' ?
3. Prove that the identity element for transformation group must be identity transformation.

3.1.4 Permutation group

In this section, we introduce permutation group. It is the permutation group that Galois used to determine if a given equation is radical solvable. Permutation group is a special transformation group. Let's first define what is permutation. A **permutation** is a one to one transformation for a finite set. The permutations of a finite set form a **permutation group** under the composite operation. As the name indicates, it permutes elements in the set. Further, the group of *all* permutations of a set with n elements is the **symmetric group** of degree n , denoted as S_n .

We know from the permutation theory learned in high school, there are $n!$ different permutations for n elements. Therefore, the symmetric group of degree n has the order of $n!$. A permutation maps element a_i in the set to a_{k_i} , where $i = 1, 2, \dots, n$. The permutation can be determined by n pairs of $(1, k_1), (2, k_2), \dots, (n, k_n)$. We can express the permutation as:

$$\begin{pmatrix} 1 & 2 & \dots & n \\ k_1 & k_2 & \dots & k_n \end{pmatrix}$$

For example

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 5 & 4 & 3 & 1 \end{pmatrix}$$

It represents a permutation. Since the first row is always in the form $1, 2, \dots, n$, we can further simplify the permutation to $(2, 5, 4, 3, 1)$. This permutation moves the 2nd element to the 1st position; moves the 5th element to the 2nd position and so on. We can use this notation to list all permutations for a set with 3 elements. It is group S_3 :

$$(1, 2, 3), (1, 3, 2), (2, 1, 3), (2, 3, 1), (3, 1, 2), (3, 2, 1)$$

When do the multiplication, which is composition essentially, we can determine the element for every position in this way: for the i -th position, first check what the new position should be in the first permutation, for example j ; then check where j should be mapped to in the second permutation, for example k . Therefore, the final position as the result of the multiplication is k . Let's pick two elements from S_3 to examine if it is abelian:

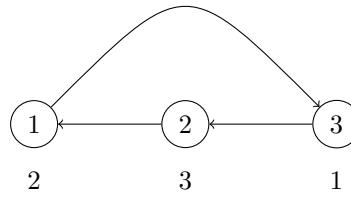
$$\begin{aligned} (1, 3, 2)(2, 1, 3) &= (2, 3, 1) \\ (2, 1, 3)(1, 3, 2) &= (3, 1, 2) \end{aligned}$$

They are different results. S_3 is not abelian. Actually, it is the smallest finite non abelian group. A finite non abelian group contains at least 6 elements. Observe the permutation for 5 elements $(2, 3, 1, 4, 5)$, only the first three ones change, while the rest two keep same. The changes for the first three elements have a pattern: $2 \rightarrow 1, 3 \rightarrow 2, 1 \rightarrow 3$, which is cyclic.

With this pattern, we can simplify the permutation notation from $(2, 3, 1, 4, 5)$ to $(2 \ 3 \ 1)$. Note there is no comma between elements, and we treat $(1 \ 2 \ 3)$, $(2 \ 3 \ 1)$, and $(3 \ 1 \ 2)$ represent the same 3-cyclic permutation¹⁵. Formally, we define k -cyclic permutation $(i_{j_1} i_{j_2} \dots i_{j_k})$. It maps the next element to its previous position, and send the first element to the last to form the cycle:

$$i_{j_2} \rightarrow i_{j_1}, i_{j_3} \rightarrow i_{j_2}, \dots, i_{j_1} \rightarrow i_{j_k}$$

¹⁵There are people use (231) notation without any spaces as delimiter. However, when there are over 10 elements, it causes ambiguity. As it can also be $(23 \ 1)$.

Figure 3.13: Permutation $(2\ 3\ 1)$ is cyclic

The elements in k -cyclic permutation are not necessary adjacent, for example $(3\ 9\ 4)$, nor in a fixed order, for example $(2\ 4\ 1\ 3)$. If there are only two element in the circle, like $(i\ j)$, we call it transposition (swap). Identity transformation is the special case, for example $(1, 2, 3, 4, 5)$, we denote it as $\epsilon = (1)$, and let:

$$\epsilon = (1) = (2) = \dots = (n)$$

Observe the permutation $(2, 1, 4, 5, 3)$, it contains two cycles, one is the 2-cyclic permutation $(1\ 2)$, the other is 3-cyclic permutation $(3\ 4\ 5)$. We can use the permutation multiplication to express this fact:

$$(2, 1, 4, 5, 3) = (1\ 2)(3\ 4\ 5)$$

In fact, every permutation π for n elements can be expressed with some mutual exclusive (without any common numbers) cyclic permutations. This approach brings us a useful advantage, although permutations are not commutative in common case, as k -cyclic permutations don't share same numbers, they are commutative. For example $(1\ 2)(3\ 4\ 5) = (3\ 4\ 5)(1\ 2)$.

The following list express all the permutations in S_3 in this way:

$$(1), (1\ 2), (1\ 3), (2\ 3), (1\ 2\ 3), (1\ 3\ 2)$$

Given a permutation (k_1, k_2, \dots, k_n) , how to express it as the product of cyclic forms? We can do it with such prescription. From left to right compare every position with the number in the permutation, if k_i equals i , it means the element has already in the right position; otherwise open a pair of parentheses, write down the number in position k_i in the parentheses, let say it is k_j , then check the position k_j with j , if they are not equal, write in the parentheses. Repeat this step till we found some element form a cycle to the starting point. At this time point, we close the parentheses of this cycle. After that, we go on checking from left to right till all the elements are processed ([25], pp27). If for all positions, we have k_i equals to i , then we write down (1) to present the identity permutation. It very convenient to realize this process in programming. Below is the algorithm for it with example source code in a real programming language.

```

function K-CYCLES( $\pi$ )
   $r \leftarrow []$ 
  for  $i \leftarrow 1$  to  $|\pi|$  do
     $p \leftarrow []$ 
    while  $i \neq \pi[i]$  do
       $p \leftarrow \pi[i] : p$ 
      Exchange  $\pi[i] \leftrightarrow \pi[\pi[i]]$ 
    if  $p \neq []$  then
       $r \leftarrow r : (\pi[i] : reverse(p))$ 
  
```

```

if r ≠ [] then
    return r
else
    return [[1]]                                ▷ Identity permutation

```

Express any permutation as product of k -cycle notation:

```

def kcycles(a):
    r = []
    n = len(a)
    for i in range(n):
        p = []
        while i + 1 != a[i]:
            p.append(a[i])
            j = a[i] - 1
            a[i], a[j] = a[j], a[i]
        if p != []:
            r.append([a[i]] + p)
    return r if r != [] else [[1]]

```

From the theorem in previous section, we further have:

Theorem 3.1.6. *Every finite group is isomorphic to a permutation group.*

For any finite group, like the solutions of a equation, we can study it with the permutation group, as it is easy to manipulate. This is exactly how Galois solve the problem to determine if the equation is radical solvable.

Exercise 3.4

1. List all the elements in S_4 .
2. Express all the elements in S_3 as the product of cyclic forms.
3. Write a program to convert the product of k -cycles back to permutation.

3.1.5 Groups and symmetry

Why the group of all permutations of a set is called symmetric group? Because it exactly defines what is symmetry. Let us consider a regular triangle for example:

Denote the 3 vertex as 1, 2, 3. We choose from S_3 three elements, $(1\ 2)$, $(1\ 2\ 3)$, and $(1\ 3\ 2)$, and apply them to the triangle vertexes as shown in figure 3.14.

1. The shape on the right side is the transformed result after applying $(1\ 2)$. The three vertexes change from 123 to 213. It is exactly the mirrored result to flip the triangle against the axis at vertex 3. Because the shapes are same before and after transform, it means the regular triangle is reflection symmetric. Besides $(1\ 2)$, there are another two reflection symmetric transforms, which are $(1\ 3)$ and $(2\ 3)$ respectively. Their corresponding axes are the heights at vertex 2 and 1.
2. The bottom right is the transformed result after applying $(1\ 2\ 3)$. The three vertexes change from 123 to 231. It means the triangle rotate 120° clockwise against its centre. Because the shapes are same before and after, it tells us that the regular triangle has rotational symmetry of 120° .
3. The bottom shape is the transformed result after applying $(1\ 3\ 2)$. The tree vertexes change from 123 to 312. It is equivalent that the triangle rotates 120° counter clockwise, or rotates 240° clockwise. Because the shapes are same before and after, it tells us that the regular triangle has rotational symmetry of 240° .

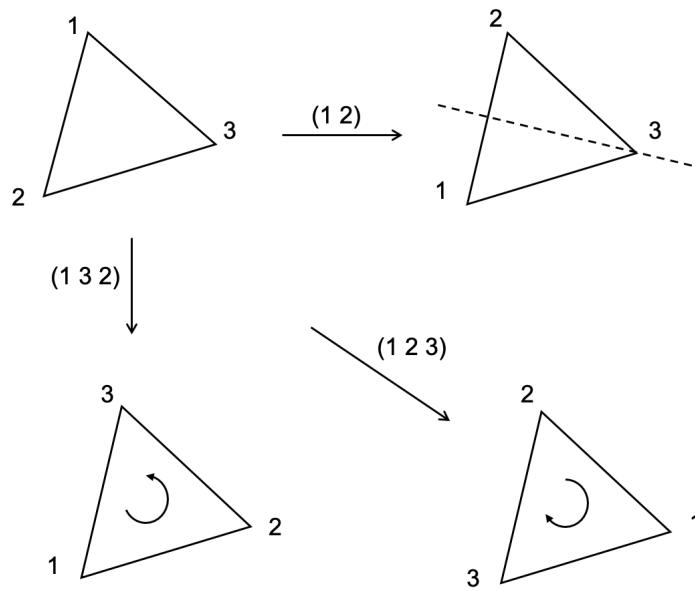


Figure 3.14: Symmetric transforms of a regular triangle

Every symmetry of the regular triangle (3 reflection symmetries, 2 rotational symmetries, and the identity) is exactly defined by an element in the symmetric group S_3 . This is the beautiful relation between groups and symmetry.

Among the transformations, those do not change the dimensions of the body are called *congruences*. A congruence is either proper or improper. For the difference, consider the two sea snails in below figure: on the left is the normally sinistral (left-handed) shell of *Neptunea angulata*, (now extinct) found mainly in the Northern Hemisphere; on the right is the normally dextral (right-handed) shell of *Neptunea despecta*, a similar species found mainly in the Southern Hemisphere. Although they look so symmetric side by side, we can not make them congruence no matter what rotations, flips, or motions being applied. They are reflexive in fact.

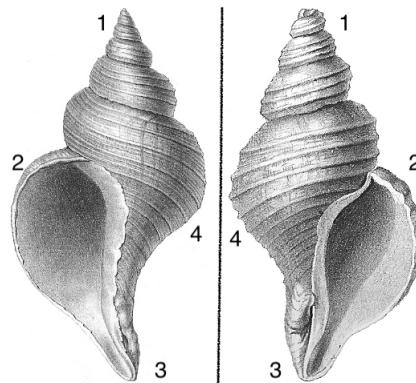


Figure 3.15: Chirality

In real world, there is no way to transform a sinistral object to its dextral image in the mirror. This is the difference between proper and improper congruence. The proper

congruence carries a left skew into a left and a right one into a right; the improper (or reflexive) congruence changes a left skew into a right one and vice versa[37]. The reflection sends any points P to its antipode point P' with respect to O found by joining P with O and prolonging the straight line PO by its own length: $|PO| = |OP'|$. It's also called inversion in a point. For the four points in the figure of sea snails, the transform (2 4) in symmetric group S_4 is an improper congruence. It carries the left skew snail into the right. As such, the symmetric groups depict both the real and the mirrored worlds.

Exercise 3.5

What symmetries for what shape are defined by the symmetric group S_4 ?

3.1.6 Cyclic group

Among all the groups, the cyclic group is the easiest. It's the group we completely understand as of today. Is it possible that all the elements in a group G are powers of a given element? Such group does exist. Consider the integer multiplicative group modulo 5 without 0 for example. It contains 4 elements $\{1, 2, 3, 4\}$. If list all the powers of 2 modulo 5, we have the following result:

$$\begin{aligned} 2^1 \bmod 5 &= 2 \\ 2^2 \bmod 5 &= 4 \\ 2^3 \bmod 5 &= 3 \\ 2^4 \bmod 5 &= 1 \end{aligned}$$

Thus the powers of 2 generate all the elements in this group. We define:

Definition 3.1.4. *If all the elements in a group G are the powers of a fixed element a , we say G is a **cyclic group**. In other words G is generated by element a , denoted as:*

$$G = (a)$$

We call a the generator of group G .

Let us see two important examples of cyclic group:

Example 3.1.1. *The additive group of integers. All integers are ‘powers’ of 1. Here the binary operation is addition, therefore, power means keep adding 1. For any positive integer m :*

$$\begin{aligned} 1^m &= 1 \cdot 1 \cdot 1 \cdot \dots \cdot 1 && m\text{-power of} \\ &= 1 + 1 + \dots + 1 && + \text{ is the multiplicative operation for } G(\mathbb{Z}, +) \\ &= m \end{aligned}$$

In additive group of integers, the inverse element of 1 is -1. This is because $1 + (-1) = 0$, and 0 is the identity element. For any negative integer $-m$:

$$\begin{aligned} 1^{-m} &= (-1)^m && \text{inverse element} \\ (-1)^m &= (-1) \cdot (-1) \cdot (-1) \cdot \dots \cdot (-1) && m \text{ times} \\ &= -1 + (-1) + \dots + (-1) && + \text{ is the multiplicative operation for } G(\mathbb{Z}, +) \\ &= -m \end{aligned}$$

As 0 is the identity element, we define $0 = 1^0$. Summarize all these three cases, we have $\mathbb{Z} = (1)$.

The additive group of integers is an example of infinite cyclic group. Let us see an example of finite cyclic group.

Example 3.1.2. Consider integers residue modulo n . For integer a , we use notation $[a]$ to represent the residue a belongs to. Define the binary operation as the addition modulo n .

$$[a] + [b] = [a + b]$$

For instance, when compute $[3] + [4]$ modulo 5, the result is $7 \bmod 5 = [2]$. It's easy to verify this binary operation satisfies the associative axiom. The identity element is $[0]$, and all elements have inverse. We call this group additive group of integer residues modulo n . The group elements are $[0], [1], \dots, [n-1]$. It is a cyclic group because every element $[i]$ can be expressed as i -power of $[1]$.

$$[i] = [1] + [1] + \dots + [1] \quad \text{Total } i \text{ times}$$

We intent to choose these two examples. In fact, we've already understood all the cyclic groups through these two examples! This is because of the following theorem:

Theorem 3.1.7. If G is a cyclic group generated by element a , then the algebraic structure of G is completely determined by order of a :

- If the order of a is infinite, then G is isomorphic to the additive group of integers;
- If the order of a is n , then G is isomorphic to the additive group of integer residues modulo n .

Proof. If the order of a is infinite, we have $a^h = a^k$ if and only if $h = k$. Otherwise, if $h \neq k$, let $h > k$ without loss of generality, we have $a^{h-k} = e$, which conflicts with the condition that the order of a is infinite. Therefore, we can construct a one to one map:

$$f : a^k \rightarrow k$$

This map is isomorphic between the cyclic group $G = (a)$ and the additive group of integers \mathbb{Z} . That is $a^h a^k \rightarrow h + k$.

If the order of a is integer n , which means $a^n = e$. Then $a^h = a^k$ if and only if $h \equiv k \pmod{n}$. We can construct another one to one map:

$$f : a^k \rightarrow [k]$$

This map is isomorphic between the cyclic group $G = (a)$ and the additive group of integer residues modulo n . That is:

$$a^h a^k = a^{h+k} \rightarrow [h + k] = [h] + [k]$$

□

Therefore, from the abstract perspective, there is only one group that the order of generator is infinite, and there is only one group that the order of generator is a given positive integer. We clearly understand the algebraic structure of these cyclic groups:

- The order of a is infinite
 - Group elements: $\dots, a^{-2}, a^{-1}, a^0, a^1, a^2, \dots$
 - Binary operation: $a^h a^k = a^{h+k}$
- The order of a is n
 - Group elements: $a^0, a^1, a^2, \dots, a^{n-1}$

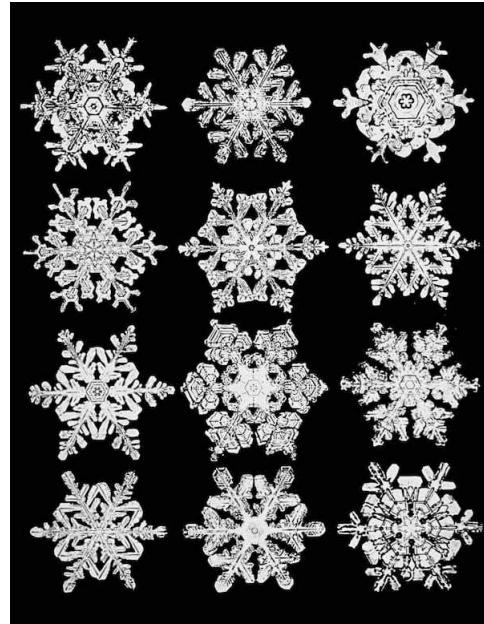


Figure 3.16: Snowflake crystal photos by Wilson Bentley, 1902

- Binary operation: $a^h a^k = a^{(h+k) \bmod n}$

Now we are ready to explain the symmetry of snowflake. The hexagon shaped snowflake has rotational symmetry, that can be exactly defined by C_6 . A snowflake aligns with itself after rotate $60^\circ, 120^\circ, 180^\circ, 240^\circ, 300^\circ$, and 360° around its centre. Each rotation is corresponding to an element of C_6 . Generic speaking, the cyclic group of order n defines the symmetry of regular n -polygon. It also defines the symmetry of the cross section of a regular polygonal prism. The famous photographer Wilson Bentley in US (1865 - 1931) took over 5000 photographs of snowflake crystal. He donated his collection, the crystal of science and art, to the Buffalo Museum of Science. From these photos, Bentley came to the conclusion that every child today knows: each snowflake is unique.

Exercise 3.6

1. Proof that cyclic groups are abelian.

3.1.7 Subgroup

The next important concept is subgroup. It helps us to understand a group through its subset.

Definition 3.1.5. *For a given group G , a subset H of G is called **subgroup** if H forms a group under the multiplication of G .*

For any group G , there are at least two subgroups. One is G itself, the other is the singleton set only contains the identity element $\{e\}$. They are called trivial subgroups. Taking the additive group of integers for example, all the even numbers under addition form a subgroup, while the odd numbers do not form a subgroup (do you know why?). Here is another example, for the permutation group S_3 , consider its subset $H = \{(1), (1\ 2)\}$. H form a subgroup under the composite operation. We can verify it as the following:

1. Close under multiplication: $(1)(1) = (1)$, $(1)(1\ 2) = (1\ 2)$, $(1\ 2)(1) = (1\ 2)$, $(1\ 2)(1\ 2) = (1)$. The last equation means to swap the first two elements, then swap again. It equals to identity transformation;
2. The associative law applies to all elements in S_3 , thus also applies to H ;
3. The identity element $(1) \in H$;
4. All elements in H have inverse: $(1)(1) = (1)$, $(1\ 2)(1\ 2) = (1)$.

It's tedious to verify all these group properties for arbitrary subset. Fortunately, there is a powerful tool:

Theorem 3.1.8. *For any group G , a none empty subset H forms a subgroup if and only if:*

1. *For all $a, b \in H$, the product $ab \in H$;*
2. *For all element $a \in H$, its reverse $a^{-1} \in H$.*

We leave the proof to this theorem as exercise. From this theorem, we can deduce that, if H is a subgroup of G , then the identity element in H is also the identity element in G , and for any element a in H , its reverse element in H is also the reverse element in G . We can combine the two conditions in this theorem into one:

Theorem 3.1.9. *For any group G , a none empty subset H forms a subgroup if and only if, for all $a, b \in H$, then $ab^{-1} \in H$ holds.*

Proof. First prove the sufficiency. Let $a, b \in H$, from the second condition in theorem 3.1.8, we have $b^{-1} \in H$. Then using the first condition, we get $ab^{-1} \in H$;

Next prove the necessity. Let $a \in H$, according to the identity axiom, we have $aa^{-1} = e \in H$. Since $e, a \in H$, therefore, $ea^{-1} = a^{-1} \in H$. This is the second condition in theorem 3.1.8; For all $a, b \in H$, we just proved $b^{-1} \in H$. Therefore, $a(b^{-1})^{-1} = ab \in H$. This is the first condition in theorem 3.1.8. \square

If the subset H is finite, then the condition to form a subgroup can be further simplified:

Theorem 3.1.10. *For any group G , the finite subset H forms a subgroup if and only if for all $a, b \in H$, $ab \in H$ holds.*

For another example, one of the most important subgroups for the symmetric group S_n of n objects, is the *alternating group* A_n . This subgroup contains all the permutations that when apply to x_1, x_2, \dots, x_n , the product

$$\Delta = \prod_{i < k} (x_i - x_k)$$

keeps same. Such permutations are called even permutations. They flip the sign of Δ even times, hence it is not changed. The rest permutations are odd permutations. The product of two even or odd permutations is still even, while an even one and an odd one gives an odd permutation. There are equal numbers of even and odd permutations, both are $n!/2$.

With integer n , we can partition all the integers into residues modulo n . Using the similar idea, we can partition the elements in a group. Consider the additive group of integers Z , let H be the subset of all multiples of n , i.e. $H = \{kn\}$ where $k = 0, \pm 1, \pm 2, \dots$. For any two elements hn and kn , $hn + (-k)n = (h - k)n \in H$. While $-kn$ is exactly the

inverse of kn in Z , and $+$ is the binary operation in additive group of integers. According to the theorem 3.1.9, H is the subgroup of Z .

When we partition integers into residues modulo n , we use the equivalence relation as:

$$a \equiv b \pmod{n}, \text{ if and only if } n|(a - b)$$

Using subgroup H , this equivalence relation can also be defined as:

$$a \equiv b \pmod{n}, \text{ if and only if } (a - b) \in H$$

Thus we use subgroup H to partition Z into residues. Now let's expand this to more generic case, to partition any group G with subgroup H . To do that, we need define the equivalence relation \sim with subgroup H first:

$$a \sim b, \text{ if and only if } ab^{-1} \in H$$

Given a and b , we can strictly determine if ab^{-1} belongs H . Why is \sim an equivalence relation? Because it satisfies all the three conditions for equivalence:

1. As $aa^{-1} = e \in H$, we have $a \sim a$. It is reflexive;
2. If $ab^{-1} \in H$, then its reverse $(ab^{-1})^{-1} = ba^{-1} \in H$. Therefore, $a \sim b \Rightarrow b \sim a$. It is symmetric;
3. If $ab^{-1} \in H, bc^{-1} \in H$, we have $(ab^{-1})(bc^{-1}) = ac^{-1} \in H$. Therefore $a \sim b, b \sim c \Rightarrow a \sim c$. It is transitive.

Definition 3.1.6. *The subset determined by the equivalence relation \sim is called the **right coset** of subgroup H . if a is a group element, the right coset that includes a is denoted as Ha .*

When using a to multiply every element in H from right, we get the coset that includes a . In other words, Ha contains all the elements in G with the form ha , where $h \in H$.

$$Ha = \{ha \mid h \in H\}$$

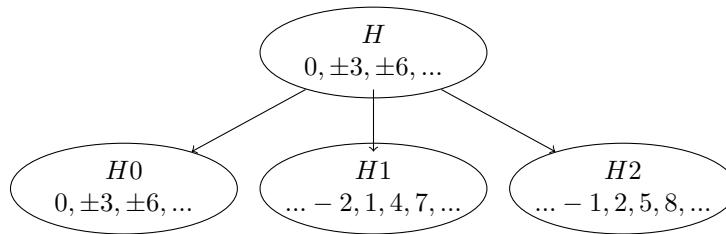


Figure 3.17: Right coset. For the additive group of integers, the subgroup H contains all multiples of 3. Using 0, 1, 2 add to all these multiples, we get three non-overlapping sets. It is exactly a partition of integers.

As shown in figure 3.17, Z is the additive group of integers. Set H contains all multiples of 3, includes $0, \pm 3, \pm 6, \dots$. It forms a subgroup under the addition. We add 0 in Z from right¹⁶ to every element in H , which gives $H0$. Obviously, $H0$ equals to H . It

¹⁶As the additive group of integers is abelian, the addition is commutative, therefore, the left and right cosets are same.

contains all the remainders of 0 modulo 3, denoted as [0]. Adding 1 in Z from right to every element in H gives $H1$. It contains all remainders of 1 modulo 3, denoted as [1]; Adding 2 in Z from right to every element in H gives $H2$. It contains all remainders of 2 modulo 3, denoted as [2]. If add 3 to every element in H , the result is as same as $H0$. In fact, whatever element a we choose from $H0$ to generate a right coset Ha , it is always as same as $H0$; whatever element b we choose from $H1$ to generate a right coset Hb , it is always as same as $H1$; whatever element c we choose from $H2$ to generate a right coset Hc , it is always as same as $H2$. Put the three right cosets $H0$, $H1$, and $H2$ together, the result is exactly all the integers Z . It is a partition of Z : [0], [1], [2].

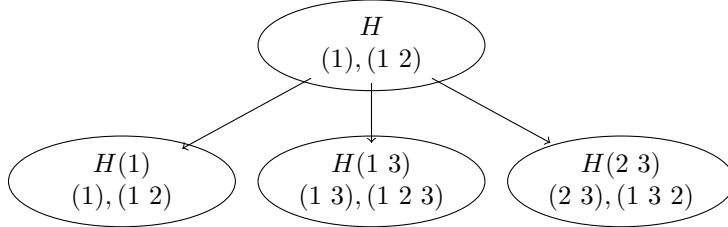


Figure 3.18: right cosets of a finite non-abelian group.

Let's see another example about finite non-abelian group. Consider a permutation group with degree 3.

$$G = S_3 = \{(1), (1 2), (1 3), (2 3), (1 2 3), (1 3 2)\},$$

It has a subgroup $H = \{(1), (1 2)\}$. We use the identity permutation (1) and another two permutations $(1 3)$, $(2 3)$ to multiply H from right. It gives 3 right cosets:

$$\begin{aligned} H(1) &= \{(1), (1 2)\} \\ H(1 3) &= \{(1 3), (1 2 3)\} \\ H(2 3) &= \{(2 3), (1 3 2)\} \end{aligned}$$

We can use another three different elements to form the right cosets:

$$H(1 2), H(1 2 3), H(1 3 2)$$

Because $(1 2) \in H(1)$, $(1 2 3) \in H(1 3)$, $(1 3 2) \in H(2 3)$, therefore we have:

$$\begin{aligned} H(1) &= H(1 2) \\ H(1 3) &= H(1 2 3) \\ H(2 3) &= H(1 3 2) \end{aligned}$$

The subgroup H partitions the group $G = S_3$ into three disjoint right cosets $H(1), H(1 3)$, and $H(2 3)$. Putting these three right cosets together gets exactly G . They form a partition of G .

Symmetric to right coset, we can also define left coset. Define the symmetric equivalence relation \sim' as:

$$a \sim' b \text{ if and only if } b^{-1}a \in H$$

The subset determined by equivalence relation \sim' is called left coset of subgroup H . The left coset respect to element a is denoted as aH . It contains all the elements in form of $ah, h \in H$ in G . As the multiplication operation for group may not be necessarily commutative, \sim is not identical to \sim' typically, therefore the left and right cosets are not necessarily same as well. However, there is a common property for both left and right cosets:

Theorem 3.1.11. *For a given subgroup H , there are same numbers of left and right cosets. They are either infinity many, or the same finite number.*

To prove this theorem, we can construct a map from the left coset to the right coset of H : $f : Ha \rightarrow a^{-1}H$. It's easy to verify this map is bijective (one to one correspondence). For all $Ha = Hb$, we have $ab^{-1} \in H$, and $(ab^{-1})^{-1} = ba^{-1} \in H$. Therefore, $a^{-1}H = b^{-1}H$. Since there exists one to one mapping, the numbers of left and right cosets are same.

With this theorem, we can use the number of the cosets (left or right) for a subgroup H to define the **index** of H in G .

In common cases, the right coset Ha does not equal to the left coset aH . If they are same, such subgroup is called normal subgroup. It was Galois, who first introduced normal subgroups to analyze if equations are solvable in radicals.

Definition 3.1.7. For a group G , the subgroup N is called **normal subgroup** (or invariant subgroup) if for every element a in G , the equation:

$$Na = aN$$

holds. The left (or right) coset for a normal group is called **coset** of N .

For the symmetric reason, a normal subgroup is also called the center of a group. We have two theorems to determine if a group is normal subgroup:

Theorem 3.1.12. For a group G , the subgroup N is a normal subgroup (or invariant subgroup) if and only if for every element a in G , equation:

$$aN a^{-1} = N$$

holds.

Theorem 3.1.13. For a group G , the subgroup N is a normal subgroup (or invariant subgroup) if and only if for every element a in G , n in N , equation:

$$a n a^{-1} \in N$$

holds.

For a normal subgroup N , all the cosets $\{aN, bN, cN, \dots\}$ form a set. We define the multiplication for this set as:

$$(xN)(yN) = (xy)N$$

It easy to verify that, the set of cosets form a group under this multiplication operation. This group is called **quotient group**, denoted as G/N . There is an important relation between the normal subgroup, quotient group, and homomorphism. First, there is a homomorphism between G and every its quotient group G/N . To proof it, we can construct a map $a \rightarrow aN$, $a \in G$. It is obvious that this map is surjective. For all elements a, b in G , we have $ab \rightarrow abN = (aN)(bN)$. Therefore, it is a surjective homomorphism. This fact tell us, we can either use the subgroup, or quotient group to understand the property of G . To do that, let us define the concept of 'kernel'.

Definition 3.1.8. If f is a surjective homomorphism from group G to another group G' , consider the identity element e' in G' , its preimage under f is a subset of G . This subset is called the **kernel** of the homomorphism.

We have the following theorem. If G is homomorphic to G' , then the kernal N of the homomorphism is the normal subgroup of G . And the quotient group $G/N \cong G'$. A group is homomorphic to every of its quotient group, and from the abstract point of view,

G can only be homomorphic to its quotient groups. Sometimes, we find G is homomorphic to G' , and we don't know well about the properties of G' . However, we are sure to be able to find a normal subgroup N of G , that the properties of G' are essentially identical to the quotient group G/N . We can see the importance of the normal subgroup and the quotient group. Galois did use this idea to figure out the way to define the equation group is solvable or not.

Exercise 3.7

1. Proof theorem 3.1.8, which determines if a subset forms a subgroup.
2. List the left cosets for H in figure 3.18.

3.1.8 Lagrange's theorem

Lagrange's theorem greatly demonstrates the power of abstract algebra. We can reveal the inner pattern of the abstract structure even without knowing any concrete meanings of the group elements or their operations.

Joseph-Louis Lagrange, was an mathematics, physics, and astronomer. He was born on January 25, 1736 in Turin, Italy. Lagrange was of Italian and French descent. His paternal great-grandfather was a French army officer who had moved to Turin, and married an Italian. His father, who had charge of the king's military chest and was Treasurer of the Office of Public Works and Fortifications in Turin. But before Lagrange grew up he had lost most of his property in speculations. A career as a lawyer was planned out for Lagrange by his father, and certainly Lagrange seemed to have accepted this willingly. He later claimed: "If I had been rich, I probably would not have devoted myself to mathematics." Lagrange studied at the University of Turin. At first he had no great enthusiasm for mathematics, finding Greek geometry rather dull.

It was not until he was seventeen that he showed any taste for mathematics – his interest in mathematics being first excited by a paper by Edmond Halley which he came across by accident. That paper introduced about the new calculus invented by Newton. Alone and unaided he threw himself into mathematical studies.

Starting from 1754, he worked on the problem of tautochrone, discovering a method of maximising and minimising functionals in a way similar to finding extrema of functions. Lagrange wrote several letters to Leonhard Euler between 1754 and 1756 describing his results. His work made him one of the founders of the calculus of variations. Euler was very impressed with Lagrange's results. As an accomplished mathematician, Lagrange was appointed to be an mathematics assistant professor at the Royal Military Academy of the Theory and Practice of Artillery in 1755. Already in 1756, Euler and Maupertuis, seeing Lagrange's mathematical talent, tried to persuade him to come to Berlin, but Lagrange had no such intention and shyly refused the offer.

In 1766, king Frederick of Prussia wrote to Lagrange expressing the wish of "the greatest king in Europe" to have "the greatest mathematician in Europe" resident at his court. Lagrange was finally persuaded and he spent the next twenty years in Prussia,



Stamp of Joseph-Louis Lagrange

where he produced not only the long series of papers published in the Berlin and Turin transactions, but also his monumental work, the *Mécanique analytique*.

Lagrange was a favourite of the king, who used frequently to discourse to him on the advantages of perfect regularity of life. The lesson went home, and thenceforth Lagrange studied his mind and body as though they were machines, and found by experiment the exact amount of work which he was able to do without breaking down. Every night he set himself a definite task for the next day, and on completing any branch of a subject he wrote a short analysis to see what points in the demonstrations or in the subject-matter were capable of improvement. He always thought out the subject of his papers before he began to compose them, and usually wrote them straight off without a single erasure or correction.

Nonetheless, during his years in Berlin, Lagrange's health was rather poor on many occasions, and that of his wife Vittoria was even worse. She died in 1783 after years of illness and Lagrange was very depressed.

In 1786, following Frederick's death, Lagrange accepted the offer of Louis XVI to move to Paris. In France he was received with every mark of distinction and special apartments in the Louvre were prepared for his reception, and he became a member of the French Academy of Sciences. It was about the same time, 1792, that the unaccountable sadness of his life and his timidity moved the compassion of 24-year-old Renée-Françoise-Adélaïde Le Monnier, daughter of his friend, the astronomer Pierre Charles Le Monnier. She insisted on marrying him, and proved a devoted wife to whom he became warmly attached.

In September 1793, the French revolution broke out. Under intervention of Antoine Lavoisier (known as the father of modern chemistry, who recognized and named oxygen and hydrogen), who himself was by then already thrown out of the Academy along with many other scholars, Lagrange was specifically exempted by name in the decree of October 1793 that ordered all foreigners to leave France. On May 4, 1794, Lavoisier, who had saved Lagrange from arrest, and 27 other tax farmers were arrested and sentenced to death and guillotined on the afternoon after the trial. According to a story, the appeal to spare Lavoisier's life so that he could continue his experiments was cut short by the judge, J. B. Coffinhal: "The Republic has no need of scientists or chemists; the course of justice cannot be delayed.". Lagrange said on the death of Lavoisier on May 8: "It took only a moment to cause this head to fall and a hundred years will not suffice to produce its like."^[26]

After Coup of Brumaire 18, 1799, Napoleon attained the power of France. He warmly encouraged science and mathematics studies in France, and was a liberal benefactor of them. He loaded Lagrange with honors and distinctions, appointed Lagrange as senator. In 1808, Napoleon made Lagrange a Grand Officer of the Legion of Honour and a Count of the Empire. Then honoured him with the Grand Croix of the Ordre of the Reunion on April 3, 1813. A week after, Lagrange died on April 10 at the age of 77. The funeral operation was pronounced by Laplace, represented the House of Lords, and Dean Lacépède represented the French Academy. The commemorative events were held in Italian universities.

Lagrange was the great mathematician and scientist in the 18 to 19 Century. He made significant contributions to the fields of analysis, number theory, and both classical and celestial mechanics. Napoleon said "Lagrange is the lofty pyramid of the mathematical science". But above all his contribution, he is best known for his work on mechanics, where he transformed Newtonian mechanics into a branch of analysis, Lagrangian mechanics as it is now called, and presented the so-called mechanical "principles" as simple results of the variational calculus.

Lagrange made important progress in solving algebraic equations of any degree in the first decades in Berlin. He introduced a concept called Lagrange resolvents. The significance of this method is that it exhibits the already known formulas for solving equations

of second, third, and fourth degrees as manifestations of a single principle. He failed to give a general formula for solutions of an equation of degree five and higher, because the auxiliary equation involved had higher degree than the original one. Nevertheless, Lagrange's idea already implied the concept of permutation group. The permutations made the resolvent invariant form a subgroup, and the order of the subgroup is the factor of the original permutation group. This is exactly the famous Lagrange's theorem in group theory. Lagrange is a pioneer of group theory. His thoughts were adopted and developed later by Abel and Galois, and was foundational in Galois theory.

Let us first introduce a lemma before Lagrange's theorem.

Lemma 3.1.14. *For a subgroup H , there exists one to one mapping between H and every right coset Ha .*

As the left and right cosets are symmetric, this lemma applies to left cosets as well. To prove it, we can build a map $f : h \rightarrow ha$. It is one to one mapping from the subgroup to right cosets because:

1. For every element h in H , there exists unique image ha ;
2. For every element ha in Ha , it is the image of h in subgroup H ;
3. For any $h_1a = h_2a$, the equation $h_1 = h_2$ holds.

From the existence of this one to one mapping, we know that for finite group G , the number of elements for any coset must equal to the order of the subgroup H . And according to the partition nature, every element in the group must be in some coset. Therefore, we have the insight between the subgroup H and the finite group G through cosets:

Theorem 3.1.15. Lagrange's theorem: *For any finite group G , the order (number of elements) of every subgroup H of G divides the order of G .*

Proof. First, we know that G can be fully partitioned by the cosets of H ; From the equivalence relation defined for coset, we know there is no overlap among them. If element c belongs to both Ha and Hb , then $c \sim a$, and $c \sim b$, therefore, $a \sim b$. Equation $Ha = Hb$ holds. And with the one to one mapping between the subgroup H and cosets, we know the order of every coset equals to the order of H , which is $|H| = n$. Given the number of cosets is m (which equals to the index of H), we finally get the result:

$$|G| = mn$$

□

Note that there is a conversed question to Lagrange's theorem, whether every divisor of the order of a group is the order of some subgroup? This does not hold in general. Later we'll see such negative example in figure 3.32 (c). The order of alternative group A_4 is 12, but it has no subgroup of order 6. We can deduce many interesting results from Lagrange's theorem.

Corollary 3.1.16. *If G is a finite group, the order of every element a is a divisor of the order of G .*

This is because a generates a subgroup of order n , therefore n divides $|G|$.

Corollary 3.1.17. *If G has prime order, then G is cyclic.*

This is because for every element a excludes the identity element, the subgroup generated by a has the same order as G . Therefore, a is the generator of G , i.e. $G = (a)$.

Corollary 3.1.18. *For every element a in a finite group G , equation $a^{|G|} = e$ holds.*

This is because the order n of a divides $|G|$, let $|G| = nk$, we have:

$$a^{|G|} = a^{nk} = (a^n)^k = e^k = e$$

Lagrange's theorem in group theory can be used to prove the Fermat's little theorem in number theory. The theorem is named after Pierre de Fermat, who found it in 1636. In a letter he wrote to a friend¹⁷ in October 18, 1640, Fermat stated this theorem first. It is called the "little theorem" to distinguish it from Fermat's last theorem. Euler provided the first published proof in 1736. But Leibniz had given virtually the same proof in an unpublished manuscript from sometime before 1683.

Theorem 3.1.19. Fermat's little theorem: *If p is prime, for any integer a that $0 < a < p$, then p divides $a^{p-1} - 1$.*

Proof. Consider the multiplicative integer group modulo p . The group elements are all none zero residues modulo p . As p is prime, therefore the group elements are $1, 2, \dots, p-1$. The identity element $e = 1$. The order of the group is thus $p-1$. According to the corollary 3.1.18 of Lagrange's theorem, we have:

$$a^{p-1} = e$$

Since the identity element is 1, this equation can be written as:

$$a^{p-1} \equiv 1 \pmod{p}$$

Therefore, p divides $a^{p-1} - 1$. □

Compare to this method, the way to prove Fermat's little theorem in elementary theory of number is much more complex (for example, section 5.2 - 5.4 in [10]). Here we give another interesting combinatorial method called proof by counting necklace[27].

Proof. Consider there are pearls in a different colors. we are going to make strings of length p , where p is prime. Obviously there are total a^p different strings, because every pearl can be chosen among a colors, and we need make p times selection.

For example there are pearls in two different colors: A red, and B green. When make strings containing 5 pearls, that is $a = 2, p = 5$, there are total $2^5 = 32$ different strings:

AAAAA, AAAAB, AAABA, ..., BBBBA, BBBBB.

Corresponding to

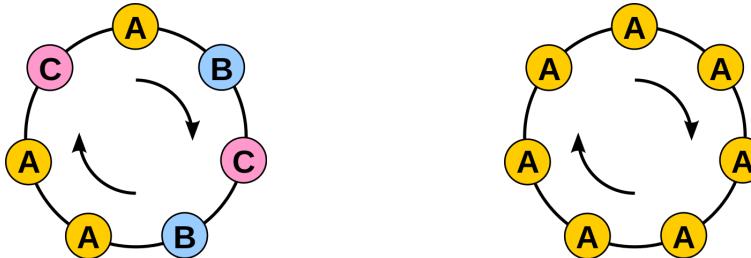
red-red-red-red-red,
red-red-red-red-green,
red-red-red-green-red,
...,
green-green-green-green-red,
green-green-green-green-green.

We are going to prove that, among these a^p strings, if remove a strings that are all in the same color (in the above example, they are strings of AAAAA and BBBBB), the rest $a^p - a$ strings can be divided in several groups. Each group has exactly p strings, thus p divides $a^p - a$.

If we link the head and tail for every pearl string to make a necklace, some different strings will become the same necklace. When a string can transform to the other by rotation, the two strings must form the same necklace. The following 5 strings form the same necklace for example:

¹⁷Friend and confidant, French mathematician Frénicle de Bessy.

AAAAAB, AAABA, AABAA, ABAAA, BAAAA.



(a) A necklace in 3 colors represents 7 different strings: ABCBAAC, BCBAACA, CBAACAB, BAACABC, AACACBC, ACABCBA, CABCBAA

(b) The necklace in same color only represent one string: AAAAAAA

Figure 3.20: Partition strings through necklace

By this means, the 32 pearl strings in above example, can be partitioned into 5 necklaces in different colors and 2 necklaces in the same color:

[AAABB, AABBA, ABBAA, BBAAA, BAAAB];
 [AABAB, ABABA, BABAA, ABAAB, BAABA];
 [AABBB, ABBBA, BBBAA, BBAAB, BAABB];
 [ABABB, BABBA, ABBAB, BBABA, BABAB];
 [BBBBB, BBBBA, BBBAB, BBABB, BABBB];
 [AAAAA];
 [BBBBB].

How many strings can a necklace represent? If a string S can be split into several same sub-string T , while T can't be split into sub-string any more, then the necklace S represents $|T|$ different strings, where $|T|$ is the length of sub-string T . For example, string $S = \text{ABBABBABBABB}$ can be split into sub-string $T = \text{ABB}$, while ABB can't be split any more. If we rotate a pearl per time, there are total 3 different results:

ABBABBABBABB,
 BBABBABBABBA,
 BABBABBABBAB.

There are not any other different strings besides these 3. Since the length of ABB is 3, further rotation must give the same result. Basically, there are two types for all the a^p pearl strings. One contains a strings in same color; the rest are strings in different colors. However, as the length of the string p is prime, it cannot be generated by duplicating sub-strings. Therefore, every necklace in different colors, represents p different strings. There are total $a^p - a$ strings in different colors. They can be partitioned into groups by the necklaces. Each group contains exactly p strings all can be represented with the same necklace. It tells us p must divide $a^p - a = a(a^{p-1} - 1)$. Since a and p are coprime, hence p divides $a^{p-1} - 1$.

□

The proof by counting necklaces might be the most straightforward method people developed. It need little mathematical knowledge. The key idea is two different counting methods must give the same result.



Pierre de Fermat, 1601-1665

It took 100 years from the birth of Fermat's little theorem to Euler's proof. It is not rare, but a typical Fermat style. Fermat's last theorem stimulated many talented mathematicians. It took 358 years till British mathematician Andrew Wiles solved it in 1995 successfully. The main tools Wiles used include elliptic curves, modularity theory, and Galois representations[12]. These conjectures left by Fermat are a rich mathematical treasure.

Pierre de Fermat was a French mathematician, born about August 1601. His father was a wealthy leather merchant. He became a lawyer at the Parlement of Toulouse French after growing up. In 1630, he bought the office of a councillor at the Parlement of Toulouse, one of the High Courts of Judicature in France, and was sworn in by the Grand Chambre in May 1631. He held this office for the rest of his life. Fermat thereby became

entitled to change his name from Pierre Fermat to Pierre de Fermat. He was fluent in six languages. Fermat studied mathematics in his spare time. But the mathematical achievements made by Fermat were the peak of his time. Fermat's pioneering work in analytic geometry was circulated in manuscript form in 1636. It was based on results he achieved in 1629¹⁸. Together with René Descartes, Fermat was one of the two leading mathematicians of the first half of the 17th Century developed analytic geometry.

Fermat and Blaise Pascal laid the foundation for the theory of probability through their correspondence in 1654. From this brief but productive collaboration on the problem of points, they are now regarded as joint founders of probability theory. Fermat is credited with carrying out the first ever rigorous probability calculation.

In physics, Fermat refined the ancient Greek result about light and generalized to "light travels between two given points along the path of shortest time" now known as the principle of least time. For this, Fermat is recognized as a key figure in the historical development of the fundamental principle of least action in physics. The terms Fermat's principle and Fermat functional were named in recognition of this role. Fermat also contributed to the early development of calculus.

But Fermat's crowning achievement was in the theory of numbers. Fermat was inspired by the Diophantus's great work *Arithmetica* in ancient Greek. It was translated into Latin and published in France in 1621 by Claude Bachet. Fermat bought this book in Paris, and was deeply attracted by the puzzles in theory of numbers. One special feature of this edition was there were wide margin left in pages, it turned to be Fermat's 'note book' when reading it. During the study of Diophantus' problems and answers, Fermat often got inspired to consider wider and deeper problems, then wrote his thoughts and comments in the margin.

Fermat published nearly nothing in his lifetime, although it is unbelievable from the view point of today. It was common in Fermat's time, that sometimes he wrote mails to his scholar friends about his findings. Some of the most striking of his results were found after his death on loose sheets of paper or written in the margins of works which he had read and annotated, and are unaccompanied by any proof. He was constitutionally modest and retiring, and does not seem to have intended his papers to be published. Fermat was totally driven by the strong curiosity to explore the mathematical mysteries. It's

¹⁸The 8 pages paper, *Introduction to Plane and Solid Loci* was completed in 1630, but it was published posthumously in 1679, which was 14 years after Fermat's death.

purely because he treated mathematics study as a hobby. When he found the result that had never been touched, Fermat was truly exciting and self-satisfied. It was not significant to him to publish the result and get recognition[12]. Interestingly, this silent genius sometimes liked to tease people. He often challenged other mathematicians in mails by asking them to prove his discoveries.

When Fermat died in Jan, 1665, his research results were scattered here and there. His son Clément-Samuel Fermat spent 5 years to collect Fermat's mails and notes, then produced a special edition of *Arithmetica* contained his father's achievement. On the cover page, it printed "augmented with Fermat's commentary". This edition includes 48 comments by Fermat. In 1679, Samuel collected and published the second volume of Fermat's work. His research results were finally circulated, which greatly enriched the mathematics in the 17th Century, and impact the development of mathematics later.

Before Fermat, the theory of numbers was basically a collection of problems. It was Fermat who first systematically studied the theory of numbers. He proposed a large number of theorems, and introduced generalized methods and principles, thus brought the theory of numbers to the modern development. It can be said that it was Fermat's systematic work that the theory of numbers really began to become a branch of mathematics. With his gift for number relations and his ability to find proofs for many of his theorems, Fermat essentially created the modern theory of numbers. He was called the "father of modern number theory." Before Gauss's *Arithmetic Research*, the development of number theory was originally driven by Fermat.

However, for many propositions conjectured, Fermat only provided some key part, or even without any proof. Some of them were found wrong¹⁹. Before given the strict mathematical proof, these propositions could only be called conjecture. Most of them were later solved by Euler. What's more, Euler greatly developed the theory of numbers on top of Fermat's work.

The Euler theorem in theory of numbers, is more generic than Fermat's little theorem. Euler did not satisfied after successfully proved Fermat's little theorem. What if p is not prime? After carefully studied the general condition that covered composite numbers, Euler found and proved the following theorem.

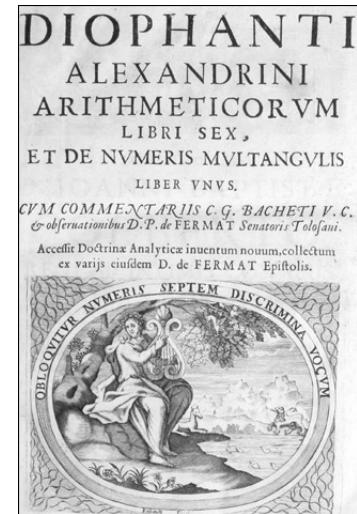
Theorem 3.1.20. Euler theorem: *If $0 < a < n$, a and n are coprime, then n divides $a^{\phi(n)} - 1$.*

Where $\phi(n)$ is Euler function²⁰. It is defined as the number of all positive integers that less than n , and coprime to n .

$$\phi(n) = |\{i|0 < i < n \text{ and } \gcd(i, n) = 1\}|$$

¹⁹For example Fermat number, named after Fermat who first studied them in 1640, is a positive number of the form $2^{2^n} + 1$. Fermat claimed all such numbers are primes. It is true when n is 0, 1, 2, 3, 4. The corresponding numbers are 3, 5, 17, 257, 65537. However, in 1732 Euler calculated that $2^{2^5} + 1 = 641 \times 6700417$, is not a prime number. As of 2017, people have found 243 negative examples, without finding the 6th Fermat prime number. It is still a unsolved conjecture whether there are any other Fermat primes.

²⁰Also know as Euler totient function, or Euler $\phi(n)$ function.



The 1670 edition of the *Arithmetica* of *Diophantus*, with Fermat's annotation.

Euler proved this theorem with the method in elementary theory of numbers. There is a elegant proof by using Lagrange's theorem in group theory.

Proof. Consider the none zero residues modulo n . We pick out all the mutually inverse residues under the multiplication modulo n . They form a multiplicative group modulo n . From the definition of Euler ϕ function, the value of $\phi(n)$ is the number of all positive integers that less than n and coprime to n . While these positive numbers are exactly the elements of the multiplicative group. Thus the order of this group is $\phi(n)$. From the corollary 3.1.18 of Lagrange's theorem, we have:

$$a^{\phi(n)} = e$$

Therefore, $a^{\phi(n)} \equiv 1 \pmod{n}$, which immediately gives the result, n divides $a^{\phi(n)} - 1$. \square

Let us see an example. Below is the multiplication table for all the none zero residues modulo 10. We can locate the identity element 1, and mark the cells underline. Then from that cell, along with the row and column, we can find two numbers, marked in bold. Their modulo product is 1, and both are coprime to 10. On the other hand, for every residue number that is not coprime to 10, the row and column where it is in also contain 0. But 0 is not group element. We can see that all the residues that are coprime to 10 are exactly the group elements 1, 3, 7, 9.

	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9
2	2	4	6	8	0	2	4	6	8
3	3	6	9	2	5	8	<u>1</u>	4	7
4	4	8	2	6	0	4	8	2	6
5	5	0	5	0	5	0	5	0	5
6	6	2	8	4	0	6	2	8	4
7	7	4	<u>1</u>	8	5	2	9	6	3
8	8	6	4	2	0	8	6	4	2
9	9	8	7	6	5	4	3	2	<u>1</u>

Given integer n , how to evaluate Euler function? As any integer greater than 1 can be factored as power of prime numbers, let us first see how to evaluate $\phi(p^m)$ for the m power of prime p . We are going to count how many numbers from 1 to $p^m - 1$ are coprime to p^m . We can easily do this by removing the multiples of p . These numbers are $p, 2p, 3p, \dots, p^m - p$. Divide them by p , gives the nature number sequence $1, 2, 3, \dots, p^{m-1} - 1$. We immediately know there are $p^{m-1} - 1$ numbers. On top of this, we deduce the value of Euler function for the power of prime as:

$$\begin{aligned} \phi(p^m) &= (p^m - 1) - (p^{m-1} - 1) \\ &= p^m - p^{m-1} \\ &= p^m \left(1 - \frac{1}{p}\right) \end{aligned}$$

We next consider number in form of $n = p^u q^v$, which is product of power of different prime numbers. We need first remove all multiples of p , then remove all multiples of q . But there are numbers that are the multiples of both p and q . They are removed

twice. We need add these multiples of pq back (Principle of inclusion and exclusion in combinatorics). Thus we have:

$$\begin{aligned}\phi(p^u q^v) &= (n-1) - \left(\frac{n}{p}-1\right) - \left(\frac{n}{q}-1\right) + \left(\frac{n}{pq}-1\right) \\ &= n\left(1-\frac{1}{p}\right)\left(1-\frac{1}{q}\right) \\ &= p^u\left(1-\frac{1}{p}\right)q^v\left(1-\frac{1}{q}\right) \\ &= \phi(p^u)\phi(q^v)\end{aligned}$$

Particularly when both u and v are 1, we have $\phi(pq) = \phi(p)\phi(q)$. We can expand this result to multiple powers of prime numbers. Given $n = p_1^{k_1} p_2^{k_2} \dots p_m^{k_m}$, the Euler function can be evaluated as:

$$\begin{aligned}\phi(n) &= n\left(1-\frac{1}{p_1}\right)\left(1-\frac{1}{p_2}\right)\dots\left(1-\frac{1}{p_m}\right) \\ &= \phi(p_1^{k_1})\phi(p_2^{k_2})\dots\phi(p_m^{k_m})\end{aligned}$$

We can develop the fast evaluation algorithm from this result. We leave this as an exercise in this chapter.

Leonhard Euler was a great Swiss mathematician and scientist. He is held to be one of the greatest mathematician in the history together with Archimedes, Newton, and Gauss. Euler was born on April 15, 1707 in Basel Switzerland. As a paster, his father urged him to study theology and became paster too. Euler enrolled at the University of Basel at the age of 13 with major of philosophy and theology. During that time, he was receiving Saturday afternoon lessons from Johann Bernoulli, the foremost mathematician in Europe. Bernoulli quickly discovered Euler's incredible talent for mathematics, and convinced his father that Leonhard was destined to become a great mathematician.

In 1727, Euler became a member of Imperial Russian Academy of Sciences in Saint Petersburg. He devoted himself to research work and later became the head of mathematics department after his friend, Daniel Bernoulli left for Basel. During the 14 years in Russia, Euler studied analytics, number theory, and mechanics. In 1747, Euler took up a position at the Berlin Academy, which he had been offered by Frederick the Great of Prussia. He lived for 25 years in Berlin, where he wrote over 380 articles. During this time, his research was more extensive, involving planetary motion, rigid body movement, thermodynamics, ballistics, and demography. These work was closely connected with his research in mathematics. Euler made groundbreakings in differential equations, and surface differential geometry. After the political situation in Russia stabilized after Catherine the Great's accession to the throne, in 1766 Euler accepted an invitation to return to the St. Petersburg Academy. He spent the rest of his life in Russia.

Euler worked in almost all areas of mathematics, such as geometry, infinitesimal calculus, trigonometry, algebra, and number theory, as well as continuum physics, lunar theory and other areas of physics. He is a seminal figure in the history of mathematics; From the age of 19 till he died at 76, he published huge number of research papers and books in half a Century. Euler's name is associated with almost every area in mathematics. He was the top productive mathematician in the history with a total of 856 papers, and 31 books. And these did not counted the loss of the fire in St. Petersburg in 1771. (Euler's record was refreshed by Hungarian mathematician Paul Erdős in the 20th Century, who published 1525 papers and 32 books) [28].



Stamps commemorating Leonhard Euler (1707 - 1783)

Euler had unbelievable strong willpower. His right eye lost sight from a fever. Three years later, he became almost blind in his right eye. But even worse, his left eye lost sight too in 1771. But Euler rather blamed the painstaking work on cartography he performed for his condition. Just as deafness did not stop Beethoven's music creation, blindness did not stop Euler's mathematical exploration[12]. Euler remarked on his loss of vision, "Now I will have fewer distractions." As he compensated for it with his mental calculation skills and exceptional memory. He could not only remember the first 100 prime numbers, but also their squares, cubics, and even higher powers. He could also perform complex mental arithmetic. With the aid of his scribes, Euler's productivity on many areas of study actually increased. He produced, on average, one mathematical paper every week in the year 1775. Half of Euler's work was dictated after his eyes were completely blind. The French physicist François Arago said "Euler calculated without any apparent effort, just as men breathe, as eagles sustain themselves in the air." Euler could work in any bad environments. He often held the child on the lap to complete the papers, regardless of any noise around.

Among Euler's work, there are difficult monographs as well as readings for the general public. Euler wrote over 200 letters to a German Princess in early 1760s, which were later compiled into a best-selling volume entitled *Letters of Euler on different Subjects in Natural Philosophy Addressed to a German Princess*. This book became more widely read than any of his mathematical works and was published across Europe and in the United States. The popularity of the "Letters" testifies to Euler's ability to communicate scientific matters effectively to a lay audience, a rare ability for a dedicated research scientist. Euler also wrote a course on elementary algebra for readers of non-mathematics background, which is still in print today. Many popular mathematical notations we are using today were carefully introduced by Euler, for example π (1736), the imaginary unit i (1777), the base of the natural logarithm e , now also known as Euler's number (1748), circular function \sin , \cos (1748), and \tg (1753), Δx (1755), summation \sum (1755), function $f(x)$ (1734) and so on[12].

On September 18, 1783, after lunch with his family, Euler was discussing the newly discovered planet Uranus and its orbit with a fellow academician. As usual, he played with one of his granddaughter while having tea. Suddenly, the pipe drop from his hand. He said “My pipe”, then bent over to pick it, but he was not able to stand, uttered only “I am dying” before he lost consciousness. “He ceased to calculate and to live.”²¹

Fermat little theorem is widely used in our everyday life, from internet shopping to electronic payment. In 1976, professor Whitfield Diffie and Martin Hellman in Stanford University developed the concept of asymmetric public-private key cryptosystem. In 1977, Ron Rivest, Adi Shamir, and Leonard Adleman in Massachusetts Institute of Technology (MIT) developed a one-way function that was hard to invert based on theory of numbers. The algorithm is now known as RSA –the initials of their surnames in same order as their paper.

The key asymmetry concept of RSA is based on the fact that we can easily create a composite numbers from two large prime numbers, while there is practical difficulty to factor them. This is known as the *factoring problem*. For a large number of over 200 digits, even the most powerful super computer will cost time longer than the age of universe. In order to construct a encrypt key hard to break, we need a method that can find large prime numbers fast. However, people do not know the exact pattern about how prime numbers distributed in nature numbers. There is no formula to enumerate prime numbers. The brute force method is to randomly pick a number n , then examine from 1 to \sqrt{n} , check if all do not divide n . But this primality test method is very ineffective. The time will also exceed the age of universe. A better method is the Eratosthenes sieve algorithm. We first enumerate all numbers from 2 to n . Starting from 2, removal all multiples of 2, then remove all multiples of 3... Repeat this every time from the next number that is not filtered out, till the number not greater than \sqrt{n} . This process gives all the prime numbers to n . However, it is still only applicable for small n , but can't serve for the large number primality test.

Interestingly, Fermat's little theorem provides a way for fast primality test. For a large number n , we can randomly select a positive integer a less than it as a ‘witness’. Then check if the remainder of a^{n-1} modulo n is 1. If not 1, n must not be prime number according to Fermat's little theorem. Otherwise if it is 1, then n may be a prime number.

The *Fermat primality test* algorithm (also known as Fermat test) based on this idea can be described as below:

```

function PRIMALITY( $n$ )
  Random select a positive integer  $a < n$ 
  if  $a^{n-1} \equiv 1 \pmod{n}$  then
    return prime number
  else
    return composite number

```

We needn't compute the exact value of $n - 1$ power of a , then divide n to get the remainder. We can use modular arithmetic to speed up. The intermediate result can be largely re-used. After we calculate $b = a^2 \pmod{n}$, we can next get $b^2 \pmod{n}$, which equals to $a^4 \pmod{n}$. For example, when evaluate $a^{11} \pmod{n}$, since:

$$a^{11} = a^{8+2+1} = ((a^2)^2)^2 a^2 a \pmod{n}$$

What have to calculated are only $a^2 \pmod{n}$, $(a^2)^2 \pmod{n}$, and $((a^2)^2)^2 \pmod{n}$. Basically, we can express n in binary format, and only iterate calculating the modular

²¹In the eulogy for the French Academy, French mathematician and philosopher Marquis de Condorcet wrote this.

product for the digits of 1. The complexity of this algorithm is $O(\lg n)$ (proportion to the logarithm of n). Fermat test is very fast because of this.

However, even a number can pass Fermat test, it is not necessarily prime. For example $341 = 11 \times 31$ is a composite number, but $2^{340} \equiv 1 \pmod{341}$ can pass the Fermat test. To reduce the probability of such failure, people developed many improvements. The first improvement is to increase the number of witness. We can prove that, if a number does not pass Fermat test, then there exist at least half of n numbers that can't pass too ([30], pp26).

Theorem 3.1.21. *For positive integer a less than n , and coprime with n , if $a^{n-1} \not\equiv 1 \pmod{n}$, then for all selected $a < n$, it also holds for at least half.*

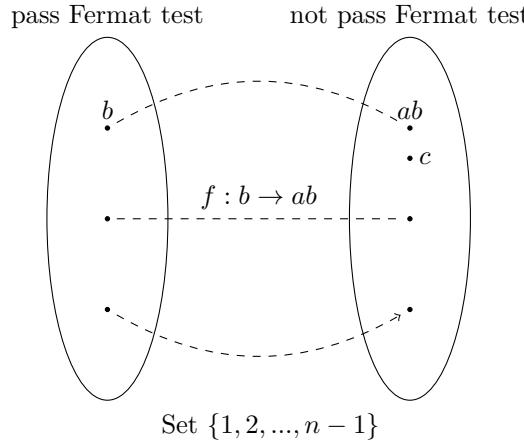


Figure 3.24: Map from the set that pass Fermat test to the set that does not

Proof. If for some a that $a^{n-1} \not\equiv 1 \pmod{n}$ holds, then for any witness b that passes Fermat test (It means $b^{n-1} \equiv 1 \pmod{n}$), we can create a negative case for Fermat test ab .

$$(ab)^{n-1} \equiv a^{n-1}b^{n-1} \equiv a^{n-1}1 \not\equiv 1 \pmod{n}$$

And because $i \neq j$, we have $a \cdot i \not\equiv a \cdot j$, therefore, all these negative cases are not same.

As shown in figure 3.24, if there exists a number that can't pass Fermat test, then the positive cases are as same amount as the negative cases. \square

If we select k different witness and perform Fermat test, we can reduce the probability of failure that n is not prime number to $\frac{1}{2^k}$. However, there exist such composite number n , that for any a less than n and coprime to n , $a^{n-1} \equiv 1 \pmod{n}$ holds. It means whatever a we selected, such composite number can pass Fermat test. Carmichael found first such number in 1910, that $561 = 3 \times 11 \times 17$. This kind of numbers is called Carmichael numbers or Fermat pseudoprime²². Erdős conjectured there are infinite many Carmichael numbers. In 1994, people proved for sufficient large n , there are at least $n^{2/7}$ Carmichael numbers from 1 to n . Thus explains Erdős' conjecture[29].

²²The Czech mathematician Václav Šimerka found the first 7 Fermat pseudoprimes: $561 = 3 \times 7 \times 11$, $1105 = 5 \times 13 \times 7$, $1729 = 7 \times 13 \times 19$, $2465 = 5 \times 17 \times 29$, $2821 = 7 \times 13 \times 31$, $6601 = 7 \times 23 \times 41$, $8911 = 7 \times 19 \times 67$. However, his work was not well known to the people

The primality test algorithm in RSA is Miller-Rabin algorithm. It is also a probabilistic algorithm²³. According to the above theorem, if select more than 100 witnesses, the probability of failure is expected less than $\frac{1}{2^{100}}$. Donald Knuth commented “far less, for instance, than the probability that a random cosmic ray will sabotage the computer during the computation!”

We summarized the relations for group, semigroup, monoid introduced so far as the following diagram.

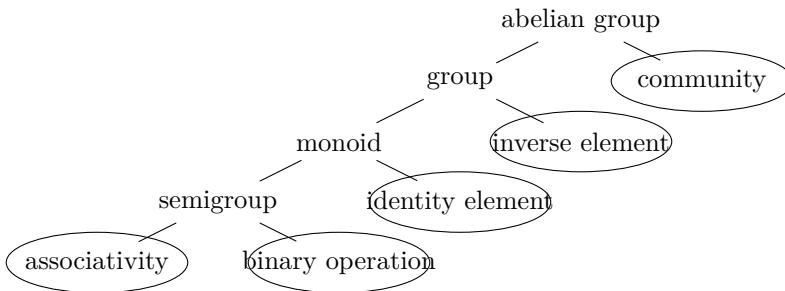


Figure 3.25: group, semigroup, monoid

Exercise 3.8

1. Today is Sunday, what day it will be after 2^{100} days?
2. Given two strings (character string or list), write a program to test if they form the same necklace.
3. Write a program to realize Eratosthenes sieve algorithm.
4. Extend the idea of Eratosthenes sieve algorithm, write a program to generate Euler ϕ function values for all numbers from 2 to 100.
5. Write a program to realize fast modular multiplication, and Fermat’s primality test.

3.2 Ring and Field

The modern theory of rings, fields, and abstract algebra was greatly developed by German mathematician Emmy Noether. Noether was born to a Jewish family in Erlangen Germany on March 23, 1882. Her father was a mathematician in the University of Erlangen. She originally planned to teach French and English after passing the required examinations in 1900, but she chose instead to continue her studies at the University of Erlangen where her father lectured.

This was an unconventional decision. Two years earlier, the Academic Senate of the university had declared that allowing mixed-sex education would “overthrow all academic order”. As one of only two women in a university of 986 students, Noether was only allowed to audit classes rather than participate fully, and required the permission of individual professors whose lectures she wished to attend. Despite these obstacles, on July 14, 1903 she passed the graduation exam at a Realgymnasium in Nuremberg.

²³There is a deterministic version of Miller-Rabin algorithm, but the correctness is on top of the generalized Riemann hypothesis (GRH)[31].

During the 1903–1904 winter semester, she studied at the University of Göttingen, attending lectures given by astronomer Karl Schwarzschild and mathematicians Hermann Minkowski, Otto Blumenthal, Felix Klein, and David Hilbert. Noether returned to Erlangen. She officially reentered the university in October 1904, and declared her intention to focus solely on mathematics. Under the supervision of Paul Gordan she wrote her dissertation. For the next seven years (1908–1915) she taught at the University of Erlangen's Mathematical Institute without pay only because she was a woman.

The mathematician Ernst Fischer was an important influencer on Noether, in particular by introducing her to the work of David Hilbert. From 1913 to 1916 Noether published several papers extending and applying Hilbert's methods to mathematical objects such as fields of rational functions and the invariants of finite groups. This phase marks the beginning of her engagement with abstract algebra, the field of mathematics to which she would make groundbreaking contributions.

In the spring of 1915, Noether was invited to return to the University of Göttingen by David Hilbert and Felix Klein. Their effort to recruit her, however, was blocked by the philologists and historians among the philosophical faculty: Women, they insisted, should not become privatdozenten. Hilbert responded with indignation, stating, "I do not see that the sex of the candidate is an argument against her admission as privatdozent. After all, we are a university, not a bath house."

During her first years teaching at Göttingen she did not have an official position and was not paid; her family paid for her room and board and supported her academic work. Her lectures often were advertised under Hilbert's name, and Noether would provide "assistance".

Soon after arriving at Göttingen, however, she demonstrated her capabilities by proving the theorem now known as Noether's theorem, which shows that a conservation law is associated with any differentiable symmetry of a physical system. American physicists Leon M. Lederman and Christopher T. Hill argue in their book *Symmetry and the Beautiful Universe* that Noether's theorem is "certainly one of the most important mathematical theorems ever proved in guiding the development of modern physics, possibly on a par with the Pythagorean theorem".

In 1919 the University of Göttingen allowed Noether to proceed with her habilitation. Noether was not paid for her lectures until she was appointed to the special position in 1923.

Noether's work in algebra began in 1920. She published the important paper *Theory of Ideals in Ring Domains* in 1921. This revolutionary work gave rise to the term "Noetherian ring" and the naming of several other mathematical objects as Noetherian. In 1931, Noether's student, Dutch mathematician Van der Waerden published *Moderne Algebra*, a central text in the field developed by Noether. Although Noether did not seek recognition, Van der Waerden included a note "based in part on lectures by E. Artin and E. Noether". This classic book influenced a generation of young mathematicians in that time. In Göttingen, Noether supervised more than a dozen doctoral students. In November 1932, Noether delivered one hour speech at the 9th International Congress of Mathematicians in Zürich. with 800 attendees. Apparently, this prominent speaking position was a recognition of the importance of her contributions to mathematics. The 1932 congress is sometimes described as the high point of her career.

However, the huge reputation and recognition did not improve her difficult situation



Emmy Noether, 1882-1935

as a woman. Her colleagues were frustrated for the fact that she was not elected to the Göttingen academy of sciences and was never promoted to the position of full professor. Her frugal lifestyle was due to being denied pay for her work; However, even after the university began paying her a small salary in 1923, she continued to live a simple and modest life.

After Hitler gain the power of German, Nazi administration persecuted Jews intensified. In 1929, Noether was taken out of the apartment where she lived. In April 1933, Noether was forced to stop teaching at the University of Göttingen. Several of Noether's colleagues, including Max Born and Richard Courant, also had their positions revoked. Noether accepted the decision calmly, providing support for others during this difficult time. Hermann Weyl later wrote that “Emmy Noether—her courage, her frankness, her unconcern about her own fate, her conciliatory spirit—was in the midst of all the hatred and meanness, despair and sorrow surrounding us, a moral solace.”

As dozens of newly unemployed professors began searching for positions outside of Germany. Albert Einstein and Hermann Weyl were appointed by the Institute for Advanced Study in Princeton, while others worked to find a sponsor required for legal immigration. Although Noether was the world famous mathematician, it was very hard for her to get a position from large institutions as a women. After a series of negotiations with the Rockefeller Foundation, a grant to Bryn Mawr College in the United States was approved for Noether and she took a position there, starting in late 1933. This is a girl's University, and Noether lectured there as a visiting scholar. Although she was invited to Princeton University to give lectures, she remarked that she was not welcome at “the men's university, where nothing female is admitted”.

In April 1935, doctors discovered a tumor in Noether's pelvis. She was died after the surgery on April 14 at the age of 53.

Noether is best remembered for her contributions to abstract algebra. Nathan Jacobson wrote that: “The development of abstract algebra, which is one of the most distinctive innovations of twentieth century mathematics, is largely due to her.” She sometimes allowed her colleagues and students to receive credit for her ideas, helping them develop their careers at the expense of her own. She was described by Pavel Alexandrov, Albert Einstein, Jean Dieudonné, Hermann Weyl and Norbert Wiener as the most important woman in the history of mathematics[32].

3.2.1 Ring

Let us see how ring is defined.

Definition 3.2.1. *A ring is a set R that satisfies below **ring axioms**:*

1. *R is an additive group, it means R forms an abelian group under the addition operation defined on it;*
2. *There is multiplication defined for R , and R is close under this multiplication operation;*
3. *The multiplication is associative. For all a, b, c , equation $(ab)c = a(bc)$ holds;*
4. *Multiplication is distributive with respect to addition, meaning that, for all a, b, c , we have:*

$$\begin{aligned} a(b+c) &= ab + ac \\ (b+c)a &= ba + ca \end{aligned}$$

Obviously, all integers form a ring under the addition and multiplication. Other examples are polynomials and matrix form ring under the addition and multiplication.

The ring definition only requires the addition operation is commutative, thus the ring is an abelian group under addition. It does not have such requirement for multiplication. When the ring multiplication is also commutative, we call it **commutative ring**. In commutative ring, for positive integer n and any two elements, the following equation holds:

$$a^n b^n = (ab)^n$$

The ring definition does not require an identity element for the multiplication too. If there exists an element e in R , it satisfies the below equation for all elements:

$$ea = ae = a$$

We call e the **unity** (multiplicative identity element) of the ring. A ring may not necessarily have unity²⁴. Traditionally, we use 1 to represent the unity. It is just a symbol, but not means the actual number 1. With unity, we can also define inverse element. If $ab = ba = 1$, we say b is the inverse element of a . According to the group property, we know that if the ring has an identity element for multiplication, it is unique. If an element is invertible, the inverse element is also unique. While it is not necessary that all elements are invertible. For example, the integer ring has unity, but except for ± 1 , all other integers are not invertible. We call the invertible element a **unit**.

From the distributive axiom, we have:

$$(a - a)a = a(a - a) = aa - aa = 0$$

in short:

$$0a = a0 = 0$$

It means for two elements a, b in the ring, if either one is 0, then $ab = 0$. But the inverse proposition does not hold. We can not deduce from $ab = 0$ to either a or b is 0. Let's see a negative example. Consider the residue class modulo n , where the addition operation is modulo addition: $[a] + [b] = [a + b]$, which takes modulo n on top of the sum. It forms an abelian addition group. The multiplication modulo n is defined as: $[a][b] = [ab]$, which takes modulo n on top of the product. It's easy to verify that this is a ring, named as residue ring modulo n .

If n is not prime, for example 10, we can multiply two none zero elements $[5][2] = [5 \times 2] = [0]$. In fact, for any two factors of n , their modulo product must be zero.

In a ring, if there exists $a \neq 0, b \neq 0$, but $ab = 0$, we call a is a **left zero divisor**, and b is a **right zero divisor** of the ring. For the commutative ring, a left zero divisor is also right zero divisor. Of course, it's possible there is no zero divisor, for example the integer ring. For $ab = 0$, we can deduce either a or b is 0 only if there is no zero divisor in the ring. And we have the following theorem:

Theorem 3.2.1. *For a ring without zero divisor, the following two cancellation rules hold.*

- If $a \neq 0$, and $ab = ac$, then $b = c$;
- If $a \neq 0$, and $ba = ca$, then $b = c$.

²⁴More people define a ring to have a multiplicative identity nowadays, and use symbol *rng* for a ring without multiplicative identity.

Conversely, if one cancellation rule holds in a ring, then the other cancellation rule also holds, and there is no zero divisor in this ring.

So far, we introduced three additional conditions: (1) Multiplication is commutative; (2) There exists unity; (3) No zero divisor. A ring that satisfies these three additional constraints is called **integral domain** (nonzero commutative ring). It's obvious that integer ring is an integral domain.

The ring constraints are strong. Sometimes, we needn't the additive inverse. By weakening the limitation, we obtain the **semiring**.

Definition 3.2.2. *A set R under the addition and multiplication forms a semiring if it satisfies the following axioms:*

1. *R forms a commutative monoid under addition, and there exists the identity element 0 ;*
2. *R forms a monoid under multiplication, and there exists the identity element 1 (unity);*
3. *Multiplication is distributive with respect to addition. For any a, b, c :*

$$\begin{aligned} a(b + c) &= ab + ac \\ (b + c)a &= ba + ca \end{aligned}$$

4. *Any element multiplies 0 gives 0 , meaning that: $a0 = 0a = 0$.*

Natural numbers N is an example of semiring. The Boolean operations form a semiring with two elements.

Exercise 3.9

1. Prove the theorem that the two cancellation rules hold in a nonzero ring (ring without zero divisor).
2. Prove that, all real numbers in the form of $a + b\sqrt{2}$, where a, b are integers form a integral domain under the normal addition and multiplication.

3.2.2 Division ring and field

We know that not all elements have inverse element in a ring. If every non-zero element has its inverse, then it forms a special ring. For example all rational numbers form a ring under the normal addition and multiplication. In this ring, every non-zero element a has its inverse $\frac{1}{a}$.

Definition 3.2.3. *A ring R is a **division ring** if it satisfies the following conditions:*

1. *R contains at least one non-zero element;*
2. *R has unity;*
3. *Every non-zero element in R is invertible (unit).*

Definition 3.2.4. *A commutative division ring is a **field** 域.*

According to this definition, all rational numbers form a field. Similarly, all real numbers and all complex numbers form fields under the addition and multiplication²⁵. There are some interesting properties for division ring and field. There is no zero divisor in a division ring. This is because if $a \neq 0$, and $ab = 0$, then we multiply the inverse element of a from left to both sides:

$$a^{-1}ab = b = 0$$

Thus b must be zero, therefore, division ring is a nonzero ring (does not contain zero divisor). The other property is that, all non-zero elements in division ring R form a group under multiplication, denoted as R^* . We call R^* the multiplicative group of division ring R . A division ring consists of two groups: addition group and multiplication group. The distributive axiom bridges the two groups together. We summarized the ring, semiring, integral domain, division ring, and field as figure 3.27.

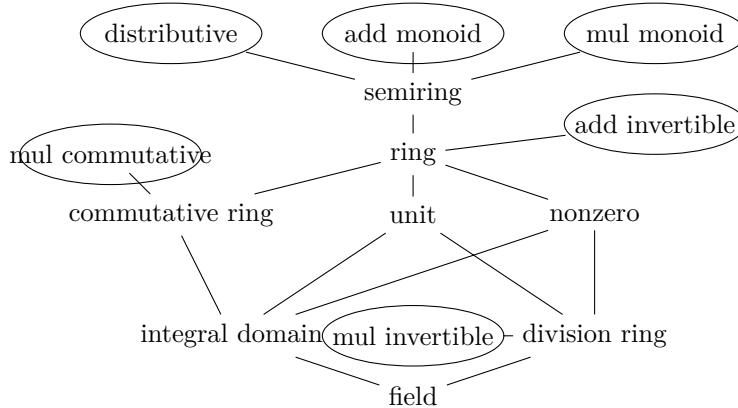


Figure 3.27: semiring, ring, integral domain, and field

We skipped some important algebraic structure here like subring, ideal, principal ideal domain, and Euclidean domain.

3.3 Galois theory

Galois theory provides connection between group and field. Using Galois theory, certain problems in field theory can be reduced to group theory, which is in some sense simpler and easier to understand. A field can contain infinite many elements with adding, subtracting, multiplying, and dividing. It's a complex mathematical object. Galois theory can simplify the problems in field to a corresponding problem in finite group with only one operation. This is the key idea of Galois theory.

Galois theory is famous for its difficulty and hard to understand. Compare to its original form, the modern Galois theory is no longer ambiguous, and much clearer and elegant. This is because more than two dozens of masters greatly developed and improved it in the past. The most important contributors are Jordan, Dedekind, and Emil Artin. Jordan and Dedekind first systematically developed Galois theory in France and Germany. The definition of Galois group we are using today was given by Dedekind. The modern form of Galois theory is made by Artin[33]. We adopted a gentle explanation that is friendly to beginners[34] in this short section. It's quite common that we can't understand

²⁵We'll see in later chapter, rational number field is countable, while real number field and complex number field are not countable.

it in the first time reading. It's always helpful to read the masters. Our life is not linear. I recommend the way to revisit what we learned before, find more great books to read. Our understanding changes along the time, and there must be aha moment in the future.

3.3.1 Field extension

From the definition of field, we know that all rational numbers form a field Q . Let us consider a set of all the numbers in the form of $a + b\sqrt{2}$, where a and b are rational numbers[35]. Obviously, the set of rational numbers is a subset of it (just let $b = 0$). It's easy to verify that, for any two such numbers, the result of add, subtract, multiply, and divide are still in the form of $a + b\sqrt{2}$. Among them, divide takes more steps to verify.

Given $x = \frac{1}{a + b\sqrt{2}}$, we can multiply both nominator and denominator by $a - b\sqrt{2}$ to get:

$$\begin{aligned} x &= \frac{a - b\sqrt{2}}{(a + b\sqrt{2})(a - b\sqrt{2})} \\ &= \frac{a - b\sqrt{2}}{a^2 - 2b^2} \end{aligned}$$

Let $p = a^2 - 2b^2$, then x can be expressed as $(a/p) - (b/p)\sqrt{2}$. Thus we verified for divide operation as well. This set is really a field. We denote it as $Q[\sqrt{2}]$. Similarly, $Q[\sqrt{3}]$ is also a field. They are both example of field extension concept.

Definition 3.3.1. *If field E contains field F , then E is **field extension** of F , denoted as $F \subseteq E$ or E/F .*

For example, the field of real numbers is the field extension of rational numbers; $Q[\sqrt{2}]$ is the field extension of rational numbers Q . Basically, for field F , if $\alpha \in F$, but $\sqrt{\alpha} \notin F$, then $F_1 = F[\sqrt{\alpha}]$ is also a field. We can keep extending the field with this method. If $\beta \in F_1$, but $\sqrt{\beta} \notin F_1$, then:

$$\begin{aligned} F_2 &= F_1[\sqrt{\beta}] \\ &= F[\sqrt{\alpha}][\sqrt{\beta}] \\ &= F[\sqrt{\alpha}, \sqrt{\beta}] \end{aligned}$$

All numbers like $a + b\sqrt{\beta}$ where $a, b \in F_1$ form a higher level field extension. Starting from rational numbers, we can obtain a series of field extensions $Q \subset F_1 \subset F_2 \dots \subset F_n$.

Why does field extension matter? For example, equation $x^2 - 2 = 0$ can not be solved in the field of rational numbers (there are not any rational numbers satisfy this equation), however, if we extend from the rational number field, there is pair of solutions in the $Q[\sqrt{2}]$, which are $x = \pm\sqrt{2}$. Here is another example, for equation $x^4 - 5x^2 + 6 = 0$, there is no rational number solution, but there are two solutions in field extension $Q[\sqrt{2}]$, which are $\pm\sqrt{2}$; if we extent the field one more step to $Q[\sqrt{2}, \sqrt{3}]$, we obtain the complete four solutions: $\pm\sqrt{2}$, and $\pm\sqrt{3}$. It leads to the important concept of **splitting field**.

Definition 3.3.2. *For equation $p(x) = 0$, the smallest field extension that contains all the roots is called the **splitting field** of $p(x)$. It's also called as **root field**.*

Thus the splitting field of equation $x^2 - 2 = 0$ is $Q[\sqrt{2}]$. Why is it named as 'splitting'? The polynomial $x^2 - 2$ can not be decomposed in rational number field Q , however, in field extension $Q[\sqrt{2}]$ it can be *split* to:

$$(x + \sqrt{2})(x - \sqrt{2})$$

For a given polynomial equation, if we can start from the basic rational number field, with a series of field extension, reach to its splitting field, then this equation is radical solvable.

We've seen the example of square root. There are more complex cases. Cubic equation may need cubic root, some equation even requires imaginary unit i . From the high school math, we know that for the simplest equation $x^3 - 1 = 0$, there are p roots of $1, \zeta, \zeta^2, \dots, \zeta^{p-1}$, where $\zeta \neq 1$ is a complex root in the unit circle. We need a strict description of field extension to cover these cases.

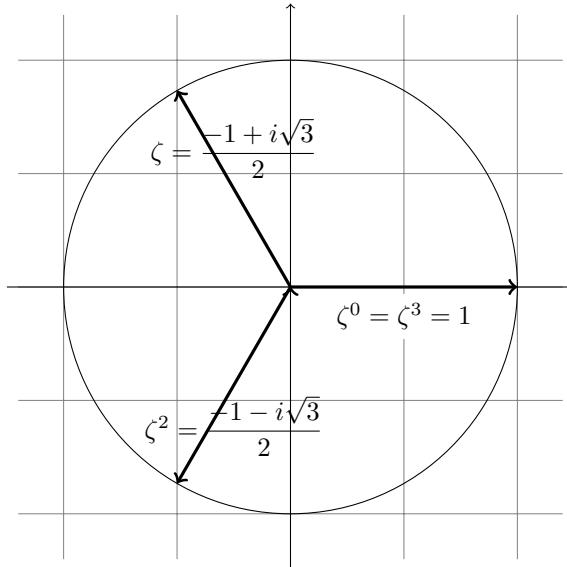


Figure 3.28: Unit root of $x^3 - 1 = 0$

If the integer power of α is some element b in field F , which means $\alpha^m = b \in F$, such α can be obtained through radical form of b as $\alpha = \sqrt[m]{b}$. We can extend the field with a series of such radical forms to $F[\alpha_1][\alpha_2] \dots [\alpha_k]$. If each α_i is radical form, we call the field extension $F[\alpha_1, \alpha_2, \dots, \alpha_k]$ as radical extension of F . Let the equation roots be x_1, x_2, \dots, x_n (Note x_i may not be radical form), if the splitting field $E = Q[x_1, x_2, \dots, x_n]$ is radical extension, then the equation is radical solvable.

Exercise 3.10

1. Prove that $Q[a, b] = Q[a][b]$, where $Q[a, b]$ contains all the expressions combined with a and b , such as $2ab, a + a^2b$ etc.

3.3.2 Automorphism and Galois group

Our thought so far is to start from the rational number field, where the coefficients of the equation belong to, then move forward to the splitting field step by step through a series of radical extensions. How to further simplify the problem to find the answer? At this stage, Galois changed his mind to another direction. He turned to consider the symmetry among the roots of the equation. For example, There is a pair of roots for equation $x^2 - 2 = 0$, which are $\pm\sqrt{2}$. Obviously, if replace $\sqrt{2}$ with $-\sqrt{2}$, the equation still holds. Besides that, replacing $\sqrt{2}$ in equation $\sqrt{2}^2 + \sqrt{2} + 1 = 3 + \sqrt{2}$ with $-\sqrt{2}$, then the equation holds too. It means, the pair of roots $\pm\sqrt{2}$ is symmetric to the relation $\alpha^2 + \alpha + 1 = 3 + \alpha$. We can further say, for any expressions that only consist of addition

and multiplication of $\sqrt{2}$, the pair of roots can be exchanged with each other. We know that the most powerful tool to describe symmetry is group, in particular, the symmetric group (we just introduced it in section 3.1.4). To connect field extension and group, Galois introduced an important concept of field automorphism.

Using the same example of $Q[\sqrt{2}]$, we can define a function from $Q[\sqrt{2}]$ to $Q[\sqrt{2}]$ with type $f : Q[\sqrt{2}] \rightarrow Q[\sqrt{2}]$. The definition is as this:

$$f(a + b\sqrt{2}) = a - b\sqrt{2}$$

Then f is a field automorphism.

Definition 3.3.3. *Field Automorphism* is a invertible function f , that maps the field to itself. It satisfies $f(x + y) = f(x) + f(y)$, $f(ax) = f(a)f(x)$, $f(1/x) = 1/f(x)$.

We can verify the example function $f(a + b\sqrt{2}) = a - b\sqrt{2}$ satisfies all the three conditions. The idea behind field automorphism is that, we can permute some elements in the field without any impact to the field structure.

Definition 3.3.4. *F-Automorphism*: If E is field extension of F , the automorphism f of field E also satisfy an extra condition, that for any element in F , equation $f(x) = x$ holds, then it is called *F-automorphism* of E .

The symmetry among roots is precisely defined under the *F-automorphism*. All the elements in F keep unchanged, while all the new elements, and only these new elements in filed extension E are changed. The changing among the elements in the field exactly means permutation, while the relations keep unchanged after permutation exactly means symmetry. For the example of $Q[\sqrt{2}]$, there are only two functions in Q -automorphism: one is the identity function $g(x) = x$, the other is f we defined above.

Let us summarize the result we obtained so far with the example $p(x) = x^2 - 2$:

1. The splitting field of $p(x)$ is $Q[\sqrt{2}]$;
2. The Q -automorphism defines the symmetry of the roots of $p(x)$. It contains two functions: $f(a + b\sqrt{2}) = a - b\sqrt{2}$, and $g(x) = x$.

However, we can't ensure the symmetry to all roots if only extend to the splitting field. For example, the splitting field of equation $x^4 - 5x + 6 = 0$ is $Q[\sqrt{2}, \sqrt{3}]$. Although there is automorphism f that swaps (permute) $\pm\sqrt{2}$, there does not exist Q -automorphism permutes $\sqrt{2}$ and $\sqrt{3}$. Otherwise, suppose there was $f(\sqrt{2}) = \sqrt{3}$, then $f(\sqrt{2})^2 = f(\sqrt{2}^2) = f(2) = 2$, because f preserves multiplicative structure and $f(x) = x$ for rational x . On the other hand, since $f(\sqrt{2}) = f(\sqrt{3})$, we have $f(\sqrt{2})^2 = \sqrt{3}^2 = 3$. It means $2 = 3$, which is clearly nonsense. Besides that, consider field extension $Q[x_1, \dots, x_n, \sqrt{x_1}]$, it contains the square root of x_1 , but does not contain the square root of x_2 . Therefore, there is not automorphism that permutes x_1 and x_2 . In order to obtain the complete symmetry, we need keep extending the field with $\sqrt{x_2}, \dots, \sqrt{x_n}$.

Theorem 3.3.1. For each radical extension E of $Q[x_1, \dots, x_n]$, there exists a bigger radical extension $E \subseteq \overline{E}$ with automorphisms σ extending all permutations of x_1, \dots, x_n .

With this, we connected the field extension with automorphism. If we put all the automorphisms together as a set, using composition as the binary operation, and let the identity function be the identity element, we obtain a group, named Galois group.

Definition 3.3.5. *Galois group*: For field extension E from F , there is a set G contains all the *F-automorphisms* of E . For any two *F-automorphisms* f and g in G , define the binary operation as $(f \cdot g)(x) = f(g(x))$. We define G as the *Galois group* of field extension E/F . Denoted as $\text{Gal}(E/F)$.

We still use $p(x) = x^2 - 2$ for example. Galois group $G = Gal(p) = \{f, g\}$, contains two elements, one is $f(a + b\sqrt{2}) = a - b\sqrt{2}$, the other is $g(x) = x$. Where the identity element is g . Let's verify if $f \cdot f = g$:

$$\begin{aligned}
 (f \cdot f)(a + b\sqrt{2}) &= f(f(a + b\sqrt{2})) && \text{Composition} \\
 &= f(a - b\sqrt{2}) && \text{Definition of } f \text{ for inner parenthesis} \\
 &= a + b\sqrt{2} && \text{Definition of } f \text{ again} \\
 &= g(a + b\sqrt{2}) && \text{Definition of identity function } g
 \end{aligned}$$

We do see that G is a group. It's isomorphic (identical) to a 2-cycle cyclic group, and is also isomorphic to the degree 2 symmetric group S_2 . This group can be written as $\{f, f^2\}$, since $f^2 = f \cdot f = 1$ is the identity element. It can also be written as the permutation group $\{(1), (1 2)\}$, where (1) keeps the two roots unchanged, and $(1 2)$ swaps the two roots.

It's a critical idea to study the equation through the symmetry of its roots in Galois theory. By using the concept of automorphism, we can give the definition symmetry: in mathematics, the symmetry of object is the automorphism to the object itself while preserving all of its structure.

Exercise 3.11

1. Prove that, for any polynomial $p(x)$ with rational coefficients, E/Q is the field extension, f is the Q -automorphism of E , then equation $f(p(x)) = p(f(x))$ holds.
2. Taking the complex number into account, what is the splitting field for polynomial $p(x) = x^4 - 1$? What are the functions in its Q -automorphism?
3. What's the Galois group for quadratic equation $x^2 - bx + c = 0$?
4. Prove that, if p is prime number, then Galois group for equation $x^p - 1$ is the $(p - 1)$ -cycle cyclic group C_{p-1} .

3.3.3 Fundamental theorem of Galois theory

We come to the center of Galois theory. Starting from the equation coefficient field F , we keep extending the fields to the splitting field $F \subset F_1 \subset F_2 \dots \subset E$. The corresponding Galois group is $Gal(E/F)$. Galois found, there was one to one correspondence between the Galois subgroups and its intermediate fields F_1, F_2, \dots in the reversed order.

Theorem 3.3.2. Fundamental theorem of Galois theory: *If E/F is field extension²⁶, G is the corresponding Galois group. There is one to one correspondence between subgroups of G and intermediate fields. If $F \subset L \subset E$, and $Gal(E/L) = H$, then H is the subgroup of $Gal(E/F)$.*

The reason why it's in reversed order is because the group gets smaller when extend the field. The starting point of field extension is F , the corresponding Galois group is $G = Gal(E/F)$; the end point is the splitting field E , while the corresponding group only contains one element, which is the identity permutation $\{1\}$. For the intermediate field L , the corresponding group is $H = Gal(E/L)$. It's a subgroup, $Gal(E/L) \subset Gal(E/F)$. If H is a normal subgroup (refer to previous section 3.1.7), then its quotient group is $G/H = Gal(L/F)$.

²⁶Strictly speaking, it should be Galois extension, we skipped the definition of Galois extension here.

Let us see a concrete example[36] (pp. 490). Consider equation $x^4 - 8x^2 + 15 = 0$, it can be factored as $(x^2 - 3)(x^2 - 5) = 0$. The coefficients are in rational field Q , the splitting field is $E = Q[\sqrt{3}, \sqrt{5}]$. The order of its Galois group $Gal(E/Q)$ is 4. It is isomorphic to a 4-cycle cyclic group. We can find 3 intermediate field extensions: $Q[\sqrt{3}]$, $Q[\sqrt{5}]$, and $Q[\sqrt{15}]$. The Galois subgroup orders corresponding to all these 3 intermediate fields are 2. While the Galois group of the equation, which is the 4-cycle cyclic group only has one subgroup of order 2. Besides that, it hasn't any other non-trivial subgroups. According to the fundamental theorem of Galois theory, we know there is no other field extension besides those 3 intermediate field extensions. We can consider it from another view point. All elements in the splitting field are in the form of $\alpha = a + b\sqrt{3} + c\sqrt{5} + d\sqrt{15}$, where a, b, c, d are rational numbers. For any elements in the 3 intermediate field extensions, among b, c, d , there are at least two are 0.

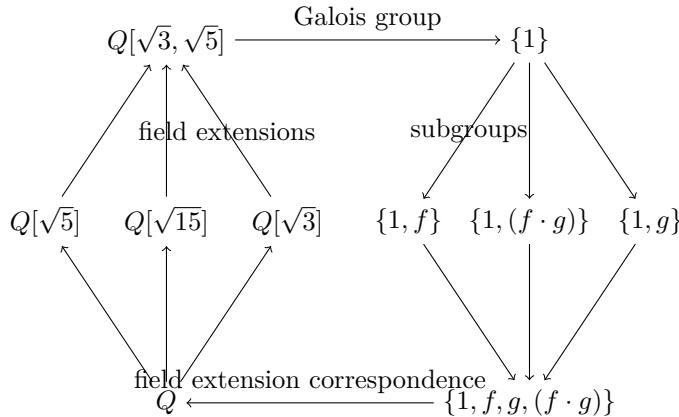


Figure 3.29: Galois correspondence

As shown in figure 3.29, any element in the splitting field can be written in form:

$$\begin{aligned}\alpha &= (a + b\sqrt{3}) + (c + d\sqrt{3})\sqrt{5} \\ &= a + b\sqrt{3} + c\sqrt{5} + d\sqrt{15}\end{aligned}$$

Where a, b, c, d are rational numbers. We define the following automorphisms:
Morphism f negates $\sqrt{3}$:

$$\begin{aligned}f((a + b\sqrt{3}) + (c + d\sqrt{3})\sqrt{5}) &= (a - b\sqrt{3}) + (c - d\sqrt{3})\sqrt{5} \\ &= a - b\sqrt{3} + c\sqrt{5} - d\sqrt{15}\end{aligned}$$

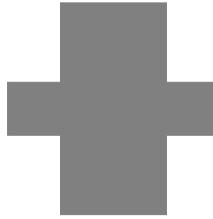
Morphism g negates $\sqrt{5}$:

$$\begin{aligned}g((a + b\sqrt{3}) + (c + d\sqrt{3})\sqrt{5}) &= (a + b\sqrt{3}) - (c + d\sqrt{3})\sqrt{5} \\ &= a + b\sqrt{3} - c\sqrt{5} - d\sqrt{15}\end{aligned}$$

The composite morphism $f \cdot g$ negates $\sqrt{3}$ and $\sqrt{5}$ at the same time:

$$\begin{aligned}(f \cdot g)((a + b\sqrt{3}) + (c + d\sqrt{3})\sqrt{5}) &= (a - b\sqrt{3}) - (c - d\sqrt{3})\sqrt{5} \\ &= a - b\sqrt{3} - c\sqrt{5} + d\sqrt{15}\end{aligned}$$

In addition to the identity morphism 1, for the base field Q , there are four elements in its Galois group: $G = \{1, f, g, (f \cdot g)\}$. This group is isomorphic to Klein four group, while the Klein group is the product of two 2-cycle cyclic groups. It has 5 subgroups, each is corresponding to a field extension:



Klein four group. Symmetric when flip horizontally, vertically, or flip at the same time.

- The subgroup only contains the identity element $\{1\}$, it corresponds to the splitting field $Q[\sqrt{3}, \sqrt{5}]$;
- G itself, corresponds to the rational field Q ;
- Order 2 subgroup $\{1, f\}$, corresponds to field extension $Q[\sqrt{5}]$. f only negates $\sqrt{3}$ and leaves $\sqrt{5}$ unchanged;
- Order 2 subgroup $\{1, g\}$, corresponds to field extension $Q[\sqrt{3}]$. g only negates $\sqrt{5}$ and leaves $\sqrt{3}$ unchanged;
- Order 2 subgroup $\{1, (f \cdot g)\}$, corresponds to field extension $Q[\sqrt{15}]$. $(f \cdot g)$ negates $\sqrt{5}$ and $\sqrt{3}$ at the same time, and leaves $\sqrt{15}$ unchanged.

We can also write the Galois group of this equation in the form of permutation group $\{(1), (1 2), (3 4), (1 2)(3 4)\}$. Where (1) keeps all the 4 roots unchanged; $(1 2)$ swaps $\pm\sqrt{3}$; $(3 4)$ swaps $\pm\sqrt{5}$; while $(1 2)(3 4)$ swaps these two pairs at the same time.

Through the fundamental theorem of Galois theory, we managed to convert the equation problem in field to an equivalent problem about permutation group. The last attack is to reveal the solvability essence with group.

3.3.4 Solvability

Through Galois correspondence, we connect the radical extension with group structure. To simplify the problem, we add some extra conditions. First, for every radical extension $F[\alpha_i]$, we assume the α_i being extended is a p -th root for some prime p . For example, we will factor $\sqrt[3]{\alpha}$ to $\sqrt[3]{\alpha} = \beta$ and $\sqrt[3]{\beta}$. Then extend the field two times with them one by one. Second, if α_i is p -th root, and the radical extension $F[\alpha_1, \dots, \alpha_{i-1}]$ does not contain p -th root of unity, we can assume the next radical extension $F[\alpha_1, \dots, \alpha_i]$ does not contain p -th root of unity too, unless α_i itself is a p -th root of unity. For example, when we want to extend the field with root $\alpha = \sqrt[3]{2}$, if the field F does not contain $\zeta = \frac{-1 + i\sqrt{3}}{2}$ as shown in figure 3.28, we can assume the radical extension $F[\sqrt[3]{2}]$ does not contain ζ too, unless $\alpha^3 = 1$, which means α itself equals ζ . If it isn't the case, we can adjoin ζ before α_i , then the radical extension $F[\alpha_1, \dots, \alpha_{i-1}, \zeta]$ contains all the p -th root of unity: $1, \zeta, \zeta^2, \dots, \zeta^{p-1}$. With both these modifications, the final field $F[\alpha_1, \dots, \alpha_k]$ is the same, and it remains the same if the newly adjoined root ζ are included in the list $\alpha_1, \dots, \alpha_k$.

Hence any radical extension $F[\alpha_1, \dots, \alpha_k]$ is a chain of ascending fields:

$$F = F_0 \subseteq F_1 \subseteq F_2 \subseteq \dots \subseteq F_k = F[\alpha_1, \dots, \alpha_k]$$

Where every $F_i = F_{i-1}[\alpha_i]$. α_i is the p -th root of some element in F_{i-1} , where p is prime. They satisfy the condition about root of unity above. Corresponding to this chain of radical extensions, we have a chain of descending groups:

$$Gal(F_k/F_0) = G_0 \supseteq G_1 \supseteq \dots \supseteq G_k = Gal(F_k/F_k) = \{1\}$$

Where $G_i = Gal(F_k/F_i) = Gal(F_k/F_{i-1}[\alpha_i])$. Every step moves from G_{i-1} to its subgroup G_i , corresponding to adjoin p -th root α_i to field F , where p is prime. Using the terms in group theory, G_i is the normal subgroup of the previous group G_{i-1} , while the quotient group G_{i-1}/G_i is abelian (commutative). We have the following theorem reflects this fact:

Theorem 3.3.3. *In field extension $B \subseteq B[\alpha] \subseteq E$, $\alpha^p \in B$ for some prime p , if $B[\alpha]$ contains no p -th roots of unity not in B unless α itself is a p -th root of unity, then $Gal(E/B[\alpha])$ is a normal subgroup of $Gal(E/B)$, and the quotient group*

$$Gal(E/B)/Gal(E/B[\alpha])$$

is abelian.

The proof is a bit complex, reader can skip it in the this frame

Proof. We'll use the homomorphism theorem for groups introduced in section 3.1.7 to prove this theorem. We need find a homomorphism of $Gal(E/B)$, with the kernel of $Gal(E/B[\alpha])$, into an abelian group (onto a subgroup of an abelian group, which of course is also abelian). The obvious map with kernel $Gal(E/B[\alpha])$ is restriction to field $B[\alpha]$, we denote this map as $|_{B[\alpha]}$. According to the definition of Galois group, the group elements are automorphisms:

$$\sigma \in Gal(E/B[\alpha]) \iff \sigma|_{B[\alpha]} \text{ is the identity map}$$

All such automorphisms σ in Galois group $Gal(E/B)$ satisfy the homomorphism property:

$$\sigma' \sigma|_{B[\alpha]} = \sigma' |_{B[\alpha]} \sigma|_{B[\alpha]}$$

Since σ fixes all elements in B , $\sigma|_{B[\alpha]}$ is completely determined by the value of $\sigma(\alpha)$. There are two cases. In the first case, α is a p -th root of unity ζ , then:

$$(\sigma(\alpha))^p = \sigma(\alpha^p) = \sigma(\zeta^p) = \sigma(1) = 1$$

Hence $\sigma(\alpha) = \zeta^i \in B[\alpha]$, since each p -th root of unity is some ζ^i . In the second case, α is not a root of unity, then:

$$(\sigma(\alpha))^p = \sigma(\alpha^p) = \alpha^p$$

According to radical extension rule, α^p is in field B , hence $\sigma(\alpha) = \zeta^j \alpha$ for some p -th root of unity ζ , and $\zeta \in B$, so again $\sigma(\alpha) \in B[\alpha]$. Thus $B[\alpha]$ is closed for every σ .

This also implies that $|_{B[\alpha]}$ maps $Gal(E/B)$ into $Gal(E/B[\alpha])$, so it now remains to check that $Gal(B[\alpha]/B)$ is abelian. There are two cases: In the first case, α is a root of unity, then every element in Galois group, which is automorphism $\sigma_i = \sigma|_{B[\alpha]} \in Gal(B[\alpha]/B)$ can be written in form $\sigma_i(\alpha) = \alpha^i$, hence:

$$\sigma_i \sigma_j(\alpha) = \sigma_i(\alpha^j) = \alpha^{ij} = \sigma_j \sigma_i(\alpha)$$

In the second case, α is not a root of unity, then each group element, the automorphism σ_i is of the form $\sigma_i(\alpha) = \zeta^i \alpha$, hence:

$$\sigma_i \sigma_j(\alpha) = \sigma_i(\zeta^j \alpha) = \zeta^{i+j} \alpha = \sigma_j \sigma_i(\alpha)$$

Since $\zeta \in B$, therefore ζ is fixed under the automorphism. Hence in either case, $Gal(B[\alpha]/B)$ is abelian. \square

When observe the chain of Galois subgroups:

$$Gal(F[\alpha_1, \dots, \alpha_k]/F) = G_0 \supseteq G_1 \supseteq \dots \supseteq G_k = \{1\}$$

Along this chain, if every group G_i is the normal subgroup of the previous group G_{i-1} , and every quotient group G_{i-1}/G_i is abelian, then we say the Galois group $Gal(F[\alpha_1, \dots, \alpha_k]/F)$ is **solvable**.

Now let us see the general n -th degree equation:

$$x^n - a_1 x^{n-1} + a_2 x^{n-2} \dots \pm a_n = 0$$

The base field contains coefficients is $Q[a_1, \dots, a_n]$. Let its n roots (not necessarily be radical) be x_1, \dots, x_n , which means:

$$(x - x_1) \dots (x - x_n) = x^n - a_1 x^{n-1} \dots \pm a_n$$

The splitting field of the equation is $Q[x_1, \dots, x_n]$. If whatever radical extension we obtain, it can't contain the splitting field, then the equation is not radical solvable.

Theorem 3.3.4. *When $n \geq 5$, any radical extension of $Q[a_1, \dots, a_n]$ does not contain the splitting field $Q[x_1, \dots, x_n]$.*

Proof. We use the reduction to absurdity method. Suppose some radical extension E contains the splitting field $Q[x_1, \dots, x_n]$. According to the theorem we proved previously, there exists a bigger radical extension $\bar{E} \supseteq E$, such that the corresponding Galois group $G_0 = Gal(\bar{E}/Q[a_1, \dots, a_n])$ includes automorphisms of permutations of all roots. Hence we can construct a chain of Galois subgroups:

$$G_0 \supseteq G_1 \supseteq \dots \supseteq G_k = \{1\}$$

Where each G_i is the normal subgroup of the previous group G_{i-1} , and the quotient group G_{i-1}/G_i is abelian. We now show that this is impossible. Since the normal subgroup G_i is the kernel of the homomorphism of G_{i-1} , for any two automorphisms σ, τ in G_{i-1} , we have:

$$\sigma^{-1} \tau^{-1} \sigma \tau \in G_i$$

When $n \geq 5$, G_0 contains the permutations of all roots. It is isomorphic to symmetric group S_n , it must include 3-cycle permutation $(a \ b \ c)$. Suppose G_{i-1} also include 3-cycle permutation, we have:

$$(a \ b \ c) = (d \ a \ c)^{-1} (c \ e \ b)^{-1} (d \ a \ c) (c \ e \ b) \in G_i$$

Where a, b, c, d, e are distinct. It means G_i includes 3-cycle permutation as well. According to mathematical induction, all groups in the chain include 3-cycle permutation. But this is impossible, because the last group in this chain is $\{1\}$, it does not include 3-cycle permutation. \square

Thus we proved the general 5th degree equation is not radical solvable. We can explain from another perspective. The symmetric group S_5 has 120 elements, its only normal subgroup is A_5 . Where A_5 is called alternating group, which contains 60 elements.



Symmetric group S_5 is the production of alternating group A_5 and 2-cycle cyclic group. It can describe the symmetry of a football

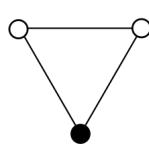
The only normal subgroup of A_5 is $\{1\}$. Hence the subgroup chain is $S_5 \supset A_5 \supset \{1\}$. However, the quotient group $A_5/\{1\}$ is not abelian (not commutative). Therefore, it is not solvable group. The corresponding general 5th degree equation is not radical solvable.

For the general 4th degree equation, the symmetric group S_4 has a normal subgroup A_4 , and the alternating group A_4 has a normal subgroup:

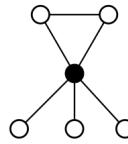
$$\{(1), (12)(34), (13)(24), (14)(23)\}$$

This group is isomorphic to Klein group, which is solvable.

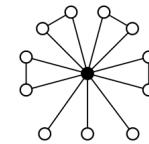
For the general cubic equation, the symmetric group S_3 has a normal subgroup A_3 , and the alternating group A_3 is isomorphic to 3-cycle cyclic group. Hence it is also solvable.



(a) Alternating A_3 is isomorphic to 3-cycle cyclic group



(b) Symmetric group S_3 , the upper part is A_3



(c) Alternating group A_4

Figure 3.32: Symmetric group and alternating group. Black point is identity element, cycle is closed loop

We introduced Galois theory in this short section. Galois theory is a powerful tool in abstract algebra. It can solve many problems, like to prove the fundamental theorem of algebra, prove Gauss's finding that the regular heptadecagon (17-sided polygon) can be constructed with straightedge and compass. Galois theory can also prove that the three classic geometric problems with straightedge and compass in ancient Greek, squaring the circle, trisecting the angle, and doubling the cube, are all impossible. We hope that readers can appreciate the depth and beauty of abstract thinking through this chapter.

Exercise 3.12

1. The 5th degree equation $x^5 - 1 = 0$ is radical solvable. What's its Galois subgroup chain?

3.4 Further reading

There are many textbooks introduce about abstract algebra. *Groups and Symmetry* by Armstrong is a good book about basic concept of groups and how it related and define symmetry in mathematics. *Algebra* by Micheal Artin is also a widely used textbook. It introduces Galois theory in the last chapter. *Galois Theory* by Emil Artin published in 1944 is the classic book. It was republished several times in 1998 and 2007. Artin was the important expositor. He gave the modern form of Galois theory. Harold M. Edwards introduced the development history in his *Galois Theory*, and took the classic way to explain this topic.

Chapter 4

Category

Mathematics is the art of giving the same name to different things.

—Henri Poincaré

Welcome to the world of category! We just see the wunderful scence about abstract algebra in previous chapter. Congratulation to enter the door to the new kingdom of abstraction. The road to this kingdom is developed by may talent minds. In the first stage, people abstracted the concept of number and shape from the concrete things; in the second stage, we removed the meaning of numbers, shapes, and concrete arithmatics, abstract them to algebraic structures (for example group) and relations (for example isomorphism); category theory can be considered as the third stage.

What is category? Why does it matter? Any relationship between category and programming? Category theory was a ‘side product’ when mathematicians studied homological algebra in 1940s. In recent decades, category theory has been widely used in varies of areas. Because of the powerful abstraction capability, it is applicable to many problems. It may also be used as an axiomatic foundation for mathematics, as an alternative to set theory and other proposed foundations. Category theory has practical applications in programming language theory. More and more mainstream programming languages adopted category concepts. The usage of monads is relized in more than 20 languages^[39]. Speaking in language of category, a monad is a monoid in the category of endofunctors. Most of the foundamental computations have been abstracted in this way nowadays¹.

The most important thing is abstraction. Hermann Weyl said “Our mathematics of the last few decades has wallowed in generalities and formalizations.” The samething happens in programming. The problems in modern computer science are challenging us with the complexicity we never seen before. Big-data, distributed computation, high concurrency, as well as the requirement to consistency, security, and integrity. We can’t catch them up

¹For example, the traditional generic folding from right:
`foldr _ z [] = z`
`foldr f z (x:xs) = f x (foldr f z xs)`
is written in the language of categories as: `foldr f z t = appEndo (foldMap (Endo . f) t) z`
We'll explain it later in this chapter.



Escher, Dewdrop, 1948

in the traditional way, like brute-force exhaustive search, pragmatic engineering practice, a smart idea plus some luck. It forces us to learn the new methods and tools from other science and mathematics.

As Dieudonne said: “This abstraction which has accrued in no way sprang from a perverse desire among mathematicians to isolate themselves from the scientific community by the use of a hermetic language. Their task was to find solutions for problems *handed down by the ‘classical’ age*, or arising directly from new discoveries in physics. They found that this was possible, but only through the *creation* of new objets and new methods, whose abstract character was *indispensable* to their success.” [38](page 2)

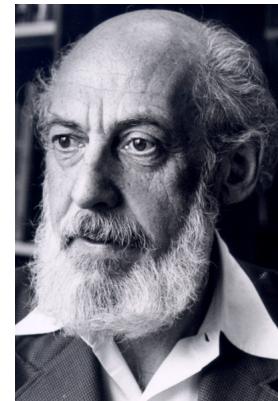
Category theory was developed by mathematician Samuel Eilenberg and Saunders Mac Lane in 1940s.

Samuel Eilenberg born in Warsaw, Kingdom of Poland to a Jewish family. His father is a brewer. Eilenberg studied at the University of Warsaw. A remarkable collection of mathematicians were on the staff there. He earned his Ph.D. from University of Warsaw in 1936. The second mathematical centre in Poland at that time was Lvov. It was there that Eilenberg met Banach, who led the Lvov mathematicians. He joined the community of mathematicians working and drinking in the Scottish Café and he contributed problems to the Scottish Book, the famous book in which the mathematicians working in the Café entered unsolved problems. In 1939 Eilenberg’s father convinced him that the right course of action was to emigrate to the United States. Once there he went to Princeton. This was not too long in coming and, in 1940, he was appointed as an instructor at the University of Michigan. In 1949 André Weil was working at the University of Chicago and he contacted Eilenberg to ask him to collaborate on writing about homotopy groups and fibre spaces as part of the Bourbaki project. Eilenberg became a member of the Bourbaki team. He wrote the 1956 book *Homological Algebra* with Henri Cartan. Eilenberg mainly studied algebraic topology. He worked on the axiomatic treatment of homology theory with Norman Steenrod, and on homological algebra with Saunders Mac Lane. In the process, Eilenberg and Mac Lane created category theory. Eilenberg spent much of his career as a professor at Columbia University. Later in life he worked mainly in pure category theory, being one of the founders of the field. He was awarded Wolf prize in 1987 for his fundamental work in algebraic topology and homological algebra. Eilenberg died in New York City in January 1998.

Eilenberg was also a prominent collector of Asian art. His collection mainly consisted of small sculptures and other artifacts from India, Indonesia, Nepal, Thailand, Cambodia, Sri Lanka and Central Asia. He donated more than 400 items to the Metropolitan Museum of Art in 1992[40].

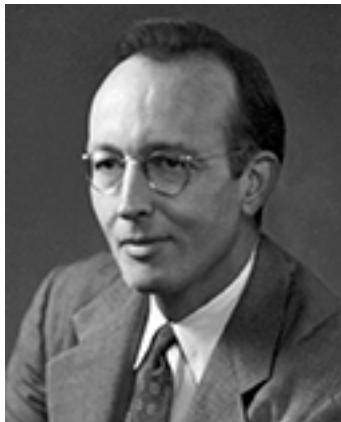
Saunders Mac Lane was born in Norwich, Connecticut in United State in 1909. He was christened “Leslie Saunders MacLane”, but “Leslie” was later removed because his parents dislike it. He began inserting a space into his surname because his wife found it difficult to type the name without a space.

In high school, Mac Lane’s favorite subject was chemistry. While in high school, his father died, and he came under his grandfather’s care. His half-uncle helped to send him to Yale University, and paid his way there beginning in 1926. His mathematics teacher, Lester S. Hill, coached him for a local mathematics competition which he won, setting the direction for his future work. He studied mathematics and physics as a double



Samuel Eilenberg, 1913 - 1998

major, and graduated from Yale with a B.A. in 1930. In 1929, at a party of Yale football supporters in New Jersey, Mac Lane was awarded a prize for having the best grade point average yet recorded at Yale. He met Robert Maynard Hutchins, the new president of the University of Chicago, who encouraged him to go there for his graduate studies[41]. Mac Lane Joined University of Chicago². At the University of Chicago he was influenced by Eliakim Moore, who was nearly seventy years old. He advised Mac Lane to study for a doctorate at Göttingen in Germany certainly persuaded Mac Lane to work at the foremost mathematical research centre in the world at that time. In 1931, having earned his master's degree, Mac Lane earned a fellowship from the Institute of International Education and became one of the last Americans to study at the University of Göttingen prior to its decline under the Nazis.



Saunders Mac Lane, 1909 - 2005

At Göttingen, Mac Lane studied with Paul Bernays and Hermann Weyl. Before he finished his doctorate, Bernays had been forced to leave the university because he was Jewish, and Weyl became his main examiner. Mac Lane also studied with Gustav Herglotz and Emmy Noether. In 1934, he finished his doctor degree and returned to United State³.

In 1944 and 1945, Mac Lane also directed Columbia University's Applied Mathematics Group, which was involved in the war effort. Mac Lane was vice president of the National Academy of Sciences and the American Philosophical Society, and president of the American Mathematical Society. While presiding over the Mathematical Association of America in the 1950s, he initiated its activities aimed at improving the teaching of modern mathematics. He was a member of the National Science Board, 1974–1980, advising the American government. In 1976, he led a delegation of mathematicians to China to study the conditions affecting mathematics

there. Mac Lane was elected to the National Academy of Sciences in 1949, and received the National Medal of Science in 1989.

Mac Lann's early work was in field theory and valuation theory. In 1941, while giving a series of visiting lectures at the University of Michigan, he met Samuel Eilenberg and began what would become a fruitful collaboration on the interplay between algebra and topology. He and Eilenberg originated category theory in 1945.

Mac Lane died on April 14th, 2005 in San Francisco, California, USA.

4.1 Category

Let's use an example of ecosystem to understand category. There are varies of living species on African grassland, like lion, hyena, cheetah, antelope, buffalo, zebra, vulture, lizard, cobra, ant, ... We call these animals objects. Every type of animals and plants is an object. There are relation among these living things, for example, lion eats antelope, antelope eats grass. We can easily form a food chain by drawing an arrow from antelope to lion, and another arrow from grass to antelope. Hence these objects together with the arrows form a structured and connected system.

²Hutchins soon offered Mac Lane a scholarship after the party. But Mac Lane neglected to actually apply to the program, but showed up and was admitted anyway.

³Within days of finishing his degree, Mac Lane married Dorothy Jones, from Chicago, who had typed his thesis. It was a small ceremony followed by a wedding dinner with a couple of friends in the Rathaus

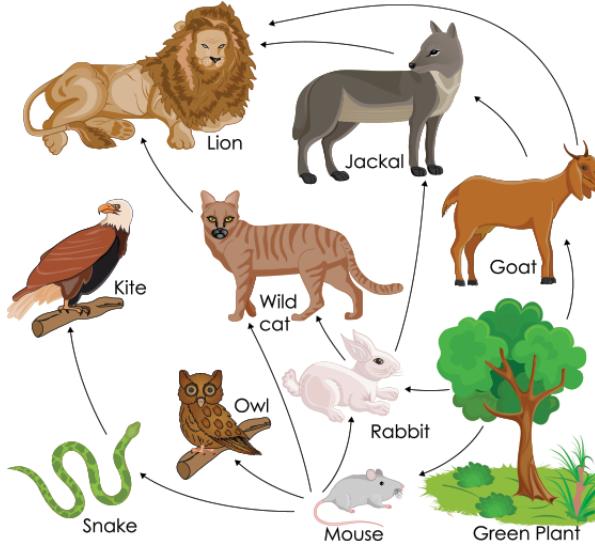
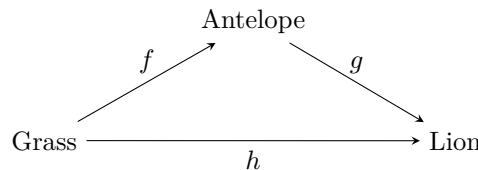


Figure 4.4: Living objects and food chain arrows for a structured system

To become a category, this system has to satisfy two conditions. First, every object must have an arrow to itself. Such special arrow is called the identity arrow. For the animals and plants on grassland, the arrow from lion to antelope means the antelope is downstream to the lion along the foodchain. We can define that every species is downstream to itself along the foodchain (it does not mean one eats itself, but means one does not eat itself), hence every species has an arrow points to itself. Second, arrows need be composable. What does composable mean? There is an arrow f from grass to antelope, an arrow g from antelope to lion. We can compose them to $g \circ f$ (read as g after f), which means grass is downstream to lion along the food chain. This composed arrow can be further composed with the third one. For example, vulture eats the dead lion, thus we can draw an arrow h from the lion to the vulture. Composing them together gives $h \circ (g \circ f)$. This arrow means grass is downstream to vulture along the food chain. It is identical to the arrow $(h \circ g) \circ f$. Hence the upstream, downstream relations are associative.

Besides composable, there is another concept called commutative. As shown in below diagram:



There are two paths from grass to lion. One is the composite arrow $g \circ f$, which means grass is downstream to antelope, and antelope is downstream to lion; the other is arrow h , which means grass is downstream to lion. Hence we obtain a pair of parallel arrows:

$$\begin{array}{ccc} \text{Grass} & \xrightarrow{g \circ f} & \text{Lion} \\ & \xrightarrow{h} & \end{array}$$

These two arrows may or may not be the same, when they are, we say they are commutative:

$$h = g \circ f$$

and the grass, antelope, lion triangle commutes. We say the living things in the grassland form a category under the food chain arrows. From this example, we can give the formal definition of category.

Definition 4.1.1. A category \mathbf{C} consists of a collection of objects⁴, denoted as A, B, C, \dots , and a collection of arrows, denoted as f, g, h, \dots . There are four operations defined on top of them:

- Two total operations⁵, called source and target⁶. They assign source and target objects to an arrow. Denoted as $A \xrightarrow{f} B$. It means the source of arrow f is A , and the target is B ;
- The third total operation is called identity arrow⁷. For every object A , the identity arrow points to A itself. Denoted as $A \xrightarrow{id_A} A$;
- The fourth operation is a partial operation, called composition. It composes two arrows. Given arrows $B \xrightarrow{f} C$ and $A \xrightarrow{g} B$, the composite of f and g is $f \circ g$, read as ‘ f after g ’. It means $A \xrightarrow{f \circ g} C$.

Besides these four operations, category satisfies the following two axioms:

- **Associativity:** Arrows are associative. For every three composable arrows f, g, h , the equation:

$$f \circ (g \circ h) = (f \circ g) \circ h$$

holds. We can write it as $f \circ g \circ h$.

- **Identity element:** The identity arrow is the identity element for the composition. For every arrow $A \xrightarrow{f} B$, equations:

$$f \circ id_A = f = id_B \circ f$$

hold.

Categories and arrows are abstract compare to the foodchain system on the grassland. Let's use some concrete examples to understand this definition.

⁴It has nothing related to the object oriented programming. Here the object means abstract thing.

⁵Total operation is defined for every object without exception. Its counterpart is partial operation, which is not defined from some objects. For example, the negate $x \mapsto -x$ is total operation for natural numbers, while the inverse operation $x \mapsto 1/x$ is not defined for 0, hence it is a partial operation.

⁶They should be treated as verb, which means assign source object as... and assign target object as...

⁷Similarly, identity arrow should be treated as verb, which means assign the identity arrow as...

4.1.1 Examples of categories

In mathematics, a set is called a monoid if it is defined with a special identity element, and associative binary operation. The integers for example, with 0 as the identity element, and plus as the binary operation, form a monoid called integer additive monoid.

A monoid can contain other things besides numbers. Consider the set of English words and sentences (called strings in programming). We can define the binary operation that concatenates two English strings. For example “red” + “apple” = “red apple”. These English strings form a monoid, where the identity element is the empty string “”. It’s easy to verify the monoid conditions:

$$\text{red} + (\text{apple} + \text{tree}) = (\text{red} + \text{apple}) + \text{tree}$$

The string concatenation is associative.

$$"" + \text{apple} = \text{apple} = \text{apple} + ""$$

Concatenating any string with empty string (identity element) gives itself.

Put the monoid of strings aside, let’s consider another monoid. The elements are sets of characters. The binary operation is the set union; the identity element is the empty set. Union of two character sets gives a bigger set, for example:

$$\{a, b, c, 1, 2\} \cup \{X, Y, Z, 0, 1\} = \{a, b, c, X, Y, Z, 0, 1, 2\}$$

Together with the first example, the integer additive monoid, we have three monoids on hand. Let us setup the transforms among them⁸. First is the map from monoid of English strings to monoid of character sets. For any given English word or sentence, we can collect all the unique characters used in it to form a set. Let’s call this operation “chars”. We can verify this operation satisfies the following:

$$\text{chars}(\text{red} + \text{apple}) = \text{chars}(\text{red}) \cup \text{chars}(\text{apple}) \quad \text{and} \quad \text{chars}("") = \emptyset$$

It means, the unique characters used in a concatenated English string are same as the union result of the character sets for each one; empty string does not contain any characters (corresponding to the empty set). This is a strong property named *homomorphism* defined in previous chapter.

Next, we define a transform from the character sets monoid to integer additive monoid. For every given character set, we can count how many characters there are. Empty set contains 0 character. We name this operation “count”⁹.

So far, we defined two transforms: “chars” and “count”, let us see how they composite:

$$\text{String} \xrightarrow{\text{chars}} \text{Character set} \xrightarrow{\text{count}} \text{Integer}$$

and

$$\text{String} \xrightarrow{\text{count} \circ \text{chars}} \text{Integer}$$

Obviously, the composite result “count \circ chars” is also a transform. It means we first collect the unique characters from English string, then count the number of characters. We obtain the *monoid category* **Mon**. The objects are varies of monoids, arrows are the transforms among them. The identity arrow is from a monoid to itself.

⁸The transform is often called morphism in mathematics

⁹Although ‘count’ is not homomorphic, for example: $|\{r, e, d\}| + |\{a, p, l, e\}| = 3 + 4 = 7 \neq |\{r, e, d\} \cup \{a, p, l, e\}| = |\{a, e, d, l, p, r\}| = 6$, it is a valid transform from set to integer.

This is a very big category. It contains *all* the monoids in the universe¹⁰. On the other hand, category can also be small. Let's see another example, a category that contains only one monoid. Consider the monoid of English strings. There is only one object. It doesn't matter what this object is. It can be any fixed set, or even the set of all English strings (I am sorry that this set does not look small). For any English string, for example 'hello', we can define a prefix operation, that prepend 'hello' to any other English strings. We call this operation as 'prefix hello'. Apply 'prefix hello' to word 'Alice' gives result 'helloAlice'; Similarly, we can define 'prefix hi', when applies to 'Alice' gives 'hiAlice'. If we define them as two different arrows, then their composition is:

$$\text{prefix hello} \circ \text{prefix hi} = \text{prefix hellohi}$$

It means first prepend prefix 'hi', then prepend prefix 'hello'. It's easy to verify that, any three such arrows are associative. For the identity element, any string keeps same when prepended with empty string as prefix, hence it is the identity arrow. Now we have a monoid category with only one object, as shown in figure 4.5. Every arrow in the category is an element in the monoid, the arrow composition is given by monoid binary operation, and the identity arrow is the identity element in the monoid.

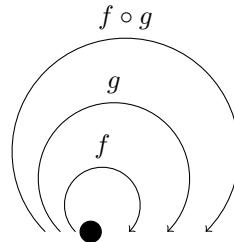


Figure 4.5: The monoid category with only one object

For monoid, we see both the category **Mon**, that contains all the monoids in the universe, and the category with only one monoid. It is like a hidden world in a piece of sand. What's more interesting, for any object A in a category \mathbf{C} , define set $\text{hom}(A, A)$ as all the arrows from A to A itself. Then this set of arrows forms a monoid under composite operation, where the identity element is the identity arrow. Such symmetry emerges surprisingly.

Our second example is set. Let every set be an object¹¹, the arrow is a function (or map) from one set A to another set B . We call A the domain of definition, and B is the domain of the function value¹². The composition is function composite. For functions $y = f(x)$ and $z = g(y)$, their composition is $z = (g \circ f)(x) = g(f(x))$. It's easy to verify that function composition is associative. The identity element is the identity function $\text{id}(x) = x$. Hence we obtain the category **Set** of all sets and functions.

Our third example contains a pair of concepts. *partial order set* and *pre-order set*. Given a set, pre-order means we can compare every two elements in it. We often use the symbol \leq to represent this binary relation. It does not necessarily mean less than or equal relation between numbers, it can mean one set is subset of the other, one string is post-fix of the other, one person is descendant of the other etc. If the relation \leq satisfies the following conditions, we call it a pre-order:

¹⁰You may think about the Russel's paradox. Strictly speaking, **Mon** contains all 'small' monoids in the universe

¹¹Whenever consider the set of all sets, it ends up of Russel's paradox 'the set all sets that does not contain itself'. We'll revisit Russel's paradox in chapter 7.

¹²It is total function, which can be applied to every elements in domain A .

- **reflexive:** For every element a in the set, we have $a \leq a$;
- **transitive:** If $a \leq b$ and $b \leq c$, then $a \leq c$;

On top of these two, if the relation is also anti-symmetric, we call it partial order:

- **anti-symmetric:** if $a \leq b$ and $b \leq a$, then $a = b$;

We define the set satisfies pre-order, and the set satisfies partial order as:

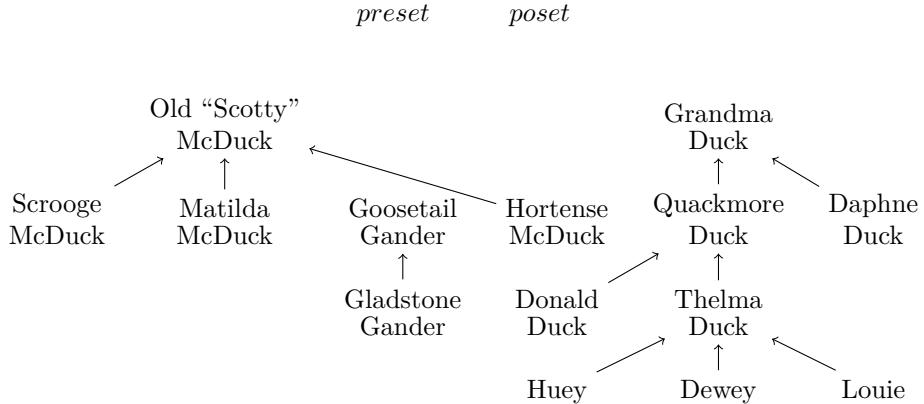


Figure 4.6: Duck family tree

Not every two elements are comparable in partial order set. The figures in Duck Donald family form a partial order set under the descendant relation, as shown in figure 4.6. We can see $\text{Donald} \leq \text{Quackmore}$, but there is no \leq relation between Huey and Donald , or between Donald and Scrooge . Although every one has its ancestor in this family tree (The figures at the root can be considered as the ancestor of themselves according to reflexive rule), but the figures at the same level or in different branches are not comparable.

As shown in 4.7, for a given set $\{x, y, z\}$, all its subsets form a partial order set under the inclusion relationship. Although every element (a subset) has a subset, elements at the same level are not comparable. Besides that there are also non-comparable elements, like $\{x\}$ and $\{y, z\}$ for example.

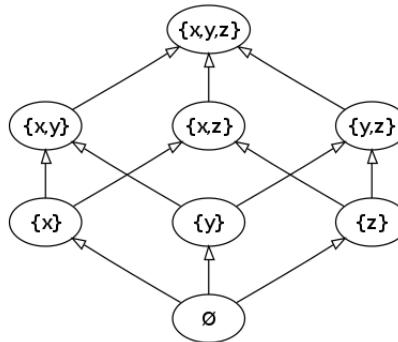


Figure 4.7: All subsets form a partial order set under inclusion relationship.

In general, any partial order set is a pre-order set, but the reverse is not necessarily true. A pre-order set may not be partial order set. We learn monotone function in

high school math. Monotone function never decreases, if $x \leq y$, then $f(x) \leq f(y)$. By composing monotone functions on top of partial order set or pre-order set, we obtain a pair of categories.

Pre **Pos**

The objects are all *presets* and *posets*, the arrows are monotone maps. As the identity map is also monotone, it is the identity arrow in the category.

We intend to choose categories of monoids and pre-set as two examples. They are the two most simple categories. The study of monoids is the study of composition in the miniature. The study of presets is the study of comparison in the miniature. It is the core of the entire category to study the composition and comparison of mathematical structures. In a sense every category is an amalgam of certain monoids and presets([42], p13).

Pre is a big category that contains all the pre-order sets in the world. On the other hand, preset category can also be very small, that only contains one pre-order set. Consider the set of elements i, j, k, \dots , if i and j are comparable, and $i \leq j$, we define

$$i \longrightarrow j$$

Hence for every two objects, either there is no arrow between them, which means they are not comparable; or there exists one, which means there is \leq relationship. In summary, there is at most one arrow between any two objects. We can verify that such arrows are composable, and for every element i , relation $i \leq i$ holds, hence there exists self-pointed arrow. Such preset itself forms a category. Again, we see the symmetry between preset and monoid categories. Monoid category contains only one object, but has many arrows; while preset category contains many objects, but has at most one arrow between objects.

Exercise 4.1

1. Prove that the identity arrow is unique (hint: refer to the uniqueness of identity element for groups in previous chapter).
2. Verify the monoid (S, \cup, \emptyset) (the elements are sets, the binary operation is set union, the identity element is empty set) and $(N, +, 0)$ (elements are natural numbers, the binary operation is add, the identity element is zero) are all categories that contain only one object.
3. In chapter 1, we introduced Peano's axioms for natural numbers and the isomorphic structures to Peano arithmetic, like the linked-list etc. They can be described in categories. This was found by German mathematician Richard Dedekind although the category theory was not established by his time. We named this category as Peano category, denoted as **Pno**. The objects in this category is (A, f, z) , where A is a set, for example natural numbers N ; $f : A \rightarrow A$ is a successor function. It is *succ* for natural numbers; $z \in A$ is the starting element, it is zero for natural numbers. Given any two Peano objects (A, f, z) and (B, g, c) , define the morphism from A to B as:

$$A \xrightarrow{\phi} B$$

It satisfies:

$$\phi \circ f = g \circ \phi \quad \text{and} \quad \phi(z) = c$$

Verify that **Pno** is a category.

4.1.2 Arrow \neq function

In most examples so far, arrows are either functions, or function like things, such as maps or morphisms. It gives us an illusion that arrows mean function like things. The next example helps us to realize such deception. There is a relation category. The objects are sets. The arrow from set A to set B , $A \xrightarrow{R} B$ is defined as:

$$R \subseteq B \times A$$

Let's see what does it mean. Set $B \times A$ represents all element combinations from B and A . It is called the product of B and A :

$$B \times A = \{(b, a) | b \in B, a \in A\}$$

Use the Donald Duck family for example, set $A = \{ \text{Scrooge McDuck, Matilda McDuck, Hortense McDuck} \}$ contains three members in McDuck family, set $B = \{\text{Goosetail Gander, Quackmore Duck}\}$ contains two external members, then the product of $B \times A$ is $\{(\text{Goosetail Gander, Scrooge McDuck}), (\text{Goosetail Gander, Matilda McDuck}), (\text{Goosetail Gander, Hortense McDuck}), (\text{Quackmore Duck, Scrooge McDuck}), (\text{Quackmore Duck, Matilda McDuck}), (\text{Quackmore Duck, Hortense MuDuck})\}$. Set R is a subset of $B \times A$, it represents some relation between A and B . If element a in A , and element b in B satisfy relation R , then $(b, a) \in R$, we denote it as bRa . For this example, we can let $R = \{(\text{Goosetail Gander, Matilda McDuck}), (\text{Quackmore Duck, Hortense MuDuck})\}$, then R means they are couples (In Donald Duck story, Goosetail married Matilda, they adopted Gladstone as their child; Quackmore married Hortense, they are parents of Donald).

Hence the set of *all* arrows from A to B represents all possible relations from A to B . Let's consider the arrow composition:

$$A \rightarrow B \rightarrow C$$

If there exists an element b in some intermediate set, that both relations bRa and cSb hold, we say there is composition between arrows. Use the Donald Duck family for example. Let set $C = \{\text{Gladstone, Donald, Thelma}\}$. Relation $S = \{(\text{Donald, Quackmore}), (\text{Thelma, Quackmore})\}$ means son and father. Thus the composite arrow $S \circ R$ gives result $\{(\text{Donald, Hortense}), (\text{Thelma, Hortense})\}$, it means the relation that c is the son of mother a . Hence both relations are satisfied through Quackmore, so that Donald and Thelma are sons of their mother Hortense. The definition of identity arrow is simple, every element has an identical relation to itself.

We can generate a new category from any existing category, for example, by reversing the arrows in category \mathbf{C} , we obtained a dual *opposite* category \mathbf{C}^{op} . Hence when we understand a category, we understand its dual category as well.

4.2 Functors

We mentioned category theory is the ‘second’ level abstraction on top of algebraic structure. We’ve seen how to abstract all sets and maps, groups and morphisms, posets and monotone functions into categories. The next question is how to bridge these categories and compare them? Functor¹³ is used to compare categories and their inner relations (objects and arrows).

¹³Some C++ programming language materials use functor to name function object. It has nothing related to category theory.

4.2.1 Definition of functor

In some sense, functor can be considered as the transform (morphism) between categories. However, it does not only map the object from one category to the other, but also maps the arrow. This makes functor different from the normal morphism (between groups for example).

We often use \mathbf{F} to denote functor. Since functor likes the morphism for category, it must faithfully preserve the structure and relations between categories. In order to achieve this, a functor need satisfy two conditions:

1. Any identity arrow in one category is transformed to identity arrow in another category. As shown in below figure:

$$A \xrightarrow{id} A \quad \mapsto \quad \mathbf{F}A \xrightarrow{id} \mathbf{F}A$$

2. Any composition of arrows in one category is transformed to composition in another category¹⁴.

$$\begin{array}{ccc}
 \begin{array}{c} f \nearrow B \\ A \xrightarrow{g \circ f} C \\ g \searrow \end{array} & \mapsto & \begin{array}{c} \mathbf{F}(f) \nearrow \mathbf{F}B \\ \mathbf{F}A \xrightarrow{\mathbf{F}(g \circ f)} \mathbf{F}C \\ \mathbf{F}(g) \searrow \end{array}
 \end{array}$$

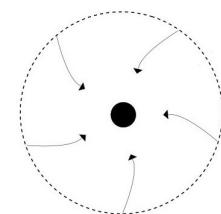
$$\mathbf{F}(g \circ f) = \mathbf{F}(g) \circ \mathbf{F}(f)$$

In summary, functor preserves identity morphisms and composition of morphisms. Functors can also be composed, for example functor $(\mathbf{FG})(f)$ means first apply \mathbf{G} on arrow f , then apply \mathbf{F} on top of the arrow in the new category.

4.2.2 Functor examples

Let's use some examples to understand functor. If a functor maps a category to itself, it is called *endo-functor*¹⁵. The simplest functor is the identity functor, which is an endo-functor, denoted as $id : \mathbf{C} \rightarrow \mathbf{C}$. It can be applied to any category, maps object A to A , and maps arrow f to f .

The second simplest functor is the constant functor. It acts like a blackhole. We denote it as $\mathbf{K}_B : \mathbf{C} \rightarrow \mathbf{B}$. It can be applied to any category, maps all objects to the blackhole object B , and maps all arrows to the identity arrow in the blackhole id_B . The blackhole category has only one identity arrow, it also satisfy the arrow composition condition: $id_B \circ id_B = id_B$.



Constant functor acts like a blackhole.

Example 4.2.1. Maybe functor. Computer scientist, Tony Hoare (Sir Charles Antony Richard Hoare), who developed the quicksort algorithm, and awarded ACM Turing award in 1980, had an interesting speaking apologised for inventing the null reference¹⁶.

¹⁴There are two different types of transformation, one is called covariance, the other is contravariance. The two terms are also used in programming language type system. We only consider covariance in this book

¹⁵Similar to the automorphism in abstract algebra, which we introduced in previous chapter.

¹⁶At a software conference called QCon London in 2009



Tony Hoare

I call it my billion-dollar mistake. It was the invention of the null reference in 1965. At that time, I was designing the first comprehensive type system for references in an object oriented language (ALGOL W). My goal was to ensure that all use of references should be absolutely safe, with checking performed automatically by the compiler. But I couldn't resist the temptation to put in a null reference, simply because it was so easy to implement. This has led to innumerable errors, vulnerabilities, and system crashes, which have probably caused a billion dollars of pain and damage in the last forty years[43].

After 2015, the mainstream programming environments gradually adopted the maybe concept to replace the error prone null¹⁷.

Below diagram illustrates how **Maybe** behaves.

$$\begin{array}{ccc}
 A & \longrightarrow & \mathbf{Maybe} \ A \\
 f \downarrow & & \downarrow \mathbf{Maybe}(f) \\
 B & \longrightarrow & \mathbf{Maybe} \ B
 \end{array}$$

The left side objects A and B can be data types, like integer, Boolean. While the right side objects are mapped types through functor **Maybe**. If A represents *Int*, then the right side corresponds to **Maybe Int**, if B represents *Bool*, then the right side corresponds to **Maybe Bool**. How does the maybe functor map objects? that part is defined as:

data **Maybe** $A = \mathbf{Nothing} \mid \mathbf{Just} \ A$

It means, if the object is type A , then the mapped object is type **Maybe A**. Note the object is type, but not value. The value of **Maybe A** is either empty, denoted as *Nothing*, or a value constructed by *Just*.

Use type *Int* for example, through the maybe functor, it is mapped to **Maybe Int**. The value could be *Nothing* or *Just 5* for instance.

Consider a binary search tree, that contains elements of type A . When search a value in the tree, it may not be found, therefore the type of search result is **Maybe A**¹⁸.

$$\begin{aligned}
 \text{lookup } \text{Nil } _ &= \mathbf{Nothing} \\
 \text{lookup } (\text{Br } l \ k \ r) \ x &= \begin{cases} x < k : \text{lookup } l \ x \\ x > k : \text{lookup } r \ x \\ x = k : \mathbf{Just} \ k \end{cases}
 \end{aligned}$$

¹⁷For example the `Optional<T>` in Java and C++

¹⁸The complete example program can be found in the appendix of this chapter

For **Maybe** type data, we must handle two different possible values, for example:

$$\begin{aligned} \text{elem } \text{Nothing} &= \text{False} \\ \text{elem } (\text{Just } x) &= \text{True} \end{aligned}$$

Functor maps object, and also maps arrow. We see how **Maybe** functor maps object. How does it map arrow? On the left side in above diagram, there is an up-down arrow $A \xrightarrow{f} B$, and there is also an arrow $\text{Maybe } A \xrightarrow{\text{Maybe}(f)} \text{Maybe } B$ on the right side. Let's name the right side arrow f' . If we know the behavior of the left side arrow f , then what does the right side arrow behave? We mentioned for **Maybe** type data, we must handle two different possible values, hence f' should behave like this:

$$\begin{aligned} f' \text{Nothing} &= \text{Nothing} \\ f' (\text{Just } x) &= \text{Just } (f x) \end{aligned}$$

For given f , maps to f' , it is exactly what **Maybe** functor does for an arrow. In programming environment, we often use $fmap$ to define the map for arrow. We can define every functor **F** satisfies:

$$fmap : (A \rightarrow B) \rightarrow (\mathbf{F}A \rightarrow \mathbf{F}B)$$

For functor **F**, it maps the arrow from A to B to the arrow from $\mathbf{F}A$ to $\mathbf{F}B$. Hence for **Maybe** functor, the corresponding $fmap$ is defined as below:

$$\begin{aligned} fmap : (A \rightarrow B) &\rightarrow (\text{Maybe } A \rightarrow \text{Maybe } B) \\ fmap f \text{Nothing} &= \text{Nothing} \\ fmap f (\text{Just } x) &= \text{Just } (f x) \end{aligned}$$

Back to the binary search tree example, if the elements in the tree are integers, we want to search a value, and convert it to binary format if find; otherwise return *Nothing*. Suppose there has already existed a function that converts a decimal number to its binary format:

$$\text{binary}(n) = \begin{cases} n < 2 : & [n] \\ \text{Otherwise} : & \text{binary}(\lfloor \frac{n}{2} \rfloor) \uparrow [n \bmod 2] \end{cases}$$

Here is the corresponding example program. It uses tail recursion for performance purpose.

```
binary = bin []
  where
    bin xs 0 = 0 : xs
    bin xs 1 = 1 : xs
    bin xs n = bin ((n `mod` 2) : xs) (n `div` 2)
```

With functor, we can ‘lift’ this function arrow up as shown in below diagram:

$$\begin{array}{ccc} \text{Maybe } \text{Int} & \xrightarrow{\text{fmap binary}} & \text{Maybe } [\text{Int}] \\ \uparrow & & \uparrow \\ \text{Int} & \xrightarrow{\text{binary}} & [\text{Int}] \end{array}$$

Hence, we directly use maybe functor and *binary* arrow to manipulate the search result from the tree:

$$fmap \text{ binary } (\text{lookup } t \ x)$$

We can consider there are two worlds in above diagram. The bottom is the world on the earth, threatened by the null reference; the upper is the world in the sky, free from null and safe. With maybe functor, all the legacy programs on the earth, even they are not capable to handle null, can be lift to the sky, to the safe world dominated by the maybe functor.

Proof. Readers can skip the contents in this box To verify Maybe is a functor, we need check the two conditions about arrow mapping:

$$\begin{aligned} fmap \text{ id} &= id \\ fmap (f \circ g) &= fmap f \circ fmap g \end{aligned}$$

For the first condition, the definition of *id* is:

$$id \ x = x$$

Therefore:

$$\begin{aligned} fmap \text{ id } Nothing &= Nothing && \text{definition of } fmap \\ &= id \ Nothing && \text{reverse of } id \end{aligned}$$

And

$$\begin{aligned} fmap \text{ id } (Just \ x) &= Just (id \ x) && \text{definition of } fmap \\ &= Just \ x && \text{definition of } id \\ &= id (Just \ x) && \text{reverse of } id \end{aligned}$$

For the second condition:

$$\begin{aligned} fmap (f \circ g) \ Nothing &= Nothing && \text{definition of } fmap \\ &= fmap \ f \ Nothing && \text{reverse of } fmap \\ &= fmap \ f (fmap \ g \ Nothing) && \text{reverse of } fmap \\ &= (fmap \ f \circ fmap \ g) \ Nothing && \text{reverse of function composition} \end{aligned}$$

And

$$\begin{aligned} fmap (f \circ g) (Just \ x) &= Just ((f \circ g) \ x) && \text{definition of } fmap \\ &= Just (f (g \ x)) && \text{function composition} \\ &= fmap \ f (Just (g \ x)) && \text{reverse of } fmap \\ &= fmap \ f (fmap \ g (Just \ x)) && \text{reverse of } fmap \\ &= (fmap \ f \circ fmap \ g) (Just \ x) && \text{reverse of function composition} \end{aligned}$$

Therefore, Maybe is really a functor. □

Example 4.2.2. List functor. We introduced the definition of list in chapter 1 as:

data List A = Nil | Cons(A, List A)

From programming perspective, it defines the linked-list data structure. The elements stored are of type *A*. We call it link-list of type *A*. From category perspective, a list functor need define maps for both object and arrow. Here objects are types, arrows are total functions. Below diagram illustrates how list functor behaves.

$$\begin{array}{ccc} A & \xrightarrow{\quad} & \mathbf{List} \ A \\ f \downarrow & & \downarrow \mathbf{List}(f) \\ B & \xrightarrow{\quad} & \mathbf{List} \ B \end{array}$$

Objects A, B on the left side are data types, like integer, Boolean, or even complex types such as **Maybe Char**. Objects on the right side, are types mapped by list functor. If A is integer type Int , then the right side corresponds to **List** Int ; if B is character type $Char$, then the right side corresponds to **List** $Char$, which is $String$ essentially.

We highlight again that here the object is type, but not value. A can be Int , but can not be 5, a particular value. Hence **List** A corresponds to list of integers, but not a particular list, like $[1, 1, 2, 3, 5]$. How to generate list values? Function **Nil** and **Cons** are used to generate empty list or list like $List(1, List(1, List(2, Nil)))$.

Now we know how list functor maps objects. But how does it map arrows? Given a function $f : A \rightarrow B$, how to obtain another function $g : \mathbf{List} A \rightarrow \mathbf{List} B$ through list functor? Similar to maybe functor, we can realize **fmap** to map arrow f to arrow g . The type signature for list functor is:

$$\mathbf{fmap} : (A \rightarrow B) \rightarrow (\mathbf{List} A \rightarrow \mathbf{List} B)$$

Let's consider how g behaves. For the simplest case, no matter how f is defined, if the value of **List** A is an empty list **Nil**, then applying g on it gives empty list anyway. Therefore:

$$\mathbf{fmap} f \mathbf{Nil} = \mathbf{Nil}$$

For the recursive case **Cons**(x, xs), where x is some value of type A , and xs is a sub-list of type **List** A . If $f(x) = y$, which maps x of type A to y of type B , then we first apply f to x , next recursively apply it to the sub-list xs to obtain a new sub-list ys with the element type of B . Finally, we concatenate y and ys as the result:

$$\mathbf{fmap} f \mathbf{Cons}(x, xs) = \mathbf{Cons}(f x, \mathbf{fmap} f xs)$$

Summarize them together, we have the complete definition of **fmap** for list:

$$\begin{aligned} \mathbf{fmap} &: (A \rightarrow B) \rightarrow (\mathbf{List} A \rightarrow \mathbf{List} B) \\ \mathbf{fmap} f \mathbf{Nil} &= \mathbf{Nil} \\ \mathbf{fmap} f \mathbf{Cons}(x, xs) &= \mathbf{Cons}(f x, \mathbf{fmap} f xs) \end{aligned}$$

In chapter 1, we introduced the simplified notation, using the infix ‘ $:$ ’ for **Cons**, and ‘ $[]$ ’ for **Nil**, then the definition of list functor that maps arrows can be simplified as:

$$\begin{aligned} \mathbf{fmap} &: (A \rightarrow B) \rightarrow (\mathbf{List} A \rightarrow \mathbf{List} B) \\ \mathbf{fmap} f [] &= [] \\ \mathbf{fmap} f (x : xs) &= (f x) : (\mathbf{fmap} f xs) \end{aligned}$$

Compare this definition with the list mapping definition in chapter 1, we find they are exactly same except for the name. It means we can re-use the list mapping to define list functor. This is the case in some programming environment, which re-uses **map** for list functor.

```
instance Functor [] where
  fmap = map
```

At the end of this example, let's verify the arrow mapping for list functor preserve identity and composition. **Readers can skip the proof in this box.**

$$\begin{aligned} \mathbf{fmap} id &= id \\ \mathbf{fmap} (f \circ g) &= \mathbf{fmap} f \circ \mathbf{fmap} g \end{aligned}$$

Proof. We use mathematical induction to verify the identity condition. First for empty list:

$$\begin{aligned} fmap id Nil &= Nil && \text{definition of } fmap \\ &= id Nil && \text{reverse of } id \end{aligned}$$

For the recursive case of $(x : xs)$, assume $fmap id xs = id xs$ holds, we have:

$$\begin{aligned} fmap id (x : xs) &= (id x) : (fmap id xs) && \text{definition of } fmap \\ &= (id x) : (id xs) && \text{induction assumption} \\ &= x : xs && \text{definition of } id \\ &= id (x : xs) && \text{reverse of } id \end{aligned}$$

Again, we use mathematical induction to verify composition condition. For empty list, we have:

$$\begin{aligned} fmap (f \circ g) Nil &= Nil && \text{definition of } fmap \\ &= fmap f Nil && \text{reverse of } fmap \\ &= fmap f (fmap g Nil) && \text{reverse of } fmap \\ &= (fmap f \circ fmap g) Nil && \text{reverse of function composition} \end{aligned}$$

For the recursive case of $(x : xs)$, assume $fmap (f \circ g) xs = (fmap f \circ fmap g) xs$ holds, we have:

$$\begin{aligned} fmap (f \circ g) (x : xs) &= ((f \circ g) x) : (fmap (f \circ g) xs) && \text{definition of } fmap \\ &= ((f \circ g) x) : ((fmap f \circ fmap g) xs) && \text{induction assumption} \\ &= (f(g x)) : (fmap f (fmap g xs)) && \text{function composition} \\ &= fmap f ((g x) : (fmap g xs)) && \text{reverse of } fmap \\ &= fmap f (fmap g (x : xs)) && \text{reverse of } fmap \text{ again} \\ &= (fmap f \circ fmap g) (x : xs) && \text{reverse of function composition} \end{aligned}$$

Thus we verified **List** is really a functor. □

Exercise 4.2

1. For the list functor, define the arrow map with *foldr*.
2. Verify that the composition of maybe functor and list functor **Maybe** \circ **List** and **List** \circ **Maybe** are all functors.
3. Proof that the composition for any functors **G** \circ **F** is still a functor.
4. Give an example functor for preset.
5. For the binary tree defined in chapter 2, define the functor for it.

4.3 Products and coproducts

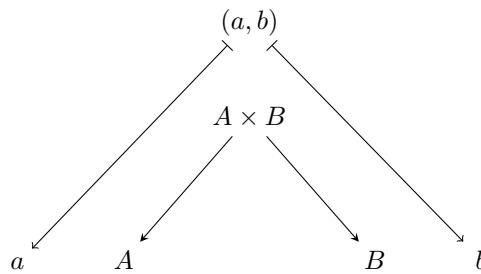
Before we introduce more complex categories and functors, let us see what product and coproduct are. We start from the product of sets. Given two set A and B , their Cartesian product $A \times B$ is the set of all ordered pairs (a, b) , where $a \in A$ and $b \in B$:

$$\{(a, b) | a \in A, b \in B\}$$

For example, the product of finite sets $\{1, 2, 3\}$ and $\{a, b\}$ is:

$$\{(1, a), (2, a), (3, a), (1, b), (2, b), (3, b)\}$$

If set A and B are the same algebraic structures, like group, ring etc., then we can define objects and arrows as shown in below diagram:



René Descartes (1596 - 1650). Portrait after Frans Hals, oil on canvas, Louvre Museum

Cartesian product, also known as direct product, is named after René Descartes, the great French philosopher, mathematician, and scientist. In the days of Descartes, Latin was a widely used academy language. He also had a Latinized name: Renatus Cartesius. Its adjective form is Cartesian. This is the reason why we say Cartesian product or Cartesian coordinate system today.

Descartes was born in La Haye, near Tours, France in 1596. His father was a member of the Parlement of Brittany at Rennes. His mother died a year after giving birth to him, and so he was not expected to survive. His father then remarried in 1600. Descartes lived with his grandmother at La Haye. His health was poor when he was a child. Throughout his childhood, up to his twenties, he was pale and had a persistent cough which was probably due to tuberculosis. It seems likely that he inherited these health problems from his mother.

In 1607, he entered the Jesuit college of La Flèche, where he was introduced to mathematics and physics, including Galileo's work. While in the school his health was poor and, instead of rising at 5 AM. like the other boys, he was granted permission to remain in bed until 11 o'clock in the morning, a custom he maintained until the year of his death.

After graduation in 1614, he studied for two years at the University of Poitiers. He received a law degree in 1616 to comply with his father's wishes but he quickly decided that this was not the path he wanted to follow. He returned to Paris, then became a volunteer in the army of Maurice of Nassau. In the army, Descartes started studying mathematics and mechanics under the Dutch scientist Isaac Beeckman, and began to seek a unified science of nature.

After this time in Holland he left the service of Maurice of Nassau and travelled through Europe. In 1619 he joined the Bavarian army and was stationed in Ulm. An important event in his life was three dreams he had in November 1619. These he believed were sent by a divine spirit with the intention of revealing to him a new approach to philosophy. The ideas from these dreams would dominate much of his work from that time on.

From 1620 to 1628 Descartes travelled through Europe, spending time in Bohemia, Hungary, Germany, Holland, through Switzerland to Italy, then Venice and Rome. He returned Paris in 1625. His Paris home became a meeting place for philosophers and mathematicians and steadily became more and more busy. By 1628 Descartes, tired of the bustle of Paris, the house full of people, and of the life of travelling he had before, decided to settle down where he could work in solitude. He gave much thought to choosing a country suited to his nature and he chose Holland. What he longed for was somewhere peaceful where he could work away from the distractions of a city such as Paris yet still have access to the facilities of a city. It was a good decision which he did not seem to regret over the next twenty years. He told his friend Marin Mersenne¹⁹ where he was living so that he might keep in touch with the mathematical world, but otherwise he kept his place of residence a secret. Descartes wrote all his major work during his 20-plus years in the Netherlands, initiating a revolution in mathematics and philosophy. In 1633, Galileo was condemned by the Italian Inquisition, and Descartes abandoned plans to publish *Treatise on the World*, his work of the previous four years. Nevertheless, in 1637 he published parts of this work in three essays: *The Meteors*, *Dioptrics* and *Geometry*, preceded by an introduction, his famous *Discourse on the Method*.

In *Geometry*, Descartes exploited the discoveries he made with Pierre de Fermat. This later became known as Cartesian Geometry. Descartes continued to publish works concerning both mathematics and philosophy for the rest of his life. In 1641 he published a metaphysics treatise, *Meditations on First Philosophy*. It was followed in 1644 by *Principles of Philosophy*. He became the most influential philosophers in Europe.

In 1649 Queen Christina of Sweden persuaded Descartes to go to Stockholm. However the Queen wanted to draw tangents at 5 AM. and Descartes broke the habit of his lifetime of getting up at 11 o'clock. After only a few months in the cold northern climate, walking to the palace for 5 o'clock every morning, he died of pneumonia February 1650 at the age of 54.

Descartes left the best known philosophical statement "I think, therefore I am" (French: Je pense, donc je suis; Latin: cogito, ergo sum). Descartes laid the foundation for 17th-century continental rationalism. He was well-versed in mathematics as well as philosophy, and contributed greatly to science as well. He is credited as the father of analytical geometry, the bridge between algebra and geometry—used in the discovery of infinitesimal calculus and analysis. Descartes was also one of the key figures in the Scientific Revolution.

Symmetric to the Cartesian product of sets, there is a dual construct. From two sets A and B , we can generate a disjoint union set (sum) $A + B$. For the element in the sum, in order to know which set A or B it comes from, we can add a tag:

$$A + B = (A \times \{0\}) \cup (B \times \{1\})$$

For every element (x, tag) in $A + B$, if the tag is 0, we know that x comes from A , else if the tag is 1, then x comes from B . Hence the sum of finite set $\{1, 2, 3\}$ and $\{a, b\}$ is

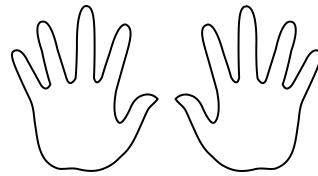
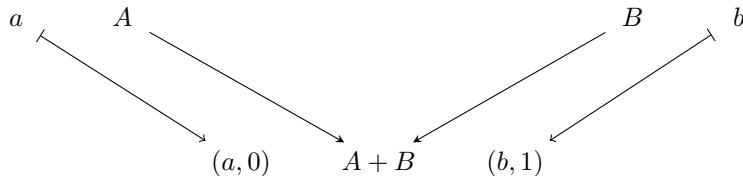
¹⁹Mersenne was a French polymath, whose works touched a wide variety of fields. He is perhaps best known today among mathematicians for Mersenne prime numbers, in the form $M_n = 2^n - 1$ for some integer n . He had many contacts in the scientific world and has been called "the center of the world of science and mathematics during the first half of the 1600s".

$$\{(1, 0), (2, 0), (3, 0), (a, 1), (b, 1)\}$$

When programming, $A + B$ can be defined as:

$$A + B = \text{zip } A \{0, \dots\} \uplus \text{zip } B \{1, \dots\}$$

If set A and B are the same algebraic structures, we can define the below objects and arrows:



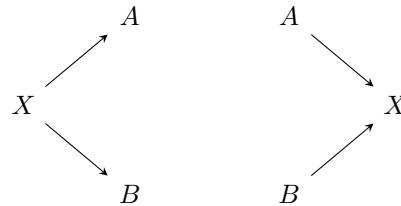
The two constructions are symmetric. If rotate the two diagrams by 90 degree, laying A on top of B , then the two problems appear left and right. They are symmetric like our two hands. We can find a lot of such symmetric concepts in category theory. Our world is full of symmetric things. When we understand one, we understand the dual one at the same time.

4.3.1 Definition of product and coproduct

Definition 4.3.1. For a pair of objects A and B in category \mathcal{C} , a *wedge*

to from

the pair A, B is an object X together with a pair of arrows



in the parent category \mathcal{C} .

For a given pair of objects A, B in the category, there may be many wedges on one side or the other. We look for the ‘best possible’ wedge, the one that is as ‘near’ the pair as possible. Technically, we look for a universal wedge. It leads to the below (pair of) definitions:

Definition 4.3.2. Given a pair object A, B of a category \mathcal{C} , a

product coproduct

of the pair is a wedge:

$$\begin{array}{ccc} & A & \\ p_A \nearrow & & \searrow i_A \\ S & & S \\ \searrow & & \nearrow i_B \\ & B & \end{array} \quad \begin{array}{ccc} & A & \\ & \searrow i_A & \\ & S & \\ \nearrow & & \nearrow i_B \\ B & & B \end{array}$$

with the following universal property. For each wedge

$$\begin{array}{ccc} & A & \\ f_A \nearrow & & \searrow f_A \\ X & & X \\ \searrow & & \nearrow f_B \\ & B & \end{array} \quad \begin{array}{ccc} & A & \\ & \searrow f_A & \\ & X & \\ \nearrow & & \nearrow f_B \\ B & & B \end{array}$$

there is a unique arrow:

$$X \xrightarrow{m} S \quad S \xrightarrow{m} X$$

such that,

$$\begin{array}{ccc} & A & \\ f_A \nearrow & & \uparrow p_A \\ X \xrightarrow{m} S & & S \\ \searrow & & \downarrow p_B \\ & B & \end{array} \quad \begin{array}{ccc} & A & \\ & \searrow f_A & \\ & S & \\ \downarrow i_A & \nearrow m & \nearrow f_B \\ S \xrightarrow{m} X & & B \end{array}$$

commutes. This arrow m is the *mediating arrow* (or mediator) for the wedge on X .

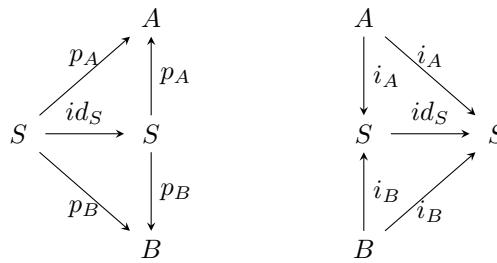
In this definition, the product or coproduct is not just an object S , but an object together with a pair of arrows. For any given X , the mediating arrow m is unique. We say the diagram commutes, it means the arrows satisfy the following equations:

$$\begin{aligned} f_A &= p_A \circ m \\ f_B &= p_B \circ m \end{aligned}$$

It immediately leads to this special case: If X equals to S , then m is an endo-arrow (points to itself).

$$S \xrightarrow{m} S$$

Hence m must be identity arrow. The diagram then simplifies to:



Among the many wedges, the product and coproduct are special, they are universal solutions. In other words, they are the ‘closest’ or the ‘best’ wedge. We can prove that product and coproduct are unique (see the appendix of this book). However, product and coproduct may not exist at the same time, it’s also possible that neither does exist. Let’s see what product and coproduct mean to set.

Lemma 4.3.1. Let A, B be a pair of sets. Then the

Cartesian product $A \times B$	disjoint union $A + B$
-----------------------------------	---------------------------

furnished with canonical functions form the

product	coproduct
---------	-----------

of the pair in the category **Set**.

The detailed proof can be found in the appendix of this book. In program environment, product is often realized with paired tuple (a, b) and functions fst, snd . However, coproduct is sometimes realized with a dedicated data type.

data Either a b = **Left** a | **Right** b

The advantage is that we needn’t the 0, 1 tags to mark if an element x of type **Either** a b comes from **a** or **b**. The below example program handles the coproduct value through pattern matching:

either :: (a → c) → (b → c) → **Either** a b → c
either f _ (**Left** x) = f x
either _ g (**Right** y) = g y

Let’s see an example, consider the coproduct **Either** String Int. It is either a string, like **s** = **Left** "hello"; or an integer, like **n** = **Right** 8. If it is a string, we want to count its length; if it is a number, we want to double it. To do this, we can utilize the **either** function: **either** length (*2) x.

Thus **either** length (*2) **s** will count the length of string "hello" which gives 5; while **either** length (*2) **n** will double 8 to give 16. Some programming environments have the concept of **union** or **enum** data type to realize coproduct partially. In the future, we sometimes call the two arrows of the coproduct *left* and *right* respectively.

4.3.2 The properties of product and coproduct

According to the definition of product and coproduct, for any wedge consist of X and arrows, the mediator arrow m is uniquely determined. In category **Set**, the mediator arrow can be defined as the following:

product $m(x) = (a, b)$	coproduct $\begin{cases} m(a, 0) = p(a) \\ m(b, 1) = q(b) \end{cases}$
----------------------------	---

For a generic category, how to define the mediator arrow? To do that, we introduce two dedicated symbols for arrow operation. Given any wedge:

$$\begin{array}{ccc}
 & \begin{array}{c} A \\ f \nearrow \\ X \\ \searrow \\ B \end{array} & \begin{array}{c} A \\ f \searrow \\ X \\ \nearrow \\ B \end{array} \\
 \end{array}$$

Define

$$m = \langle f, g \rangle \quad m = [f, g]$$

which satisfy:

$$\begin{cases} fst \circ m = f \\ snd \circ m = g \end{cases} \quad \begin{cases} m \circ left = f \\ m \circ right = g \end{cases}$$

Hence the following diagram commutes:

$$\begin{array}{ccc}
 & \begin{array}{c} A \\ f \nearrow \\ X \xrightarrow{\langle f, g \rangle} A \times B \\ \searrow \\ B \end{array} & \begin{array}{c} A \\ f \searrow \\ left \downarrow \\ A + B \xrightarrow{[f, g]} X \\ \nearrow \\ right \uparrow \\ B \end{array} \\
 \end{array}$$

From this pair of diagrams, we can obtain some important properties for product and coproduct directly. First is the *cancellation law*:

$$\begin{cases} fst \circ \langle f, g \rangle = f \\ snd \circ \langle f, g \rangle = g \end{cases} \quad \begin{cases} [f, g] \circ left = f \\ [f, g] \circ right = g \end{cases}$$

For product, if arrow f equals to fst , and arrow g equals to snd , then we obtain the special case of identity arrow mentioned in previous section. Similarly for the coproduct, if arrow f equals to $left$, and arrow g equals to $right$, then the mediator arrow is also the identity arrow. We call this property the *reflection law*:

$$id = \langle fst, snd \rangle \quad id = [left, right]$$

If there exists another wedge Y and arrows h and k , together with f, g they satisfy the following conditions:

$$\begin{array}{ll} \text{product} & \text{coproduct} \\ \begin{cases} h \circ \phi = f \\ k \circ \phi = g \end{cases} & \begin{cases} \phi \circ h = f \\ \phi \circ k = g \end{cases} \end{array}$$

For

$$\langle h, k \rangle \circ \phi \quad \phi \circ [h, k]$$

Substitute m , and apply cancellation law, then we obtain the *fusion law*:

$$\begin{array}{ll} \text{product} & \text{coproduct} \\ \begin{cases} h \circ \phi = f \\ k \circ \phi = g \end{cases} \Rightarrow \langle h, k \rangle \circ \phi = \langle f, g \rangle & \begin{cases} \phi \circ h = f \\ \phi \circ k = g \end{cases} \Rightarrow \phi \circ [h, k] = [f, g] \end{array}$$

It means:

$$\langle h, k \rangle \circ \phi = \langle h \circ \phi, k \circ \phi \rangle \quad \phi \circ [h, k] = [\phi \circ h, \phi \circ k]$$

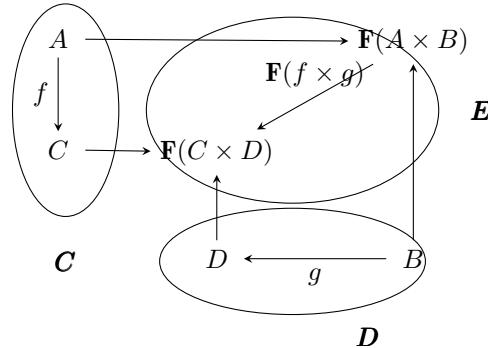
We'll see later, these laws play important roles during algorithm reasoning, simplification, and optimization.

4.3.3 Functors from products and coproducts

With product in category theory, we can introduce the concept of *bifunctor* (also known as binary functor). The functors we've seen so far transform the objects from one category to the objects in another category; and also transforms arrows from one category to the other. The bifunctor applies to the product of two categories \mathbf{C} and \mathbf{D} . In other words, the source is $\mathbf{C} \times \mathbf{D}$. Bifunctor transforms objects as the following:

$$\begin{array}{ccc} \mathbf{C} \times \mathbf{D} & \longrightarrow & \mathbf{E} \\ A \times B & \longmapsto & \mathbf{F}(A \times B) \end{array}$$

Besides objects, functor must transform arrows as well. For the arrow f in category \mathbf{C} , and the arrow g in category \mathbf{D} , what the bifunctor behaves? Observe the below diagram:



From this diagram, the arrow $A \xrightarrow{f} C$ and $B \xrightarrow{g} D$ are sent to the new arrows in category \mathbf{E} through the functor \mathbf{F} . The source object is $\mathbf{F}(A \times B)$, and the target object is $\mathbf{F}(C \times D)$. To do that, we first define the product of two arrows f and g , it applies to the product of A and B . For every $(a, b) \in A \times B$, it behaves as below:

$$(f \times g)(a, b) = (f a, g b)$$

The bifunctor F applies to this product of arrows $f \times g$, sends it to the new arrow $\mathbf{F}(f \times g)$. Hence the final definition for bifunctor is as below:

$$\begin{array}{ccc} \mathbf{C} \times \mathbf{D} & \longrightarrow & \mathbf{E} \\ A \times B & \longmapsto & \mathbf{F}(A \times B) \\ f \times g & \longmapsto & \mathbf{F}(f \times g) \end{array}$$

We need verify the bifunctor satisfies the two conditions as a functor: the identity condition and the composition condition. **Readers can skip the proof in this box.**

$$\begin{aligned} \mathbf{F}(id \times id) &= id \\ \mathbf{F}((f \circ f') \times (g \circ g')) &= \mathbf{F}(f \times g) \circ \mathbf{F}(f' \times g') \end{aligned}$$

Proof. We treat the product as an object. If we prove the following two conditions essentially, then the proof for the bifunctor can be deduced from it through the known result of normal functors.

$$\begin{aligned} id \times id &= id \\ (f \circ f') \times (g \circ g') &= (f \times g) \circ (f' \times g') \end{aligned}$$

We start from proving the identity condition. For all $(a, b) \in A \times B$, we have:

$$\begin{aligned} (id \times id)(a, b) &= (id(a), id(b)) && \text{product of arrows} \\ &= (a, b) && \text{definition of } id \\ &= id(a, b) && \text{reverse of } id \end{aligned}$$

Next we prove the composition condition.

$$\begin{aligned} ((f \circ f') \times (g \circ g'))(a, b) &= ((f \circ f') a, (g \circ g') b) && \text{product of arrows} \\ &= (f(f'(a)), g(g'(b))) && \text{arrow composition} \\ &= (f \times g)(f'(a), g'(b)) && \text{reverse of arrows product} \\ &= (f \times g)((f' \times g')(a, b)) && \text{reverse of arrows product} \\ &= ((f \times g) \circ (f' \times g'))(a, b) \end{aligned}$$

Hence proved the bifunctor satisfies both functor conditions. □

Similar to *fmap*, some programming environments have limitation to use the same symbol for both object and arrow mapping. To solve that, we can define a dedicated *bimap* for bifunctor arrow mapping. We need every bifunctor \mathbf{F} satisfy:

$$bimap : (A \rightarrow C) \rightarrow (B \rightarrow D) \rightarrow (\mathbf{F} A \times B \rightarrow \mathbf{F} C \times D)$$

It means, if \mathbf{F} is a bifunctor, then it send the two arrows $A \xrightarrow{g} C$ and $B \xrightarrow{h} D$ to an arrow that maps from $\mathbf{F} A \times B$ to $\mathbf{F} C \times D$.

With the concept of bifunctor, we can define product and coproduct functors. We'll use the infix notation. Denote:

$$\begin{array}{ccc} \text{product functor as} & & \text{coproduct functor as} \\ \times & & + \end{array}$$

Given two objects, the product functor maps them to their product; while the coproduct functor maps them to their coproduct. For arrows, define:

$$f \times g = \langle f \circ fst, g \circ snd \rangle \quad f + g = [left \circ f, right \circ g]$$

We need verify the two functor conditions, **Readers can skip the proof in this box**. For the identity condition. Substitute f, g with id :

$$\begin{array}{llll}
 & id \times id & id + id & \\
 = & \langle id \circ fst, id \circ snd \rangle & \text{Definition of } \times & = [left \circ id, right \circ id] \text{ Definition of } + \\
 = & \langle fst, snd \rangle & \text{Definition of } id & = [left, right] \text{ Definition of } id \\
 = & id & \text{Reflection law} & = id \text{ Reflection law}
 \end{array}$$

Next we need verify the composition condition:

$$\begin{array}{ll}
 \text{product} & \text{coproduct} \\
 (f \times g) \circ (f' \times g') = f \circ f' \times g \circ g' & (f + g) \circ (f' + g') = f \circ f' + g \circ g'
 \end{array}$$

In order to prove it, we will first prove the *absorption law* for

$$\begin{array}{ll}
 \text{product} & \text{coproduct} \\
 (f \times g) \circ \langle p, q \rangle = \langle f \circ p, g \circ q \rangle & [p, q] \circ (f + g) = [p \circ f, q \circ g]
 \end{array}$$

We only give the proof for the product on the left. the coproduct side can be proved similarly. We leave it as exercise.

$$\begin{array}{ll}
 (f \times g) \circ \langle p, q \rangle & \\
 = \langle f \circ fst, g \circ snd \rangle \circ \langle p, q \rangle & \text{Definition of } \times \\
 = \langle f \circ fst \circ \langle p, q \rangle, g \circ snd \circ \langle p, q \rangle \rangle & \text{Fusion law} \\
 = \langle f \circ p, g \circ q \rangle & \text{Cancellation law}
 \end{array}$$

Using the absorption law, let $p = f' \circ fst, q = g' \circ snd$, we can verify the composition condition:

$$\begin{array}{ll}
 (f \times g) \circ (f' \times g') & \\
 = (f \times g) \circ \langle f' \circ fst, g' \circ snd \rangle & \text{Definition of } \times \text{ for the 2nd term} \\
 = (f \times g) \circ \langle p, q \rangle & \text{Substitute with } p, q \\
 = \langle f \circ p, g \circ q \rangle & \text{absorption law} \\
 = \langle f \circ f' \circ fst, g \circ g' \circ snd \rangle & \text{Substitute back } p, q \\
 = \langle (f \circ f') \circ fst, (g \circ g') \circ snd \rangle & \text{association law} \\
 = (f \circ f') \times (g \circ g') & \text{Reverse of } \times
 \end{array}$$

Product functor is an instance of bifunctor. We can define it as *bimap* like below:

$$\begin{aligned}
 \text{bimap} : (A \rightarrow C) \rightarrow (B \rightarrow D) \rightarrow (A \times B \rightarrow C \times D) \\
 \text{bimap } f \ g (x, y) = (f x, g y)
 \end{aligned}$$

With the `Either` type to realize the coproduct functor, the corresponding *bimap* can be defined like this:

$$\begin{aligned}
 \text{bimap} : (A \rightarrow C) \rightarrow (B \rightarrow D) \rightarrow (\text{Either } A \ B \rightarrow \text{Either } C \ D) \\
 \text{bimap } f \ g (\text{left } x) = \text{left } (f x) \\
 \text{bimap } f \ g (\text{right } y) = \text{right } (g y)
 \end{aligned}$$

Exercise 4.3

1. For any two objects in a poset, what is their product? what is their coproduct?
2. Prove the absorption law for coproduct, and verify the coproduct functor satisfies composition condition.

4.4 Natural transformation

When Eilenberg and Mac Lane developed category theory in the early 1940s, they wanted to explain why certain ‘natural’ construction are natural, and other constructions are not. As the result, categories were invented to support functors, and these were invented to support natural transformations. Mac Lane is said to have remarked, “I didn’t invent categories to study functors; I invented them to study natural transformations.” We’ve introduced categories and functors. In category theory, arrows are used to compare objects, functors are used to compare categories. What will be used to compare functors? It is natural transformation that serves this purpose.

Consider the following two functors:

$$\begin{array}{ccc} & \mathbf{F} & \\ \xrightarrow{\hspace{2cm}} & & \\ Src & & Trg \\ \xrightarrow{\hspace{2cm}} & & \\ & \mathbf{G} & \end{array}$$

They connect two categories. Both are covariant or contravariant. How to compare them? Since functors map both objects and arrows, we need compare the mapped objects and mapped arrows. Consider an object A in category \mathbf{Src} , it is mapped to two objects $\mathbf{F}A$ and $\mathbf{G}A$ in category \mathbf{Trg} . We care about the arrow from $\mathbf{F}A$ to $\mathbf{G}A$.

$$\mathbf{F}A \xrightarrow{\phi_A} \mathbf{G}A$$

Besides A , we do the similar study to all the objects in category Src .

Definition 4.4.1. Given a parallel pair of functors \mathbf{F}, \mathbf{G} of the same variance as shown in above figure, a natural transformation

$$F \xrightarrow{\phi} G$$

is a family of arrows of \mathbf{Trg} indexed by the object A of \mathbf{Src} .

$$\mathbf{F} A \xrightarrow{\phi_A} \mathbf{G} A$$

and such that for each arrow $A \xrightarrow{f} B$ of \mathbf{Src} , the appropriate square in \mathbf{Trq} commutes

From the diagram of natural transformation, we see for every arrow f , there is a corresponding square commutes. For covariant case, commutativity means for all arrow f , the following equation holds:

$$\mathbf{G}(f) \circ \phi_A = \phi_B \circ \mathbf{F}(f)$$

4.4.1 Examples of natural transformation

Natural transformation is a higher level of abstraction on top of categories, arrows, and functors. Let's see some examples to help understand this concept.

Example 4.4.1. The first example is the *inits* function. It enumerates all the prefixes of a given string or list. For instances:

```
inits "Mississippi" = ["", "M", "Mi", "Mis", "Miss", "Missi", "Missis",  
"Mississ", "Mississi", "Mississip", "Mississipp", "Mississippi"]  
  
inits [1, 2, 3, 4] = [[], [1], [1, 2], [1, 2, 3], [1, 2, 3, 4]]
```

The behavior of *inits* function can be summarized as this:

$$\text{inits}[a_1, a_2, \dots, a_n] = [[], [a_1], [a_1, a_2], \dots, [a_1, a_2, \dots, a_n]]$$

Consider the category **Set**, for every object, a set A (or type A), there exists *inits* arrow indexed by A :

$$\text{inits}_A : \text{List}A \rightarrow \text{List}(\text{List}A)$$

There is a list functor **List**, and another embedded list functor **List List**.

With the simplified '[]' notation, this arrow can also be written as:

$$[A] \xrightarrow{\text{inits}_A} [[A]]$$

Next, we need verify that for any function $A \xrightarrow{f} B$, the equation

$$\text{List}(\text{List}(f)) \circ \text{inits}_A = \text{inits}_B \circ \text{List}(f)$$

holds. It means the below square diagram commutes:

$$\begin{array}{ccc} A & [A] & [[A]] \\ \downarrow f & \downarrow \text{List}(f) & \downarrow \text{List}(\text{List}(f)) \\ B & [B] & [[B]] \\ & \xrightarrow{\text{init}_B} & \end{array}$$

Proof. We'll prove the commutivity with the *fmap* defined in previous section. For n elements of any type A : a_1, a_2, \dots, a_n , the n elements b_1, b_2, \dots, b_n of type B are their corresponding mapped value: $f(a_1) = b_1, f(a_2) = b_2, \dots, f(a_n) = b_n$. We have:

$$\begin{aligned} & \text{List}(\text{List}(f)) \circ \text{init}_A[a_1, \dots, a_n] \\ = & \text{fmap}_{[]} (f) \circ \text{init}_A[a_1, \dots, a_n] & \text{definition of } \text{fmap} \\ = & \text{fmap}_{[]} (f)[[], [a_1], \dots, [a_1, a_2, \dots, a_n]] & \text{definition of } \text{inits} \\ = & \text{map}(\text{map } f)[[], [a_1], \dots, [a_1, a_2, \dots, a_n]] & \text{fmap is defined as map for list functor} \\ = & [\text{map } f \ [], \text{map } f [a_1], \dots, \text{map } f [a_1, a_2, \dots, a_n]] & \text{definition of map} \\ = & [[], [f(a_1)], \dots, [f(a_1), f(a_2), \dots, f(a_n)]] & \text{apply map } f \text{ to every sub-list} \\ = & [[], [b_1], \dots, [b_1, b_2, \dots, b_n]] & \text{definition of } f \\ = & \text{init}_B [b_1, b_2, \dots, b_n] & \text{reverse of } \text{init} \\ = & \text{init}_B [f(a_1), f(a_2), \dots, f(a_n)] & \text{reverse of } f \\ = & \text{init}_B \circ \text{map}(f) [a_1, a_2, \dots, a_n] & \text{reverse of map } f \\ = & \text{init}_B \circ \text{fmap}_{[]} (f)[a_1, \dots, a_n] & \text{fmap is defined as map for list functor} \\ = & \text{init}_B \circ \text{List}(f)[a_1, \dots, a_n] \end{aligned}$$

□

Therefor, $inits : \mathbf{List} \rightarrow \mathbf{List} \circ \mathbf{List}$ is a natural transformation.

Example 4.4.2. The next example is called $safeHead$, it can safely access the first element of the list. Here ‘safe’ means it can handle the empty list (Nil) case without exception. To do that, we will utilize the maybe functor defined previously. $safeHead$ is defined as the following:

$$\begin{aligned} safeHead : [A] &\rightarrow \mathbf{Maybe} A \\ safeHead [] &= \mathbf{Nothing} \\ safeHead (x : xs) &= \mathbf{Just} x \end{aligned}$$

In the category of set, every type A , which is an object, indexes the corresponding arrow $safeHead$ as:

$$[A] \xrightarrow{safeHead_A} \mathbf{Maybe} A$$

The two functors involved here are the list functor and maybe functor. We need next verify that, for every arrow (function) $A \xrightarrow{f} B$, the below square diagram commutes:

$$\begin{array}{ccccc} & & [A] & \xrightarrow{safeHead_A} & \mathbf{Maybe} A \\ A & \downarrow f & \downarrow \mathbf{List}(f) & & \downarrow \mathbf{Maybe}(f) \\ B & & [B] & \xrightarrow{safeHead_B} & \mathbf{Maybe} B \end{array}$$

That is to prove:

$$\mathbf{Maybe}(f) \circ safeHead_A = safeHead_B \circ \mathbf{List}(f)$$

Proof. We prove it for two cases. The first case is the empty list:

$$\begin{aligned} & \mathbf{Maybe}(f) \circ safeHead_A [] \\ = & \mathbf{Maybe}(f) \mathbf{Nothing} && \text{definition of } safeHead \\ = & \mathbf{Nothing} && \text{definition of } fmap f \mathbf{Nothing} \\ = & safeHead_B [] && \text{reverse of } safeHead \\ = & safeHead_B \circ \mathbf{List}(f) [] && \text{reverse of } fmap f [] \end{aligned}$$

The second case is the non-empty list $(x : xs)$:

$$\begin{aligned} & \mathbf{Maybe}(f) \circ safeHead_A (x : xs) \\ = & \mathbf{Maybe}(f) (\mathbf{Just} x) && \text{definition of } safeHead \\ = & \mathbf{Just} f(x) && \text{definition of } fmap f \mathbf{Just} x \\ = & safeHead_B (f(x) : fmap f xs) && \text{reverse of } safeHead \\ = & safeHead_B \circ \mathbf{List}(f) (x : xs) && \text{reverse of } fmap f (x : xs) \end{aligned}$$

Summarize both cases, hence proved that $safeHead : \mathbf{List} \rightarrow \mathbf{Maybe}$ is a natural transformation. \square

From the two examples, we summarize that, for any object A of the category (For set category, it is a set A ; in programming, it is a type A), functor \mathbf{F} sends it to another object $\mathbf{F}A$ (For set category, $\mathbf{F}A$ is another set; in programming, $\mathbf{F}A$ is another type),

while the other functor **G** sends the object to **GA**. Natural transformation ϕ indexed by A (For set category, it is a map; in programming, it's a function)²⁰ is in the form:

$$\phi_A : \mathbf{F}A \rightarrow \mathbf{G}A$$

Not only for A , when abstract *all* objects, we obtain a family of arrows (in programming, it is a *polymorphic function*²¹):

$$\phi : \forall A \cdot \mathbf{F}A \rightarrow \mathbf{G}A$$

In some programming environments, natural transformation can be written as²²:

```
phi :: forall a o F a -> G a
```

When we needn't explicitly call out `forall a`, natural transformation can be simplified as:

```
phi : F a -> G a
```

For the two examples above, the type of *inits* and *safeHead* are:

```
inits :: [a] -> [[a]]
```

```
safeHead :: [a] -> Maybe a
```

It only substitutes the name `phi` to their names respectively, and replace F and G to their own functors.

4.4.2 Natural isomorphism

Natural transformation was developed to compare functors. Similar to the isomorphism concept in abstract algebra, we need define when two functors are considered ‘equal’.

Definition 4.4.2. A *natural isomorphism* between two functors **F** and **G** is a natural transformation:

$$\mathbf{F} \xrightarrow{\phi} \mathbf{G}$$

such that for each source arrow A , the selected arrow

$$\mathbf{F}A \xrightarrow{\phi_A} \mathbf{G}A$$

is an isomorphism in the target category.

Two functors that are naturally isomorphic are also said to be *naturally equivalent*.

The simplest natural isomorphism example is *swap*. For the product of any two objects $A \times B$, *swap* turns it into $B \times A$:

$$\begin{aligned} \text{swap} : A \times B &\rightarrow B \times A \\ \text{swap} (a, b) &= (b, a) \end{aligned}$$

swap is a natural transformation. It transforms one bifunctor to another. Both bifunctors happen to be product functors: $\mathbf{F} = \mathbf{G} = \times$.

Since they are bifunctors, for every two arrows $A \xrightarrow{f} C$ and $B \xrightarrow{g} D$, we need the following natural condition so that the diagram commutes:

$$(g \times f) \circ \text{swap}_{A \times B} = \text{swap}_{C \times D} \circ (f \times g)$$

²⁰also called as the component at A

²¹Think about the polymorphic function in object oriented programming, and the template function in generic programming.

²²There is an `ExplicitForAll` option in Haskell, we'll see it again in the next chapter about build/foldr fusion law.

$$\begin{array}{ccc}
 & & g \\
 B & \xrightarrow{\hspace{2cm}} & D \\
 & \uparrow & \\
 A & & A \times B \xrightarrow{\text{swap}_{A \times B}} B \times A \\
 f \downarrow & f \times g \downarrow & \downarrow g \times f \\
 C & & C \times D \xrightarrow{\text{swap}_{C \times D}} D \times C
 \end{array}$$

It's easy to prove this. We can chose any two products (a, b) and (c, d) , substitute them into both sides of the natural condition. We leave the proof as an exercise of this section.

Although both are product functors, let's take some time to prove they are natural isomorphic. For the product of any two objects $A \times B$, as:

$$\text{swap}_{A \times B} \circ \text{swap}_{B \times A} = \text{id}$$

It tells us that *swap* is an one to one mapping. It is isomorphic in the target category, hence proved the natural isomorphism.

All the above three examples *inits*, *safeHead*, and *swap* are both polymorphic functions and natural transformations. This is not a coincidence. In fact, all polymorphic functions are natural transformations in functional programming[44].

Exercise 4.4

1. Prove that *swap* satisfies the natural transformation condition $(g \times f) \circ \text{swap} = \text{swap} \circ (f \times g)$
2. Prove that the polymorphic function *length* is a natural transformation. It is defined as the following:

$$\begin{aligned}
 \text{length} : [A] &\rightarrow \text{Int} \\
 \text{length } [] &= 0 \\
 \text{length } (x : xs) &= 1 + \text{length } xs
 \end{aligned}$$

3. Natural transformation is composable. Consider two natural transformations $\mathbf{F} \xrightarrow{\phi} \mathbf{G}$ and $\mathbf{G} \xrightarrow{\psi} \mathbf{H}$. For any arrow $A \xrightarrow{f} B$, draw the diagram for their composition, and list the commutative condition.

4.5 Data types

With the basic concepts like categories, functors, and natural transformation, we can realize complex data types.

4.5.1 Initial object and terminal object

We start from the two simplest data types, *initial object* and *final object*. They are symmetric like the left and right hands. They are simple, but not easy.

Definition 4.5.1. In category \mathbf{C} , if there is a special object S , such that for every object A , there is a unique arrow

$$S \longrightarrow A \qquad A \longrightarrow S$$

In other words, S has a unique arrow that

points to every other object pointed from every other object

We call this special object S

initial object final object

Sometimes, a final object is said to be *terminal*.

Traditionally, people use 0 to represent initial object, and use 1 for final object. We can also write that, for every object A , there is a unique arrow:

$$0 \longrightarrow A \qquad A \longrightarrow 1$$

Why are the initial and final objects symmetric? Suppose S is the initial object of category \mathbf{C} , by reversing the direction for all the arrows, then S becomes the final object of category \mathbf{C}^{op} . There may or may not be initial or final object in a category. A category can have one without the other, or have both. If it has both then these objects may or may not be the same. Any two initial objects (or final objects) of a category are uniquely isomorphic.

We only prove the isomorphic uniqueness for initial objects, the proof for final objects can be obtained through the symmetry.

Proof. Beside the initial object 0 , suppose there is another initial object $0'$. For the initial object 0 , there must exist an arrow f from 0 to $0'$ according to the definition of initial object; on the other hand, for the initial object $0'$, there must also exist an arrow g from $0'$ to 0 . According to the category axiom, there must be an identity arrow id_0 self-pointed to 0 (also called endo-arrow), and also an identity arrow $id_{0'}$ self-pointed to $0'$. Since the identity arrow is unique, we have:

$$id_0 = f \circ g \quad \text{and} \quad id_{0'} = g \circ f$$

This relation is illustrated in the below diagram.

$$\begin{array}{ccc}
 id_0 & \begin{array}{c} \text{---} \\ \text{---} \end{array} & 0 \xrightarrow{f} 0' \xleftarrow{g} id_{0'}
 \end{array}$$

Thus proved 0 is isomorphic to $0'$. In other words, the initial object is isomorphic unique. \square

This is the reason we usually use the word *the* initial object rather than an initial object. In the same way that two final objects are uniquely isomorphic, and we usually use the word *the* final object.

Specially, if the initial object is also the final object, we call it *zero object* or *null object*. A category may not necessarily have the zero object.

We'll next use some examples to understand what data types the initial and final objects correspond to.

Example 4.5.1. Consider a partial order set. The arrow is the ordering relation. If there is a minimum element, then it is the initial object. Similarly, if there is a maximum element, then it is the final object. Use finite figures in the duck family for example, in poset {Huey, Thelma Duck, Quackmore Duck, Grandma Duck}, with the ancestral ordering, Donald's nephew Huey is the initial object, and Grandma Duck is the final object. While for the Fibonacci numbers {1, 1, 2, 3, 5, 8, ...}, the ordering is less than or equal. 1 is the minimum number, hence it is the initial object; but there is no final object. Note that, there are two 1s in Fibonacci numbers, however, they are isomorphic under the less than or equal ordering. Consider the poset of real numbers R . The ordering is still less than or equal. There is neither the minimum nor the maximum number, hence there is not initial or final object. Consider the poset of all the figures in the duck family tree shown in figure 4.6, again, with the ancestral ordering. As there is no common ancestor, so it has not initial or final object.

Example 4.5.2. Consider the category of all groups \mathbf{Grp} . The trivial group $\{e\}$ only contains the identity element (see the previous chapter). The arrow is the morphism of groups. Any morphism sends the identity element in one group to the identity element in another group. Hence from $\{e\}$ to any group G , $e \mapsto e_G$ holds, where e_G is the identity element of G . There is a unique arrow from $\{e\}$ to any group.

$$\{e\} \longrightarrow G$$

On the other hand, from any group G , there exists a unique morphism, that sends every element to e , i.e. $\forall x \in G, x \mapsto e$. Hence there is a unique arrow from any group G to $\{e\}$.

$$G \longrightarrow \{e\}$$

Since $\{e\}$ is both initial and final object, it is the zero object. Particularly, observe the below arrow composition:

$$G \longrightarrow \{e\} \longrightarrow G'$$

The composition results a zero arrow, it connects the three groups G , $\{e\}$, and G' together. All elements in G are sent to element e , then next sent to $e_{G'}$, as shown in below figure. This is where the name 'zero' object comes from.

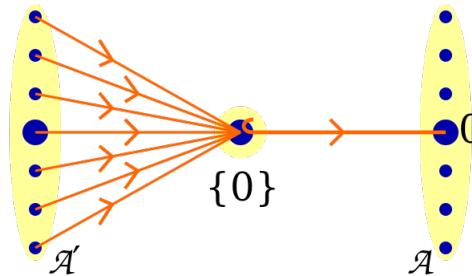


Figure 4.12: Zero object

It does not matter what we call the only element in the trivial group. It can be e , can be 1 (the trivial subgroup of the multiplicative integer group for example), can be 0 (the trivial subgroup of the additive integer group for example), can be I (the identity matrix of the multiplicative square matrix group for example), can be (1) (the identity transformation of the permutation group for example), can be id ... They are all isomorphic equivalent.

Example 4.5.3. Let's see a little bit more difficult example, the category of all sets **Set**, with the total functions as arrows. It's easy to find the final object, which is the singleton set (the set that contains only one object) $\{\star\}$. Similar to the example of groups, for any set S , we can send all elements to the only element in the singleton set: $\forall x \in S, x \mapsto \star$. Obviously, the arrow is unique.

$$S \longrightarrow \{\star\}$$

However, here comes the problem: How to map the empty set \emptyset to $\{\star\}$? In fact, the empty set²³ is the initial object in the category of set. As for any set S , we can define arrow

$$\emptyset \xrightarrow{f} S$$

Please take a while to think about this point. Let's consider the identity arrow id of the empty set, $\emptyset \xrightarrow{id_\emptyset} \emptyset$. According to the category axiom, every object has id arrow. The empty set is the object that has unique arrow to every set (include the empty set itself). The only special thing is that, the total function as the arrow has not any arguments. It can not be written as $f(x) : x \mapsto y$.

To answer the above question, since there is unique arrow from the empty set to any set, there is also unique arrow from empty set to $\{\star\}$. Hence there is unique arrow from every set (includes the empty set) to $\{\star\}$. Therefore, $\{\star\}$ is the final object in the category of set.

Why can't we let the singleton set $\{\star\}$ be the initial object like what we do for groups? The reason is because the arrow from $\{\star\}$ to set S may not be unique. Consider a set $S = \{x, y, z, \dots\}$, we can define the arrow (total function) $\{\star\} \xrightarrow{\vec{x}} S$, that sends the only element to x ; we can also define the arrow $\{\star\} \xrightarrow{\vec{y}} S$ or $\{\star\} \xrightarrow{\vec{z}} S$ to send it to other elements.

We intend to choose the symbol $\{\star\}$ to emphasize that, it does not matter what the element is, as far as it is a singleton set. Because all singleton sets are isomorphic equivalent.

The arrow from the final object $\{\star\}$ to a set S

$$\{\star\} \longrightarrow S$$

also has particular meaning. It means we can select an element from the set S , like the arrow \vec{x} we defined above. It selects x from S . For this reason, we call it *selection arrow*.

Example 4.5.4. The type system in programming environment is the category of set **Set** essentially. A data type is a set. For example, *Int* type is the set of all integers; Boolean type is the set with two elements $\{True, False\}$. We know that the final object is $\{\star\}$ in the category of sets. What data type is corresponding to the final object? Since the final object is isomorphic unique, therefore, any data type with only one value is the final object in programming.

We can define a dedicated data type, named as “()”, with only one element, also named as “()”.

data	<code>() = ()</code>
-------------	------------------------

²³The great french mathematician in the 20th Centry, André Weil said he was responsible for the null set symbol \emptyset , and that it came from the Norwegian alphabet, which he alone among the Bourbaki group was familiar with.

It is isomorphic equivalent to any singleton set, hence $\{\star\} = \{()\}$. We will see the advantage of choosing the “ $()$ ” symbol later. We can define the arrow (total function) from any data type (set) to this final object as below:

```
unit :: a → ()
unit _ = ()
```

We can also define the final object with any singleton data type, since it is isomorphic equivalent to $()$ ²⁴.

```
data Singleton = S
proj :: a → Singleton
proj _ = S
```

It looks like a constant function, which maps any value to a constant. However, not all constant functions point to the final object. For example the following two constant functions:

```
yes :: a → Bool
yes _ = True

no :: a → Bool
no _ = False
```

As the *Bool* data type contains two values, any other data type has two arrows *yes*, and *no* point to *Bool*, hence it does not satisfy the uniqueness condition for the arrow to the final object.

In the category of sets, the initial object is the empty set \emptyset . What is the corresponded data type in programming? Since the data type is essentially set, the empty set means a data type without any values. In programming, it means we declare a data type without defining it.

For example, we can declare a data type **Void**, but leave it without any values:

```
data Void
```

Thus **Void** represents an empty set. However, the initial object must have a unique arrow to any other object. It requires us to define a function from **Void** to all other data types.

```
absurd :: Void → a
absurd _ = undefined
```

The implementation does not matters here, because there does not exist a real argument to call this function. It is quite OK to implement it as this:

```
absurd :: Void → a
absurd a = case a of {}
```

We can also define the initial object by other means as far as they are isomorphic equivalent. The method is to only declare the type without define any values. For example:

```
data Empty
f :: Empty → a
f _ = undefined
iso :: Void → Empty
```

²⁴Many programming environments, like C++, Java, Scala, etc. support defining singleton object. We can also treat the force type casting as an arrow, then it also forms the final object.

```
iso _ = undefined
iso' :: Empty → Void
iso' _ = undefined
```

Here we explicitly define *iso* and *iso'* to form the isomorphism between **Empty** and **Void**. It is easy to verify that *iso* · *iso'* = *id*.

In the category of sets, we define the arrow from the final object to any set as the selection arrow. What does it mean in programming? Set corresponds to type. It means we can select a specific value from any type. The below example selects 0 and 1 from *Int* type respectively:

```
zero :: () → Int
zero () = 0

one :: () → Int
one () = 1
```

We see the advantage of choosing the “()” symbol. When call the function, it looks like we pass null argument to it: *zero* () returns 0, and *one* () returns 1.

Exercise 4.5

1. In the poset example, we say if there exists the minimum (or the maximum) element, then the minimum (or the maximum) is the initial object (or the final object). Consider the category of all posets **Poset**, if there exists the initial object, what is it? If there exists the final object, what is it?
2. In the Peano category **Pno** (see exercise 2 in section 1), what is the initial object in form (A, f, z) ? What is the final object?

4.5.2 Exponentials

The initial and final objects are similar to 0 and 1, the product and coproduct are similar to \times and $+$, if we can find something similar to exponentials, then we will be empowered with the basic arithmetic in the abstract category world. Further, we can even develop something ‘isomorphic’ to polynomials.

Observe a binary function $f(x, y) = z$ defined for natural numbers, for example $f(x, y) = x^2 + 2y + 1$. Its type is $f : N \times N \rightarrow N$. It’s natural to think about writing the type as:

$$f : N^2 \rightarrow N$$

For function *f*, can we take N^2 as a whole object, but not two arguments? i.e. treat it as $f(x, y)$ but not $f(x, y)$. If we consider the argument as a ‘hole’, then the former is $f \bullet$, then put a pair (x, y) into that hole; while the latter is $f(\bullet, \bullet)$, then we put *x* and *y* to the two holes respectively. On the other hand, we introduced Currying in chapter 2. We can consider *f* as $f : N \rightarrow (N \rightarrow N)$. After passing *x* to it, *f x* returns another function. This new function sends one natural number to another number. It seems we can’t get the exponential concept with Currying directly. However, what if we consider the Curried result, which is $N \rightarrow N$, as a whole thing? we can name it as the ‘function object’. It’s still a bit complex since *N* is a infinite set. Let’s go back a step and restart with a simpler example, the *Bool* set with only two elements:

Example 4.5.5. There are infinite many arrows (functions) from *Bool* to *Int*. They form a set $\{Bool \xrightarrow{f} Int\}$. Let us select an element, an example function from it.

$$\begin{aligned}ord : \text{Bool} &\rightarrow \text{Int} \\ord \text{ False} &= 0 \\ord \text{ True} &= 1\end{aligned}$$

The result is a pair of numbers $(0, 1)$. Similar to this instance, we can write all the arrows in above set in this way:

$$\begin{aligned}f : \text{Bool} &\rightarrow \text{Int} \\f \text{ False} &= \dots \\f \text{ True} &= \dots\end{aligned}$$

No matter how f varies, the result is always a pair of numbers (a, b) . We say the set of arrow f is isomorphic to the set of integer pair (a, b) . That is to say:

$$\begin{aligned}\{\text{Bool} \xrightarrow{f} \text{Int}\} &= \{(a = f \text{ False}, b = f \text{ True})\} \\&= \text{Int} \times \text{Int} \\&= \text{Int}^2 = \text{Int}^{\text{Bool}}\end{aligned}$$

The set (type) of arrows ²⁵: $\text{Bool} \rightarrow \text{Int}$, corresponds to an exponential Int^{Bool} . It means $\text{Bool} \rightarrow \text{Int} = \text{Int}^{\text{Bool}}$. Why can we use 2 to substitute Bool in the above reasoning to convert Int^2 to Int^{Bool} ? The reason is isomorphism. The equal symbol actually means isomorphic equivalence. Strictly speaking, we should use the “ \cong ” symbol, but not “ $=$ ”. Int^2 represents the product of two integers, which is the set of Int pairs. The set of Int pairs can be considered as a map. The map is from a set named as **2** with 2 elements of index $\{0, 1\}$ to Int .

$$\{0, 1\} \xrightarrow{f} \text{Int} = \{(f(0), f(1))\} = \text{Int}^2$$

And the index set **2** = $\{0, 1\}$ is isomorphic to Bool (under the function ord for example).

Example 4.5.6. Let's see another example. Consider the set of all the arrows (functions) $\text{Char} \rightarrow \text{Bool}$ from characters Char to the Boolean values Bool . There are many functions in this set. For example $\text{isDigit}(c)$ check whether the passed in character is digit or not. One implementation can be this (not practical but possible):

```
isDigit : Char → Bool
...
isDigit '0' = True
isDigit '1' = True
...
isDigit '9' = True
isDigit 'a' = False
isDigit 'b' = False
...
```

Although this is a naive implementation, it reflects such fact: If there are 256 different chars (ASCII code of English for example), then the result of isDigit function is isomorphic to a 256-tuple, values of $(\text{False}, \dots, \text{True}, \text{True}, \dots, \text{True}, \text{False}, \dots)$, where the positions corresponding to digit characters are True, the rest are False. Among the varies of functions of type $\text{Char} \rightarrow \text{Bool}$, like isUpper , isLower , isWhitespace etc. everyone corresponds to a specific tuple. For example in the tuple corresponding to isUpper , the positions for the upper case characters are True, the rest are False. Although there are infinite many ways to define a function from Char to Bool , from the result perspective, there are only 2^{256} different functions essentially. It means $\text{Char} \rightarrow \text{Bool}$ is isomorphic to $\text{Bool}^{\text{Char}}$, which is the set of 256-tuple Boolean values.

²⁵Strictly speaking, we should remove the pair of curly brackets of the arrow. We add the them only to make it easy to understand.

Next, we expand the examples to the infinite set. We mentioned the Curried function $f : N \rightarrow (N \rightarrow N)$ maps from the natural numbers N to ‘function objects’. Denote the function object as (\Rightarrow) , then the type of f is $N \xrightarrow{f} (\Rightarrow)$. It maps from the index set $\{0, 1, \dots\}$ to (\Rightarrow) , which forms a set of infinite long tuples. Every value in the tuple is a function object. We denote it as $(\Rightarrow)^N$.

In general, we denote the set of function $f : B \rightarrow C$ in exponential form C^B . Select a function $f \in C^B$, and an element $b \in B$, then we can use a special *apply* function that applies f to b , that gives a value $c = f(b)$ in C . As this²⁶:

$$\text{apply}(f, b) = f(b)$$

Readers may ask: where is symbol A ? We actually need A for other purpose when taking Currying into account. For the binary function $g : A \times B \rightarrow C$, when passing an element a in A , we obtained a Curried function $g(a, \bullet) : B \rightarrow C$, which is a function object belongs C^B . Hence for every binary function $A \times B \xrightarrow{g} C$, there is a unique unary function $A \xrightarrow{\lambda g} C^B$ sends $a \in A$ to $g(a, \bullet) : B \rightarrow C$. We call λg the exponential transpose of g ²⁷. And the following equation holds:

$$\text{apply}(\lambda g(a), b) = g(a, b)$$

That is to say, *apply* combines a function object $\lambda g(a)$ of type C^B and an argument b of type B together, and gives the result c of type C . Therefore, the type of *apply* is:

$$C^B \times B \xrightarrow{\text{apply}} C$$

Now we can give the complete definition of exponentials (or exponential objects).

Definition 4.5.2. If a category \mathbf{C} has final object and supports product, then an *exponential object* is a pair of object and arrow

$$(C^B, \text{apply})$$

For any object A and arrow $A \times B \xrightarrow{g} C$, there is a unique transpose arrow

$$A \xrightarrow{\lambda g} C^B$$

such that the below diagram commutes

$$\begin{array}{ccc} A & & A \times B \\ \downarrow \lambda g & & \downarrow \lambda g \times id_B \\ C^B & & C^B \times B \\ & & \xrightarrow{\text{apply}} C \end{array}$$

That is:

$$\text{apply} \circ (\lambda g \times id_B) = g$$

²⁶Some materials, like [46] pp. 111 - 112, and [47] use *eval* to highlight this evaluation process. We adopt the name in [6] pp. 72 to follow the tradition in Lisp.

²⁷Some materials use \bar{g} for exponential transpose. From the introduction about λ in chapter 2, especially the interesting story about Church and his publisher, λg looks vivid. It means $a \mapsto \lambda \bullet \cdot g(a, \bullet)$.

We didn't provide the complete definition of Currying in chapter 2. We are able to define *curry* with the help of exponentials:

$$\begin{aligned} \text{curry} : (A \times B \rightarrow C) &\rightarrow A \rightarrow (C^B) \\ \text{curry } g &= \lambda g \end{aligned}$$

Therefore, $\text{curry } g$ is the exponential transpose of g . Substitute this definition into the above diagram, we have:

$$\text{apply} \circ (\text{curry } g \times \text{id}) = g$$

In other words, we obtain such a universal property:

$$f = \text{curry } g \equiv \text{apply} \circ (f \times \text{id}) = g$$

And we can also see why the exponential transpose arrow is unique. Suppose there exists another arrow $A \xrightarrow{h} C^B$, such that $\text{apply} \circ (h \times \text{id}) = g$. According to the above universal property, we immediately get $h = \text{curry } g$.

We can also consider the exponentials as a category. Given a category \mathbf{C} , when fix object B, C , we can form a category \mathbf{Exp} . In this category, all objects are arrows in the form of $A \times B \rightarrow C$, and the arrows are defined as:

$$\begin{array}{ccc} A & & A \times B \xrightarrow{h} C \\ f \downarrow & & \downarrow j \\ D & & D \times B \xrightarrow{k} C \end{array}$$

If there is arrow $A \xrightarrow{f} D$ in category \mathbf{C} , then the arrow in \mathbf{Exp} is $h \xrightarrow{j} k$. The arrows commute if and only if when we combine the C on the right in above diagram:

$$\begin{array}{ccc} A & & A \times B \\ f \downarrow & & \downarrow f \times \text{id}_B \\ D & & D \times B \xrightarrow{k} C \\ & & \searrow h \\ & & C \end{array}$$

The diagram commutes when $k \circ (f \times \text{id}_B) = h$. There exists the final object in category \mathbf{Exp} , which exactly is $C^B \times B \xrightarrow{\text{apply}} C$. Let's verify it. According to the definition of exponential, from any object $A \times B \xrightarrow{g} C$, there is *apply* arrow $\lambda g = \text{curry } g$. On the other hand, the endo arrow to the final object must be *id*, hence we obtain the *reflection law*:

$$\text{curry apply} = \text{id}$$

Exercise 4.6

1. Verify that \mathbf{Exp} is a category. What is the *id* arrow and arrow composition in it?
2. In the reflection law $\text{curry apply} = \text{id}$, what is the subscript of the *id* arrow? Please prove it with another method.
3. We define the equation

$$(\text{curry } f) \circ g = \text{curry}(f \circ (g \times \text{id}))$$

as the fusion law for Currying. Draw the diagram and prove it.

4.5.3 Cartesian closed and object arithmetic

With the initial object as 0, the final object as 1, coproduct as add, product as multiplication, exponentials as powers, we are capable to do arithmetic or even form polynomials for categories. But hold on. As we warned at the beginning of chapter 3, we should always ask ourselves what the applicable scope for the abstraction is, when the abstraction will be invalid.

Not all categories have the initial or final object, and do not necessary have exponentials. If a category has finite product, for any object A and B , there is A^B , it is called *Cartesian closed*. A Cartesian closed category must have:

1. A final object (1);
2. Every two objects have product (\times);
3. Every two objects have exponential (A^B).

We can either consider the final object 1 as the 0th power of an object $A^0 = 1$, or the product of zero objects. Fortunately, the category of programming, with sets and total functions, is Cartesian closed. A Cartesian closed category can model the simply typed λ calculus (see chapter 2), hence serves as the foundation of all the programming languages with types([45] pp. 148).

If a Cartesian closed category also supports the dual of the final object, which the initial object; the dual of product, which is the coproduct; and also supports the distribution law of product over coproduct, then it is *Bicartesian closed*.

A Bicartesian closed category satisfies the following conditions in additional:

4. The initial object (0);
5. Every two objects have coproduct (+);
6. Product can be distributed from both sides to coproduct:

$$\begin{aligned} A \times (B + C) &= A \times B + A \times C \\ (B + C) \times A &= B \times A + C \times A \end{aligned}$$

Now we are ready to explain what the basic arithmetic operations mean in programming, which is a Cartesian closed category. It is called equational theory.

0th Power

$$A^0 = 1$$

0 represents the initial object, 1 represents the final object. The 0th power of A represents a set of all arrows $0 \rightarrow A$. Since 0 is the initial object, it has the unique (only one) arrow to any object. Therefore, set $\{0 \rightarrow A\}$ contains only one object. It is a singleton. While the singleton set $\{\star\}$ is exactly the final object 1 in the category of sets. The equal symbol used in the below reasoning should be thought as isomorphism.

$$\begin{aligned} A^0 &= \{0 \rightarrow A\} && \text{Definition of exponential} \\ &= \{\star\} && \text{Uniqueness of arrow from initial object} \\ &= 1 && \{\star\} \text{ is the final object} \end{aligned}$$

Our common sense in number arithmetic that the 0th power of any number is 1 reflects same in categories.

Powers of 1

$$1^A = 1$$

1 represents the final object, hence the exponential object 1^A represents the set of all arrows from A to the final object, which is $\{A \rightarrow 1\}$. According to the definition of final object, there is unique arrow from any object to the final object, hence there is only one element in this set of arrow. While the set contains only one element is isomorphic to $\{\star\}$, which is exactly the final object in the category of set.

$$\begin{aligned} 1^A &= \{A \rightarrow 1\} && \text{Definition of exponential} \\ &= \{\star\} && \text{Uniqueness of the arrow to the final object} \\ &= 1 && \{\star\} \text{ is the final object} \end{aligned}$$

First power

$$A^1 = A$$

This is exactly the ‘[selection arrow](#)’ introduced above. 1 is the final object, hence the exponential A^1 represents the set of arrows from the final object to A , which is $\{1 \rightarrow A\}$. If A is a set, then we can construct a function from the final object to 1 for all element $a \in A$:

$$f_a : 1 \rightarrow a$$

Such function can select an element a from A hence is called selection function. The set of all such functions from 1 to A is $\{f_a : 1 \mapsto a | a \in A\}$, which is exactly the set $1 \rightarrow A$. On the other hand, set $\{f_a\}$ has one to one mapping to $A = \{a\}$, hence they are isomorphic.

$$\begin{aligned} A^1 &= \{1 \rightarrow A\} && \text{Definition of exponential} \\ &= \{f_a : 1 \mapsto a | a \in A\} && \text{the set of maps from } 1 \text{ to } A \\ &= \{a | a \in A\} = A && \text{one to one mapping isomorphic} \end{aligned}$$

Exponentials of sums

$$A^{B+C} = A^B \times A^C$$

The exponential A^{B+C} represents the set of arrows from the coproduct $B + C$ to A , which is $\{B + C \rightarrow A\}$. Let’s explain it through [Either](#) $B C$ we introduced above²⁸. For any function of type [Either](#) $B C \rightarrow A$, we can implement it in the form:

$$\begin{aligned} f : \text{Either } B C &\rightarrow A \\ f \text{ left } b &= \dots \\ f \text{ right } c &= \dots \end{aligned}$$

It means all such functions can be considered as a pair of maps $(b \mapsto a_1, c \mapsto a_2)$, and it is exactly the product of $B \rightarrow A$ and $C \rightarrow A$. Therefore, $\{B + C \rightarrow A\} = \{B \rightarrow$

²⁸Alternatively, we can reason it with tags:

$$\begin{aligned} f : B + C &\rightarrow A \\ f(b, 0) &= \dots \\ f(c, 1) &= \dots \end{aligned}$$

$A\} \times \{C \rightarrow A\}$. On the other hand, since the exponential of $\{B \rightarrow A\}$ is A^B , and the exponential of $\{C \rightarrow A\}$ is A^C , it explains the exponentials of sum:

$$\begin{aligned}
 A^{B+C} &= \{B + C \rightarrow A\} && \text{Definition of exponentials} \\
 &= \{(b \mapsto a_1, c \mapsto a_2) | a_1, a_2 \in A, b \in B, c \in C\} && \text{Pair of arrows from } B \text{ and } C \text{ to } A \\
 &= \{B \rightarrow A\} \times \{C \rightarrow A\} && \text{Cartesian product} \\
 &= A^B \times A^C && \text{Reverse of exponentials}
 \end{aligned}$$

Exponentials of exponentials

$$(A^B)^C = A^{B \times C}$$

On the right side of the equation, the exponential $A^{B \times C}$ is actually the set of all binary functions of $B \times C \xrightarrow{g} A$. It is obviously a natural isomorphism when swap the product to $C \times B \xrightarrow{g \circ \text{swap}} A$ (refer to the *swap* in the section of natural transformation). The Curried form is $\{\text{curry}(g \circ \text{swap})\} = \{C \rightarrow A^B\}$. Then represent it with exponentials, and we obtain $(A^B)^C$.

$$\begin{aligned}
 A^{B \times C} &= \{B \times C \xrightarrow{g} A\} && \text{Definition of exponentials} \\
 &= \{C \times B \xrightarrow{g \circ \text{swap}} A\} && \text{Natural isomorphism} \\
 &= \{C \xrightarrow{\text{curry}(g \circ \text{swap})} A^B\} && \text{Currying is isomorphic} \\
 &= (A^B)^C && \text{Reverse of exponentials}
 \end{aligned}$$

Exponentials over products

$$(A \times B)^C = A^C \times B^C$$

The exponential $(A \times B)^C$ is the set of arrows of $C \rightarrow A \times B$. It is equivalent to the set of functions that returns a pair of values $\{c \mapsto (a, b)\}$, where $c \in C, a \in A, b \in B$. It is obviously isomorphic to $\{(c \mapsto a, c \mapsto b)\}$, which is exactly the product of arrows $C \rightarrow A$ and $C \rightarrow B$. We then write the two arrows in exponentials thus obtain the product of exponentials.

$$\begin{aligned}
 (A \times B)^C &= \{C \rightarrow A \times B\} && \text{Definition of exponential} \\
 &= \{c \mapsto (a, b) | a \in A, b \in B, c \in C\} && \text{Set of arrows} \\
 &= \{(c \mapsto a, c \mapsto b)\} && \text{Pair of arrows} \\
 &= \{C \rightarrow A\} \times \{C \rightarrow B\} && \text{Cartesian product} \\
 &= A^C \times B^C && \text{Reverse of exponentials}
 \end{aligned}$$

4.5.4 Polynomial functors

The arithmetic we introduced applies to objects in Cartesian closed categories. What if we take functors into consideration? As functors apply to both objects and arrows, it leads to the concept of polynomial functors. A polynomial functor is built from constant functors (refer to [The ‘black hole’ functor example](#)), product functors, and coproduct functors recursively.

- The identity functor id , and constant functor \mathbf{K}_A are polynomial functors;

- if functor \mathbf{F} and \mathbf{G} are polynomial functors, then their composition \mathbf{FG} , sum $\mathbf{F} + \mathbf{G}$, and product $\mathbf{F} \times \mathbf{G}$ are also polynomial functors. Where the sum and product are defined as below:

$$\begin{aligned} (\mathbf{F} + \mathbf{G})h &= \mathbf{F}h + \mathbf{G}h \\ (\mathbf{F} \times \mathbf{G})h &= \mathbf{F}h \times \mathbf{G}h \end{aligned}$$

For example, if functor \mathbf{F} is defined as below for objects and arrows:

$$\begin{cases} \text{Object: } \mathbf{F}X = A + X \times A \\ \text{Arrows: } \mathbf{F}h = id_A + h \times id_A \end{cases}$$

Where A is some fixed object. Then functor \mathbf{F} is a polynomial functor. It can be represented as a polynomial:

$$\mathbf{F} = \mathbf{K}_A + (id \times \mathbf{K}_A)$$

4.5.5 F-algebra

In order to construct complex algebraic data types that support recursive structure, we need the last corner stone, the F-algebra. Think about the concepts in abstract algebra, like monoid, group, ring, field, etc., they are not only abstract objects, but also have structures. It is the relation among the structures, makes them different from the traditional concrete things, like numbers, points, lines, and planes. When we understand the structures and their relations well, we actually understand all the things of the same structure and relations. Not only for numbers, points, lines, and planes, but also for ‘tables, chairs, and beer mugs’ as Hilbert said.

Example 4.5.7. Start from the simple example of monoid, we introduce the F-algebra step by step. A monoid is a set M , defined with the identity element and the associative binary operation.

Denote the binary operation as \oplus , the identity element as 1_M , we can list the two monoid axioms as the following:

$$\begin{cases} \text{Associativity: } (x \oplus y) \oplus z = x \oplus (y \oplus z), \forall x, y, z \in M \\ \text{Identity element: } x \oplus 1_M = 1_M \oplus x = x, \forall x \in M \end{cases}$$

In the first step, we represent the binary operation \oplus as a binary function, represent 1_M as a selection function. Define:

$$\begin{cases} m(x, y) = x \oplus y \\ e() = 1_M \end{cases}$$

The types of these two arrows are:

$$\begin{cases} \text{binary operation: } M \times M \rightarrow M \\ \text{selection: } 1 \rightarrow M \end{cases} \begin{array}{ll} \text{type} & \text{exponential} \\ M^{M \times M} & M^1 \end{array}$$

The first arrow means, applying the binary operation to any two elements in the monoid, the result is still an element of the monoid, hence the binary operation is closed; the second arrow is the selection function. 1 is the final object²⁹.

²⁹We intend to use ‘()’ for the final object, as $1 = \{\star\} = \{\}\}$ in terms of isomorphism. The advantage is that, $e()$ looks like a function call without any arguments, in fact, it accepts the singleton element of the final object as the argument.

Now we can substitute the monoid axioms with function m and e :

$$\begin{cases} \text{Associativity: } m(m(x, y), z) = m(x, m(y, z)), \forall x, y, z \in M \\ \text{Identity element: } m(x, e()) = m(e(), x) = x, \forall x \in M \end{cases}$$

In the second step, we remove all the concrete elements like x, y, z , and object M . Then we obtain the monoid axioms purely in function (arrow) form:

$$\begin{cases} \text{Associativity: } m \circ (m, id) = m \circ (id, m) \\ \text{Identity element: } m \circ (id, e) = m \circ (e, id) = id \end{cases}$$

These two axioms indicate the following two diagrams commute.

$$\begin{array}{ccc} M \times M \times M & \xrightarrow{(m, id)} & M \times M \\ (id, m) \downarrow & & \downarrow m \\ M \times M & \xrightarrow{m} & M \end{array}$$

(a) Associativity

$$\begin{array}{ccc} 1 \times M & \xrightarrow{id(\cong)} & M \\ (e, id) \downarrow & & \downarrow m \\ M \times M & \xrightarrow{m} & M \\ (id, e) \uparrow & & \uparrow id(\cong) \\ M \times 1 & & \end{array}$$

(b) Identity element

Figure 4.13: The diagrams for monoid axioms

Therefore, any monoid can be represented with a tuple (M, m, e) , where M is the set, m is the binary operation function, and e is the identity element selection function.

We come to the third step. (M, m, e) does not only specify the set of the monoid, but also specifies the binary operation and identity element on top of this set. It completely defines the algebraic structure of monoid. All the possible combinations of arrow m and e that form the monoid are products of two exponentials:

$$M^{M \times M} \times M^1 = M^{M \times M + 1}$$

Next, we turn the right side exponentials to the arrow form. For all the monoids defined on top of set M , the algebraic operations must be one of the following:

$$\begin{array}{lll} \alpha : 1 + M \times M & \longrightarrow & M \\ 1 & \longmapsto & 1 \\ (x, y) & \longmapsto & x \oplus y \end{array} \begin{array}{ll} \text{sum (coproduct)} \\ \text{identity element} \\ \text{binary operation} \end{array}$$

This relation can be represented as the sum through coproduct $\alpha = e + m$, which is the polynomial functor $\mathbf{F}M = 1 + M \times M$.

In summary, for the monoid example, the algebraic structure is consist of three parts:

1. Object M is the set that carries the algebraic structure for the monoid. It is called *carrier object*;
2. Functor \mathbf{F} defines the algebraic operations. It is a polynomial functor $\mathbf{F}M = 1 + M \times M$;
3. Arrow $\mathbf{F}M \xrightarrow{\alpha} M$. It is the sum (coproduct) of the identity arrow e and the binary operation arrow m , i.e. $\alpha = e + m$.

We say it defines the F -algebra (M, α) of the monoid.

For programming, we can define the type of the F -algebra arrow:

```
type Algebra f a = f a → a
```

It gives an alias to the arrow $\mathbf{F}A \rightarrow A$ essentially, named as **Algebra F, A**. For the monoid, we also need define a functor³⁰:

```
data MonoidF a = MEmptyF | MAppendF a a
```

When **a** is string for example, we can implement the arrow of the F -algebra (**String**, **evals**) as below:

```
evals :: Algebra MonoidF String
evals MEmpty = e ()
evals (MAppendF s1 s2) = m s1 s2

e :: () → String
e () = ""

m :: String → String → String
m = (++)
```

We can further simplify it by embedding **e** and **m** in **evals**:

```
evals :: Algebra MonoidF String
evals MEmpty = ""
evals (MAppendF s1 s2) = s1 ++ s2
```

evals is one implementation to the arrow $\alpha : \mathbf{F}A \rightarrow A$, where **F** is **MonoidF**, and **A** is **String**. There can be other implementations as far as they satisfy the monoid axioms.

Example 4.5.8. Compare to monoid, group need an additional inverse element axiom. Similar to the previous example, we will develop the F -algebra for groups step by step. First, we list the three axioms for group G . Denote the identity element as 1_G , use the point symbol for binary operator, and use $(\cdot)^{-1}$ for the inverse element:

$$\begin{cases} \text{Associativity: } (x \cdot y) \cdot z = x \cdot (y \cdot z), \forall x, y, z \in G \\ \text{Identity element: } x \cdot 1_G = 1_G \cdot x = x, \forall x \in G \\ \text{Inverse element: } x \cdot x^{-1} = x^{-1} \cdot x = 1_G, \forall x \in G \end{cases}$$

In the first step, we represent the binary operation, the identity element, and the reverse operation as functions. Define:

$$\begin{cases} m(x, y) = x \cdot y \\ e() = 1_G \\ i(x) = x^{-1} \end{cases}$$

The corresponding function type and exponentials are:

$$\begin{cases} \text{binary operation: } G \times G \rightarrow G & G^{G \times G} \\ \text{selection: } & G^1 \\ \text{reverse: } & G^G \end{cases}$$

The first two arrows are same as monoid. The third one tells that the reverse element still belongs to the group. Next we substitute the 1_G , the point, and the reverse symbols in the group axioms with e , m , and i .

³⁰There is a definition of monoid in Haskell standard library (see the appendix of this chapter). However, it is the definition of the monoid algebraic structure, but not the definition of the monoid functor

$$\begin{cases} \text{Associativity: } & m(m(x, y), z) = m(x, m(y, z)), \forall x, y, z \in G \\ \text{Identity element: } & m(x, e()) = m(e(), x) = x, \forall x \in G \\ \text{Reverse element: } & m(x, i(x)) = m(i(x), x) = e(), \forall x \in G \end{cases}$$

In the second step, we remove the concrete elements x, y, z , and set G to represent the group axioms purely in arrows:

$$\begin{cases} \text{Associativity: } & m \circ (m, id) = m \circ (id, m) \\ \text{Identity element: } & m \circ (id, e) = m \circ (e, id) = id \\ \text{Reverse element: } & m \circ (id, i) = m \circ (i, id) = e \end{cases}$$

It means every group can be represented in a tuple (G, m, e, i) . In the third step, we use the coproduct of m, e, i for the sum $\alpha = e + m + i$, then through the polynomial functor $\mathbf{F}A = 1 + A + A \times A$, and let $A = G$ to describe the algebraic operations on G .

$$\begin{array}{lll} \alpha : 1 + A + A \times A & \longrightarrow & A \\ 1 & \longmapsto & 1 \\ x & \longmapsto & x^{-1} \\ (x, y) & \longmapsto & x \cdot y \end{array} \begin{array}{ll} \text{sum (coproduct)} \\ \text{identity element} \\ \text{reverse element} \\ \text{binary operation} \end{array}$$

Therefore, the algebraic structure of group is consist of three parts:

1. The carrier object G , which is the set furnished with the algebraic structure;
2. Polynomial functor $\mathbf{F}A = 1 + A + A \times A$, which defined the algebraic operations on the group;
3. Arrow $\mathbf{F}A \xrightarrow{\alpha = e + m + i} A$. It is the sum of the identity element arrow e , the binary operation arrow m , and the reverse element arrow i .

We say it defines the F-algebra (G, α) for group.

It's ready to give the definition of F-algebra. There are many dual concepts appear in pairs. We benefit from them like buy one, get one. When define F-algebra, we obtain F-coalgebra at the same time. We will post-pone the examples of F-coalgebra in chapter 6 when introduce about the infinity.

Definition 4.5.3. Let \mathbf{C} be a category, $\mathbf{C} \xrightarrow{\mathbf{F}} \mathbf{C}$ is an endo-functor of category \mathbf{C} . For the object A and morphism α in this category, arrow:

$$\mathbf{F}A \xrightarrow{\alpha} A \quad A \xrightarrow{\alpha} \mathbf{F}A$$

forms a pair (A, α) . It is called

$$\begin{array}{ll} \text{F-algebra} & \text{F-coalgebra} \end{array}$$

Where A is the carrier object.

We can treat

$$\begin{array}{ll} \text{F-algebra } (A, \alpha) & \text{F-coalgebra } (A, \alpha) \end{array}$$

as object. When the context is clear, we denote the object as a pair (A, α) . The arrows between such objects are defined as the following:

Definition 4.5.4. F-morphism is the arrow between F-algebra or F-coalgebra objects:

$$(A, \alpha) \longrightarrow (B, \beta)$$

If the arrow $A \xrightarrow{f} B$ between the carrier objects make the below diagram commutes:

$$\begin{array}{ccc} \mathbf{F}A & \xrightarrow{\alpha} & A \\ \mathbf{F}(f) \downarrow & & \downarrow f \\ \mathbf{F}B & \xrightarrow{\beta} & B \end{array} \quad \begin{array}{ccc} A & \xrightarrow{\alpha} & \mathbf{F}A \\ f \downarrow & & \downarrow \mathbf{F}(f) \\ B & \xrightarrow{\beta} & \mathbf{F}B \end{array}$$

Which means:

$$f \circ \alpha = \beta \circ \mathbf{F}(f) \quad \beta \circ f = \mathbf{F}(f) \circ \alpha$$

F-algebra and F-morphism, F-coalgebra and F-morphism form

F-algebra category $\mathbf{Alg}(\mathbf{F})$

F-coalgebra category $\mathbf{CoAlg}(\mathbf{F})$

respectively.

Exercise 4.7

1. Draw the diagram to illustrate the reverse element axiom for group.
2. Let p be a prime. Use the F-algebra to define the α arrow for the multiplicative group for integers modulo p (refer to the previous chapter for the definition of this group).
3. Define F-algebra for ring (refer to the previous chapter for definition of ring).
4. What is the *id* arrow for F-algebra category? What is the arrow composition?

Recursion and fixed point

We introduced Peano axioms for natural numbers and things that are isomorphic to natural numbers in chapter 1. In fact, we can use F-algebra to describe all natural number like things.

Consider a set A furnished with algebraic structure. Use the Fibonacci numbers introduced in chapter 1 for instance, where A is the set of pairs $(\mathbf{Int}, \mathbf{Int})$, the initial pair is $(1, 1)$, the successor function is $h(m, n) = (n, m + n)$.

In order to model this *kind* of algebraic structure in F-algebra, we need three things: functor, carrier object, and the α arrow. From the Peano axiom, natural number functor should be defined as below:

data $\mathbf{NatF} A = \mathbf{ZeroF} \mid \mathbf{SuccF} A$

This is a polynomial functor $\mathbf{NatF}A = 1 + A$. Here is an interesting question: let $A' = \mathbf{NatF}A$, substitute it into functor \mathbf{NatF} , then what will the $\mathbf{NatF}A'$ look like? It will look like applying the functor twice $\mathbf{NatF}(\mathbf{NatF}A)$. We can repeat this process to apply the functor three times as $\mathbf{NatF}(\mathbf{NatF}(\mathbf{NatF}A))$. Actually, we can repeat infinite many times, and name the type $\mathbf{NatF}(\mathbf{NatF}(\dots))$ as \mathbf{Nat} .

data $\mathbf{Nat} = \mathbf{NatF}(\mathbf{NatF}(\dots))$	Infinite many times
$= \mathbf{ZeroF} \mid \mathbf{SuccF} (\mathbf{SuccF}(\dots))$	Infinite many times of \mathbf{SuccF}
$= \mathbf{ZeroF} \mid \mathbf{SuccF} \mathbf{Nat}$	Infinite many times of \mathbf{SuccF} , named as \mathbf{Nat}
$= \mathbf{Zero} \mid \mathbf{Succ} \mathbf{Nat}$	Rename

It is exactly the type of natural numbers we defined in chapter 1. Let us list **Nat** and **NatF A** together:

```
data Nat = Zero | Succ Nat

data NatF A = ZeroF | SuccF A
```

This is a recursive functor. We've already met the similar concept in chapter 2. Applying the endo-functor to itself infinite many times gives the *fixed point*:

$$\mathbf{Fix F} = \mathbf{F}(\mathbf{Fix F})$$

The fixed point of endo-functor **F** is **Fix F**. When applying **F** to the fixed point, still gives the fixed point. Hence **Nat** is the fixed point of the functor **NatF**. Which means **Nat = Fix NatF**.

Exercise 4.8

1. Someone write the natural number like functor as the below recursive form. What do you think about it?

```
data NatF A = ZeroF | SuccF (NatF A)
```

2. We can define an α arrow for $\mathbf{NatF} \mathbf{Int} \rightarrow \mathbf{Int}$, named *eval*:

$$\begin{aligned} eval &: \mathbf{NatF} \mathbf{Int} \rightarrow \mathbf{Int} \\ eval \ ZeroF &= 0 \\ eval \ (SuccF n) &= n + 1 \end{aligned}$$

We can recursively substitute $A' = \mathbf{NatF} A$ to functor **NatF** by n times. We denote the functor obtained as $\mathbf{NatF}^n A$. Can we define the following α arrow?

$$eval : \mathbf{NatF}^n \mathbf{Int} \rightarrow \mathbf{Int}$$

Initial algebra and catamorphism

If there exists initial object in the category of F-algebra $\mathbf{Alg}(F)$, what are the properties for it? There must be unique arrow from the initial object to other objects, what kind of relationship does it represent? All the objects of the F-algebra category can be written as pair (A, α) . Using the F-algebra for natural number like things for example, in category $\mathbf{Alg}(\mathbf{NatF})$, all objects can be written as (A, α) , where **NatF** is defined as above section; A is the carrier object, $\mathbf{NatF} A \xrightarrow{\alpha} A$ is the arrow.

Fibonacci numbers $(\mathbf{Int} \times \mathbf{Int}, fib)$ form a F-algebra object. Where the carrier object is the product of integers, the arrow $\mathbf{NatF}(\mathbf{Int} \times \mathbf{Int}) \xrightarrow{fib} (\mathbf{Int} \times \mathbf{Int})$ is defined as:

$$\begin{aligned} fib &: \mathbf{NatF}(\mathbf{Int} \times \mathbf{Int}) \rightarrow (\mathbf{Int} \times \mathbf{Int}) \\ fib \ ZeroF &= (1, 1) \\ fib \ (SuccF (m, n)) &= (n, m + n) \end{aligned}$$

This definition is concise. We can also write it in the form of sum (coproduct) $fib = [start, next]$, where *start* always returns $(1, 1)$, and *next* $(m, n) = (n, m + n)$.

We denote the initial object 0 of category **NatF** as a pair (I, i) . There is unique arrow from the initial object to any object (A, α) . It means there exists arrow $I \xrightarrow{f} A$, such that the below diagram commutes³¹.

³¹For (I, i) , We use the upper case first character in word *initial* for the set, and the lower case *i* for the arrow of the initial object.

$$\begin{array}{ccc}
 \mathbf{NatF}A & \xrightarrow{\alpha} & A \\
 \mathbf{NatF}(f) \uparrow & & \uparrow f \\
 \mathbf{NatFI} & \xrightarrow{i} & I
 \end{array}$$

Since there is unique arrow from the initial object to *every* object, hence for the object formed by recursively applying the functor $\mathbf{NatF}(\mathbf{NatFI})$, there must be the unique arrow as well. The recursive F-algebra is furnished with three properties:

1. The common functor \mathbf{NatF} ;
2. The carrier object \mathbf{NatFI} ;
3. Arrow $\mathbf{NatF}(\mathbf{NatFI}) \rightarrow \mathbf{NatFI}$. Because functor applies to both object and arrow, by using functor \mathbf{NatF} , we can ‘lift’ the arrow i in the initial object. Hence the arrow for this recursive F-algebra is $\mathbf{NatF}(i)$.

With the above three properties, we can denote the recursive F-algebra as $(\mathbf{NatFI}, \mathbf{NatF}(i))$. Since there is unique arrow from the initial object to it, there exists an arrow $I \xrightarrow{j} \mathbf{NatFI}$, such that the below diagram commutes:

$$\begin{array}{ccc}
 & \mathbf{NatF}(i) & \\
 \mathbf{NatF}(\mathbf{NatFI}) & \xrightarrow{\quad} & \mathbf{NatFI} \\
 \mathbf{NatF}(j) \uparrow & & \uparrow j \\
 \mathbf{NatFI} & \xrightarrow{i} & I
 \end{array}$$

Observe the two arrows along the path of the recursive functor \mathbf{NatF}

$$\mathbf{NatF}(\mathbf{NatFI}) \xrightarrow{\mathbf{NatF}(i)} \mathbf{NatFI} \xrightarrow{i} I$$

To make it obvious, we draw the second section in top-down direction:

$$\begin{array}{ccc}
 & \mathbf{NatF}(i) & \\
 \mathbf{NatF}(\mathbf{NatFI}) & \xrightarrow{\quad} & \mathbf{NatFI} \\
 & & \downarrow i \\
 & & I
 \end{array}$$

Compare with the above diagram, we find the directions of i and j are reversed. Composite the two together, we obtain an endo-arrow from I to itself. As the only endo-arrow of the initial object is id , therefore:

$$i \circ j = id$$

We can use the similar analysis for \mathbf{NatFI} . Since the arrow from the initial object to the recursive object is unique, the arrow i from \mathbf{NatFI} to I , connected with the arrow j back to \mathbf{NatFI} must also be id

$$j \circ i = id_{\mathbf{NatFI}}$$

It means $\mathbf{NatF}I$ is isomorphic to I .

$$\mathbf{NatF}I = I$$

This is exactly the concept of fixed point introduced above. From the previous section, we know that the fixed point of \mathbf{NatF} is \mathbf{Nat} , hence

$$\mathbf{NatF} \mathbf{Nat} = \mathbf{Nat}$$

This result is as same as the arithmetic axioms given by Peano in 1889. For any algebraic structure $(A, [c, f])$ that satisfies Peano axioms, there exists a unique isomorphism from natural numbers $(N, [\text{zero}, \text{succ}])$, which sends number n to $f^n(c) = f(f(\dots f(c))\dots)$. We can give the definition of *initial algebra* now:

Definition 4.5.5. If there exists the initial object in the category of F-algebra $\mathbf{Alg}(F)$, then the initial object is called the initial algebra.

In the category of set furnished with total functions, many functors, including polymorphic functors have the initial algebra. We skipped the proof for existence of initial algebra, readers can refer to [48]. Given a functor \mathbf{F} , we can obtain the initial object of the F-algebra through its fixed point.



Joachim Lambek, 1922 - 2014

Lambek, in 1968, pointed out i is an isomorphism, and named the initial algebra (I, i) the fixed point of functor \mathbf{F} . This result is called *Lambek theorem* nowadays[49].

If there exists the initial algebra (I, i) in F-algebra, then there is unique morphism to any other algebra (A, f) . Denote the morphism from I to A as (f) , which makes the below diagram commute:

$$\begin{array}{ccc} \mathbf{F}I & \xrightarrow{i} & I \\ \mathbf{F}(\mathbf{f}) \downarrow & & \downarrow (\mathbf{f}) \\ \mathbf{F}A & \xrightarrow{f} & A \end{array}$$

$$h = (\mathbf{f}), \text{ if and only if } h \circ i = f \circ \mathbf{F}(h)$$

We call the arrow (f) the *catamorphism*. It comes from Greek word $\kappa\alpha\kappa\alpha$, means downward. The brackets ' $\langle \rangle$ ' look like a pair of bananas, therefore, people call them 'banana brackets'.

Catamorphism is powerful, it can convert the function f on a non-recursive structure, to (f) for recursive structure. Hence build the complex recursive computation. Let us see how it works with the example of natural numbers.

The natural number functor \mathbf{NatF} is not recursive, while the initial algebra for natural number \mathbf{Nat} is recursive:

```
data NatF A = ZeroF | SuccF A

data Nat = Zero | Succ Nat
```

Although the arrow $\mathbf{NatF}A \xrightarrow{f} A$ is not recursive, the catamorphism *cata* builds the arrow $\mathbf{Nat} \rightarrow A$ from f , which can apply computation to the recursive **Nat**. The type of *cata* is as below:

$$(\mathbf{NatF}A \xrightarrow{f} A) \xrightarrow{\text{cata}} (\mathbf{Nat} \rightarrow A)$$

The Curried *cata*(f) should be able to apply to the both values of **Nat**, *Zero* and *Succ n*. We can define *cata* from this fact:

$$\begin{aligned} \text{cata } f \text{ Zero} &= f \text{ ZeroF} \\ \text{cata } f \text{ (Succ } n) &= f(\text{SuccF} \text{ (cata } f \text{ } n)) \end{aligned}$$

The first clause handles the edge condition for the recursion. For the *Zero* value of **Nat**, it applies f to it; for the recursive case, which is *Succ n*, it first recursively evaluates *cata f n* to get a value a of type A ; then uses *SuccF a* to convert it to a value of type **NatF** A ; finally applies f on top of it. This natural number catamorphism is generic as it is applicable to any carrier object A . Let's further see two concrete examples. The first one converts any value of **Nat** back to *Int*.

```
toInt :: Nat → Int
toInt = cata eval where
  eval :: NatF Int → Int
  eval ZeroF = 0
  eval (SuccF x) = x + 1
```

With this definition, *toInt* *Zero* gives 0, and *toInt* (*Succ* (*Succ* (*Succ* *Zero*))) gives 3. We can define a helper function to make the verification easy:

```
fromInt :: Int → Nat
fromInt 0 = Zero
fromInt n = Succ (fromInt (n-1))
```

For any integer n , $n = (\text{toInt} \circ \text{fromInt}) n$ holds. This example looks trivial. Let us see the second example about Fibonacci numbers.

```
toFib :: Nat → (Integer, Integer)
toFib = cata fibAlg where
  fibAlg :: NatF (Integer, Integer) → (Integer, Integer)
  fibAlg ZeroF = (1, 1)
  fibAlg (SuccF (m, n)) = (n, m + n)
```

We intentionally rename the previous function from *fib* to *fibAlg* to call out our purpose. From the algebraic relation of Fibonacci numbers (non-recursive), we make it capable to recursively calculate the Fibonacci sequence with catamorphism. *toFib* *Zero* gives pair (1, 1), and *toFib* (*Succ* (*Succ* (*Succ* *Zero*))) gives pair (3, 5). The following helper function calculates the n -th Fibonacci number.

$$\text{fibAt} = \text{fst} \circ \text{toFib} \circ \text{fromInt}$$

In fact, for any natural number like algebraic structure $(A, c + f)$ that satisfies Peano axioms, we can use the catamorphism and the initial algebra $(\mathbf{Nat}, \text{zero} + \text{succ})$ to build the recursive computation tool $(c + f)$ ³². Let's prove this fact. Observe the below diagram.

³²We simplified to symbol $(c + f)$ because there are too many levels of brackets in $([c, f])$.

$$\begin{array}{ccc}
 \mathbf{NatF} \mathbf{Nat} & \xrightarrow{[zero, succ]} & \mathbf{Nat} \\
 \mathbf{NatF}(h) \downarrow & & \downarrow h \\
 \mathbf{NatF} A & \xrightarrow{[c, f]} & A
 \end{array}$$

The catamorphism make this diagram commute.

$$\begin{aligned}
 & h \circ [zero, succ] = [c, f] \circ \mathbf{NatF}(h) && \text{Commute} \\
 \Rightarrow & h \circ [zero, succ] = [c, f] \circ (id + h) && \text{Polynomial functor} \\
 \Rightarrow & h \circ [zero, succ] = [c \circ id, f \circ h] && \text{Absorption law for coproduct on the right} \\
 \Rightarrow & [h \circ zero, h \circ succ] = [c, f \circ h] && \text{Fusion law of coproduct on the left} \\
 \Rightarrow & \begin{cases} h \circ zero = c \\ h \circ succ = f \circ h \end{cases} \\
 \Rightarrow & \begin{cases} h \circ zero = c \\ h \circ (Succ n) = f(h(n)) \end{cases} \\
 \Rightarrow & \begin{cases} h(0) = c \\ h(n + 1) = f(h(n)) \end{cases}
 \end{aligned}$$

This is exactly the folding definition of natural numbers:

$$h = foldn(c, f)$$

It tells us, the catamorphism of natural numbers $\langle c + f \rangle = foldn(c, f)$. For Fibonacci numbers, we can use

$$\langle fibAlg \rangle = \langle start + next \rangle = foldn(start, next)$$

to calculate. It seems we go back to the first chapter after a long journey. It's actually a spiral of understanding. In chapter 1, we obtained this result from induction and abstraction; Here we reach to a higher level, by applying the abstract pattern to the concrete problem, we obtain the same result.

Algebraic data type

We can define more algebraic data types with initial F-algebra, like list and binary tree.

Example 4.5.9. The definition of list given in chapter 1 is as below.

data **List** A = **Nil** | **Cons** A (**List** A)

The corresponding non-recursive functor is:

data **ListF** A B = **NilF** | **ConsF** A B

Actually, **List** is the fixed point of functor **ListF**. To verify it, let $B' = \mathbf{ListF} A B$, then recursively apply to itself infinite many times. Denote the result of it as **Fix** (**ListF** A).

$$\begin{aligned}
 \mathbf{Fix} (\mathbf{ListF} A) &= \mathbf{ListF} A (\mathbf{Fix} (\mathbf{ListF} A)) && \text{Definition of fixed point} \\
 &= \mathbf{ListF} A (\mathbf{ListF} A (...)) && \text{Expand} \\
 &= \mathbf{NilF} | \mathbf{ConsF} A (\mathbf{ListF} A (...)) && \text{Definition of } \mathbf{ListF} A \\
 &= \mathbf{NilF} | \mathbf{ConsF} A (\mathbf{Fix} (\mathbf{ListF} A)) && \text{Reverse of fixed point}
 \end{aligned}$$

Compare with the definition of **List A**, we have:

$$\mathbf{List} A = \mathbf{Fix} (\mathbf{ListF} A)$$

Fixing A in functor **ListF**, for any carrier object B , define arrow

$$\mathbf{ListF} A B \xrightarrow{f} B$$

It forms F-algebra for list. We know that the initial algebra is $(List, [nil, cons])$, hence the catamorphism is:

$$\begin{array}{ccc} \mathbf{ListF} A (\mathbf{List} A) & \xrightarrow{[nil, cons]} & (\mathbf{List} A) \\ (\mathbf{ListF} A)(h) \downarrow & & \downarrow h \\ \mathbf{ListF} A B & \xrightarrow{[c, f]} & B \end{array}$$

Given a non-recursive computation f , we can build from it through the catamorphism that can be applied to recursive list.

```
cata :: (ListF a b → b) → (List a → b)
cata f Nil = f NilF
cata f (Cons x xs) = f (ConsF x (cata f xs))
```

For example, we define the algebraic rules to calculate the length of list:

```
len :: (List a) → Int
len = cata lenAlg where
  lenAlg :: ListF a Int → Int
  lenAlg NilF = 0
  lenAlg (ConsF _ n) = n + 1
```

Hence len *Zero* gives 0, and $len (Cons 1 (Cons 1 Zero))$ gives 2. We can define a helper function to convert the list in bracket symbol to *List*.

```
fromList :: [a] → List a
fromList [] = Nil
fromList (x:xs) = Cons x (fromList xs)
```

We can chain them together, for example as $len(fromList[1, 1, 2, 3, 5, 8])$ to calculate the length of a list.

Different algebraic rule f gives different computation on list. Below example sums up the elements in a list:

```
sum :: (Num a) ⇒ (List a) → a
sum = cata sumAlg where
  sumAlg :: (Num a) ⇒ ListF a a → a
  sumAlg NilF = 0
  sumAlg (ConsF x y) = x + y
```

Next we use list diagram to prove, the catamorphism on list is the folding computation *foldr* essentially. List functor is a polynomial functor. When fix A , for carrier object B , it sends to $\mathbf{ListF} A B = 1 + A \times B$; for arrow h , it sends to $(\mathbf{ListF} A)(h) = id + (id \times h)$. As the diagram commutes, we have:

$$\begin{aligned}
& h \circ [nil, cons] = [c, f] \circ (\mathbf{ListF} A)(h) && \text{Commute} \\
\Rightarrow & h \circ [nil, cons] = [c, f] \circ (id + (id \times h)) && \text{Polynomial functor} \\
\Rightarrow & h \circ [nil, cons] = [c \circ id, f \circ (id \times h)] && \text{Absorption law of coproduct on the right} \\
\Rightarrow & [h \circ nil, h \circ cons] = [c, f \circ (id \times h)] && \text{Fusion law of coproduct on the left} \\
\Rightarrow & \begin{cases} h \circ nil = c \\ h \circ cons = f \circ (id \times h) \end{cases} \\
\Rightarrow & \begin{cases} h Nil = c \\ h (Cons a x) = f(a, h(x)) \end{cases}
\end{aligned}$$

This is exactly the definition of list folding:

$$h = foldr(c, f)$$

It tells us, the catamorphism of list F-algebra $(c + f) = foldr(c, f)$. Hence $foldr(0, (a, b) \mapsto b + 1)$ computes the length of a list, and $foldr(0, +)$ sums up the elements of a list.

Example 4.5.10. In chapter 2, we defined binary tree as below:

```
data Tree A = Nil | Br A (Tree A) (Tree A)
```

All the binary tree like structure can be described with F-algebra. First, we need define the binary tree functor:

```
data TreeF A B = NilF | BrF A B B
```

B is the carrier object, the initial algebra is $(\mathbf{Tree} A, [nil, branch])$. We leave the proof of this fact as exercise of this section. Below diagram illustrates the catamorphism of binary tree:

$$\begin{array}{ccc}
\mathbf{TreeF} A (\mathbf{Tree} A) & \xrightarrow{[nil, branch]} & (\mathbf{Tree} A) \\
(\mathbf{TreeF} A)(h) \downarrow & & \downarrow h \\
\mathbf{TreeF} A B & \xrightarrow{[c, f]} & B
\end{array}$$

The catamorphism of binary tree accepts an arrow of F-algebra $\mathbf{TreeF} A B \xrightarrow{f} B$, and returns a function of $\mathbf{Tree} A \rightarrow B$:

$$\begin{aligned}
cata : (\mathbf{TreeF} A B \rightarrow B) & \rightarrow (\mathbf{Tree} A \rightarrow B) \\
cata f Nil & = f NilF \\
cata f (Br k l r) & = f (BrF k (cata f l) (cata f r))
\end{aligned}$$

Below example defines the algebra to sum the elements in the binary tree, then it applies the catamorphism to recursively sum along any binary tree:

$$\begin{aligned}
sum : \mathbf{Tree} A \rightarrow A \\
sum = cata \circ sumAlg \\
\text{其中: } & \begin{cases} sumAlg : \mathbf{TreeF} A B \rightarrow B \\ sumAlg NilF = 0_B \\ sumAlg (Br k l r) = k + l + r \end{cases}
\end{aligned}$$

Next, we prove the catamorphism is the essentially the folding operation $foldt$ for binary tree. Fix object A , the binary tree functor $\mathbf{TreeF} A B$ is a polynomial functor.

For a given carrier object B , it sends to $\mathbf{TreeF} A B = 1 + (A \times B \times B)$; for arrow h , it sends to $(\mathbf{TreeF} A)(h) = id + (id_A \times h \times h)$.

Since the diagram commutes, we have:

$$\begin{aligned}
 & h \circ [nil, branch] = [c, f] \circ (\mathbf{TreeF} A)(h) && \text{Commute} \\
 \Rightarrow & h \circ [nil, branch] = [c, f] \circ (id + (id_A \times h \times h)) && \text{Polynomial functor} \\
 \Rightarrow & h \circ [nil, branch] = [c \circ id, f \circ (id_A \times h \times h)] && \text{Absorption law of coproduct on the right} \\
 \Rightarrow & [h \circ nil, h \circ branch] = [c, f \circ (id_A \times h \times h)] && \text{Fusion law of coproduct on the left} \\
 \Rightarrow & \begin{cases} h \circ nil = c \\ h \circ branch = f \circ (id_A \times h \times h) \end{cases} \\
 \Rightarrow & \begin{cases} h Nil = c \\ h (Br k l r) = f(k, h(l), h(r)) \end{cases}
 \end{aligned}$$

It is exactly the definition of folding on binary tree $h = foldt(c, f)$, where:

$$\begin{cases} foldt c h Nil = c \\ foldt c h (Br k l r) = h(k, foldt c h l, foldt c h r) \end{cases}$$

Therefore, the computation to sum all the elements in a binary tree can be represented by folding as $foldt(0_B, (\bullet + \bullet + \bullet))$.

Exercise 4.9

1. For the binary tree functor $\mathbf{TreeF} A B$, fix A , use the fixed point to prove that $(\mathbf{Tree} A, [nil, branch])$ is the initial algebra

4.6 Summary

In this chapter, we introduced the basic concepts in category theory, including category, functor, natural transformation, product and coproduct, initial object and final object, exponentials, and F-algebra. They can construct complex algebraic structures. Before the end of this chapter, let us see how the generic folding operation is realized in the language of category theory[50]:

```
foldr f z t = appEndo (foldMap (Endo o f) t) z
```

The reason why it is defined like this is abstraction, to make $foldr$ not only limit to list, but also expand to any foldable structures. It's common to define the folding operation for list as below:

```
foldr :: (a → b → b) → b → [a] → b
foldr f z [] = z
foldr f z (x:xs) = f x (foldr f z xs)
```

Where f is a binary operation, we can denote it as \oplus , and explicitly denote z as the identity element e . According to this definition, $foldr (\oplus) e [a, b, c]$ can be expanded as:

$$a \oplus (b \oplus (c \oplus e))$$

It reminds us monoid, $foldr$ essentially repeats the binary operation on top of monoid. In the monoid definition, besides the identity element and binary operation, we can add an additional operation to 'sum up' a list of elements with the \oplus and e symbols:

$$\begin{aligned} concat_M &: [M] \rightarrow M \\ concat_M &= foldr (\oplus) e \end{aligned}$$

It is named as **mconcat** in some programming environment. For any monoid M , function *concat* processes a list of elements of M , fold them together through the binary operation and the identity element. For example, string is an instance of monoid, the identity element is the empty string, and the binary operation is string concatenation. Hence $concat_M ["Hello", "String" "Monoid"]$ gives the following result.

`"Hello" ++ ("String" ++ ("Monoid" ++ "")) = "HelloStringMonoid"`

We are able to sum up any monoid elements. Can we make it more generic? Consider there is a list of elements that don't belong to monoid. We can still sum them if there is a way to convert them to monoid elements. List functor behaves exactly as we expect, to convert a list of some types to a list of monoids. In other words, we can use the list functor to 'lift' the arrow $A \xrightarrow{g} M$ to **List**(g) for sum.

$$\begin{array}{ccccc} [A] & \xrightarrow{fmap\ g} & [M] & \xrightarrow{concat} & M \\ \text{List} \uparrow & & & \uparrow \text{List} & \\ A & \xrightarrow{g} & M & & \end{array}$$

$$\begin{aligned} foldMap &: (A \rightarrow M) \rightarrow [A] \rightarrow M \\ foldMap\ g &= concat_M \circ fmap\ g \end{aligned}$$

However, there is still limitation. For the binary operation $f : A \rightarrow B \rightarrow B$ passed to *foldr*, if B is not monoid, we can't do folding. To solve it, consider the Curried f of $f : A \rightarrow (B \rightarrow B)$. When we treat the arrow $B \rightarrow B$ as object, they can form a monoid. Where the identity element is the *id* arrow, and the binary operation is arrow composition. To make it clear, we wrap the $B \rightarrow B$ arrow as a type (set) through a functor:

newtype Endo B = Endo (B → B)

We name it as 'endo' functor as it point to B from B itself. Besides, we also define a function of **Endo** $B \xrightarrow{appEndo} B$:

$$appEndo(Endo\ a) = a$$

Now we can declare **Endo** is a monoid, with *id* arrow as the identity element, and function composition as the binary operation.

```
instance Monoid Endo where
  mempty = Endo id
  Endo f `mappend` Endo g = Endo (f ∘ g)
```

Given any binary function f , we can use *foldMap* to fold on *Endo* monoid:

$$\begin{aligned} foldCompose &: (A \rightarrow (B \rightarrow B)) \rightarrow [A] \rightarrow \mathbf{Endo}B \\ foldCompose\ f &= foldMap (Endo ∘ f) \end{aligned}$$

When compute $foldCompose\ f [a, b, c]$, it will be expanded as the following:

$$\begin{aligned} & Endo(f\ a) \oplus (Endo(f\ b) \oplus (Endo(f\ c) \oplus Endo(id))) \\ &= Endo(f\ a \oplus (f\ b \oplus (f\ c \oplus id))) \end{aligned}$$

Here is a more concrete example

$$\begin{aligned}
 & \text{foldCompose } (+) [1, 2, 3] \\
 \Rightarrow & \text{foldMap } (\text{Endo} \circ (+)) [1, 2, 3] \\
 \Rightarrow & \text{concat}_M (\text{fmap } (\text{Endo} \circ (+))) [1, 2, 3] \\
 \Rightarrow & \text{concat}_M (\text{fmap Endo } [(+1), (+2), (+3)]) \\
 \Rightarrow & \text{concat}_M [\text{Endo } (+1), \text{Endo } (+2), \text{Endo } (+3)] \\
 \Rightarrow & \text{Endo } ((+1) \circ (+2) \circ (+3)) \\
 \Rightarrow & \text{Endo } (+6)
 \end{aligned}$$

As the last step, we need extract the result from the *Endo* object. For this example, we use *appEndo* to extract $(+6)$, then apply it to the initial value z passed to *foldr*:

$$\begin{aligned}
 \text{foldr } f \ z \ xs &= \text{appEndo } (\text{foldCompose } f \ xs) \ z \\
 &= \text{appEndo } (\text{foldMap } (\text{Endo} \circ f) \ xs) \ z
 \end{aligned}$$

This is the complete definition of *foldr* in the language of categories. We can further define a dedicated type *Foldable*, such that for any data structure, user can either realize *foldMap* or *foldr*. See the appendix of this chapter for detail.

4.7 Further reading

We can not introduce the category theory in depth just in one chapter. We only scratch the surface of the iceberg. The key idea is abstraction in category theory. The dual concepts like initial and final objects, product and coproduct, can be further abstracted to the higher level dual things, limits and colimits. We did not introduce the adjunctions, nor Yoneda lemma. We did not explain about the increasingly popular monad concept in programming. We hope this chapter could help the reader to find the door to the category world. Here are some good materials for further reading: Mac Lane, who developed category theory, wrote a classic textbook[51] targeted to working mathematicians. It's a bit hard for the new comers. *An introduction to category theory* by Simons[42] and *Category Theory - A Gentle Introduction* by Smith are more suitable at the begining. For readers with programming background, *Category Theory for Programmers* by Milewski[45] is a good book with many example programs in Haskell, and their corresponding implementations in C++. However, this book limits to the set category on total functions (the Hask category to be accurate). *Algebra of programming* by Bird[6] introduced category framework for programming in depth.

4.8 Appendix - example programs

Definition of functors:

```

class Functor f where
  fmap      :: (a → b) → f a → f b
  
```

The Maybe functor:

```

instance Functor Maybe where
  fmap _ Nothing   = Nothing
  fmap f (Just a) = Just (f a)
  
```

Look up in binary search tree with Maybe functor, and convert the result to binary format:

```

lookup Nil _ = Nothing
lookup (Node l k r) x | x < k = lookup l x
                        | x > k = lookup r x
                        | otherwise = Just k

lookupBin = (fmap binary)  $\circ$  lookup

```

Definition of bifunctor:

```

class Bifunctor f where
  bimap :: (a  $\rightarrow$  c)  $\rightarrow$  (b  $\rightarrow$  d)  $\rightarrow$  f a b  $\rightarrow$  f c d

```

Definition of product and coproduct functors:

```

instance Bifunctor (,) where
  bimap f g (x, y) = (f x, g y)

instance Bifunctor Either where
  bimap f _ (Left a) = Left (f a)
  bimap _ g (Right b) = Right (g b)

```

Definition of *curry* and its reverse *uncurry*:

```

curry      :: ((a, b)  $\rightarrow$  c)  $\rightarrow$  a  $\rightarrow$  b  $\rightarrow$  c
curry f x y = f (x, y)

uncurry    :: (a  $\rightarrow$  b  $\rightarrow$  c)  $\rightarrow$  ((a, b)  $\rightarrow$  c)
uncurry f (x, y) = f x y

```

Definition of monoid:

```

class Semigroup a  $\Rightarrow$  Monoid a where
  mempty :: a

  mappend :: a  $\rightarrow$  a  $\rightarrow$  a
  mappend = ( $\diamond$ )

  mconcat :: [a]  $\rightarrow$  a
  mconcat = foldr mappend mempty

```

Definition of foldable:

```

newtype Endo a = Endo { appEndo :: a  $\rightarrow$  a }

class Foldable t where
  foldr :: (a  $\rightarrow$  b  $\rightarrow$  b)  $\rightarrow$  b  $\rightarrow$  t a  $\rightarrow$  b
  foldr f z t = appEndo (foldMap (Endo  $\circ$  f) t) z

  foldMap :: Monoid m  $\Rightarrow$  (a  $\rightarrow$  m)  $\rightarrow$  t a  $\rightarrow$  m
  foldMap f = foldr (mappend  $\circ$  f) mempty

```


Chapter 5

Fusion

... mathematical knowledge ... is, in fact, merely verbal knowledge. “3” means “2+1”, and “4” means “3+1”. Hence it follows (though the proof is long) that “4” means the same as “2+2”. Thus mathematical knowledge ceases to be mysterious.

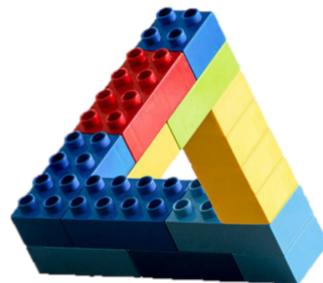
– Bertrand Russell

I still remember the mathematics class in high school. My teacher often wrote down a complex formula with many alphabetic symbols in the blackboard, then asked us to simplify it. Some student stood up, volunteered to do it in front of the class. Combining like terms, factorization, ... all means were attempted. It liked a magic process often led to unbelievable simple result. Of course sometimes the guy was stuck or trapped in loops, and finally saved by our teacher.

Chalk and blackboard, Such experience was unforgettable, just like happened yesterday. I was so impressed to the mythical power of reasoning. I always wanted to know more formulas, that could help me to deduce the result.

The magic is that, we even needn't care about the concrete meanings when doing the deduction. It likes building bricks, from different parts, we finally build an interesting toy. These formulas and theorems can also be combined together, and finally build an interesting result. When meet $a^2 + 2ab + b^2$, then turn it into $(a + b)^2$, just like mating two bricks. We needn't force ourselves to remind the geometric meanings for this formula when deducing it.

We use two examples in this chapter to demonstrate how to do deduction in programming. For every example, we'll both explain the intuitive concrete meanings, and give the purely formal deduction process. Just like the $(a + b)^2$ case, on one hand, we can explain it as the total area from two different squares and two equal rectangles; on the other hand, we can also deduce the same result step by step.



Penrose triangle

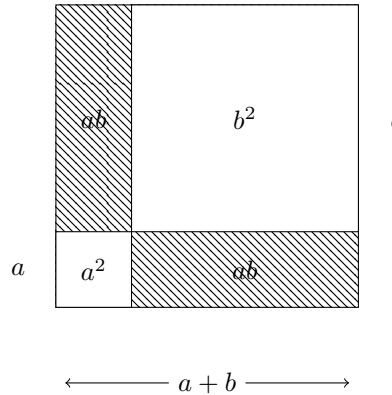


Figure 5.2: Geometric illustration for $(a + b)^2 = a^2 + 2ab + b^2$

$$\begin{aligned}
 (a + b)^2 &= (a + b)(a + b) && \text{Definition of square} \\
 &= a(a + b) + b(a + b) && \text{Distribution law for multiplication} \\
 &= a^2 + ab + ba + b^2 && \text{Distribution law again} \\
 &= a^2 + 2ab + b^2 && \text{Combined } ab \text{ and } ba
 \end{aligned}$$

5.1 foldr/build fusion

The first example is the foldr/build fusion law. In 2015, a main stream programming language Java adopted lambda expression and a set of functional tools in its 1.8 version. However, some programmer soon found the chained function calls brought elegance and expressiveness at the penalty of performance if using carelessly. One reason is the chained functions may generate excessive intermediate results. While these intermediate results are not necessary simple values, but the complete list, container, or collection of complex structures. They are thrown away after consumed by the next functions. However, the same level of new result will be generated step by step. Such pattern that produce, one time consume, thrown away, then produce again, happens along the function chain repeatedly, which causes computation overhead.

For example, we can define below function to examine if every element in a list satisfies a given prediction[52].

$$all(p, xs) = and(map(p, xs))$$

Such that $all(prime, [2, 3, 5, 7, 11, 13, 17, 19, \dots])$ tells if all numbers in the list are primes. However, the performance of this realization is poor. Firstly, $map(prime, xs)$ generates a list of the same length as xs , every element is a Boolean value [True, True, ...], indicating the corresponding number is prime or not. Then the list of Boolean values is passed to and function, examine if there exists False value. Finally, both xs and the Boolean list are thrown away, only one Boolean value is returned as the result.

Below is an alternative definition. It can avoid generating the intermediate Boolean list.

$$\begin{aligned}
 all(p, xs) &= h(xs) \\
 \begin{cases} h([]) = \text{True} \\ h(x : xs) = p(x) \wedge h(xs) \end{cases}
 \end{aligned}$$

Although this realization does not generate intermediate result, it is neither intuitive

nor elegant compare to $and(map(p, xs))$. Are there any way that has the advantage of both methods? We found some transformations satisfy such requirement, for example:

$$map\ sqrt\ (map\ abs\ xs) = map\ (sqrt \circ abs)\ xs$$

First generate a list of absolute values, then evaluate square root for every number. It is equivalent to take absolute value first, then evaluate the square root for every number in that list. Hence we have the following rule:

$$map\ f\ (map\ g\ xs) = map\ (f \circ g)\ xs \quad (5.1)$$

However, there are too many rules. We can't list them all. It's not practical to apply them on a complex program. Gill, Launchbury, and Peyton Jones developed a method in 1993, starting from the basic build and folding operation, they found the pattern to optimize the chained function.

5.1.1 Folding from right for list

We defined the fold from right operation for list in chapter 1 as below:

$$\begin{aligned} foldr\ \oplus\ z\ [] &= z \\ foldr\ \oplus\ z\ (x : xs) &= x \oplus (foldr\ \oplus\ z\ xs) \end{aligned}$$

It can be expanded as:

$$foldr\ \oplus\ z\ [x_1, x_2, \dots, x_n] = x_1 \oplus (x_2 \oplus (\dots (x_n \oplus z) \dots)) \quad (5.2)$$

Many list operations can be realized by folding, for example:

1. Sum:

$$sum = foldr\ +\ 0$$

2. The *and* function, that applies logic and for all Boolean values in a list:

$$and = foldr\ \wedge\ True$$

This is because:

$$and\ [x_1, x_2, \dots, x_n] = x_1 \wedge (x_2 \wedge (\dots (x_n \wedge True) \dots))$$

3. Test if a given element belongs to a list:

$$elem\ x\ xs = foldr\ (a\ b \mapsto (a = x) \vee b)\ False\ xs$$

4. Map:

$$\begin{aligned} map\ f\ xs &= foldr\ (x\ ys \mapsto f(x) : ys)\ []\ xs \\ &= foldr\ ((:) \circ f)\ []\ xs \end{aligned}$$

5. Filter the elements with a given prediction:

$$filter\ f\ xs = foldr\ (x\ ys \mapsto \begin{cases} f(x) : & x : ys \\ \text{otherwise} : & ys \end{cases})\ []\ xs$$

6. Concatenate two lists:

$$xs \uplus ys = foldr (:) ys xs \quad (5.3)$$

This is because:

$$[x_1, x_2, \dots, x_n] \uplus ys = x_1 : (x_2 : (\dots (x_n : ys)) \dots)$$

7. Concatenate multiple lists:

$$concat\ xss = foldr \uplus []\ xss$$

Actually all list operations can be realized by folding (We proved it in the F-algebra section in previous chapter), if we can simplify folding, we then can simplify all list operations.

5.1.2 foldr/build fusion law

Let's consider, what if fold from right by the cons operation `(:)` starting from an empty list `[]`?

$$foldr (:) [] [x_1, x_2, \dots, x_n] = x_1 : (x_2 : (\dots (x_n : [])) \dots) \quad (5.4)$$

We end up with the list itself. You may remind the fixed point introduced in previous chapter, we'll return to this topic later. In other words, if we have an operation g , from a starting value, for example `[]`, and a binary operation, from example `:`, it generates a list. We define such list construction process as *build*:

$$build(g) = g((:), []) \quad (5.5)$$

Next, if we fold the list with another start value z and binary operation f , the result is equivalent to call g by replacing `[]` with z , and replacing `((:))` with f .

$$foldr(f, z, build(g)) = g(f, z) \quad (5.6)$$

Written in pointless format (without parentheses and named arguments) is:

$$foldr\ f\ z\ (build\ g) = g\ f\ z \quad (5.7)$$

We named this formula *foldr/build fusion law*.

Let us start from an example. Consider how to sum up all the integers from a to b , which is $sum([a, a + 1, \dots, b - 1, b])$. First, we need generate all the integers from a to b . Below definition enumerates $a, a + 1, a + 2, \dots, b - 1, b$.

$$range(a, b) = \begin{cases} a > b : & [] \\ \text{otherwise} : & a : range(a + 1, b) \end{cases}$$

Such that $range(1, 5)$ builds the list $[1, 2, 3, 4, 5]$. Sum up the enumerated list gives the answer.

$$sum(range(a, b))$$

Next, we extract the start value `[]` and the binary operation `(:)` out as parameters:

$$\text{range}'(a, b, \oplus, z) = \begin{cases} a > b : & z \\ \text{otherwise :} & a \oplus \text{range}'(a + 1, b, \oplus, z) \end{cases}$$

We can further Curry the last two arguments of range' .

$$\text{range}' a b = f c \mapsto \begin{cases} a > b : & c \\ \text{otherwise :} & f a (\text{range}'(a + 1) b f c) \end{cases}$$

Now we can redefine range with range' and build :

$$\text{range}(a, b) = \text{build}(\text{range}'(a, b))$$

Next, we simplify the sum with the fusion law.

$$\begin{aligned} \text{sum}(\text{range}(a, b)) &= \text{sum}(\text{build}(\text{range}'(a, b))) && \text{substitute} \\ &= \text{foldr } (+) 0 (\text{build}(\text{range}' a b)) && \text{define sum with foldr} \\ &= \text{range}' a b (+) 0 && \text{fusion law} \end{aligned}$$

It gives the simplified result, which avoid generating the intermediate list, hence optimize the algorithm. Let's see how this result expands:

$$\text{range}' a b (+) 0 = \begin{cases} a > b : & 0 \\ \text{otherwise :} & a + \text{range}'(a + 1, b, (+), 0) \end{cases}$$

5.1.3 build forms for list

To leverage fusion law conveniently, we can rewrite the common functions that generate list in the form of $\text{build} \dots \text{foldr}$. Such that when composite with folding, we can simplify $\text{foldr} \dots (\text{build} \dots \text{foldr})$ with fusion law.

1. The simplest one generates an empty list.

$$[] = \text{build} (f z \mapsto z)$$

We can substitute it with the definition (5.5) of build to prove this result.

Proof.

$$\begin{aligned} \text{build} (f z \mapsto z) &= (f z \mapsto z) (:) [] && \text{definition of build} \\ &= (:) [] \mapsto [] && \beta - \text{reduction, see chapter 2} \\ &= [] \end{aligned}$$

□

2. The next one is cons operation, which links an element to a list.

$$x : xs = \text{build} (f z \mapsto f x (\text{foldr } f z xs))$$

Let us verify it.

Proof.

$$\begin{aligned} &\text{build} (f z \mapsto f x (\text{foldr } f z xs)) \\ &= (f z \mapsto f x (\text{foldr } f z xs)) (:) [] && \text{definition of build} \\ &= x : (\text{foldr} (:) [] xs) && \beta - \text{reduction} \\ &= x : xs && \text{By (5.4), the fixed point of folding} \end{aligned}$$

□

3. List concatenation:

$$xs \uplus ys = \text{build } (f z \mapsto \text{foldr } f (\text{foldr } f z ys) xs)$$

Proof.

$$\begin{aligned} & \text{build } (f z \mapsto \text{foldr } f (\text{foldr } f z ys) xs) \\ &= (f z \mapsto \text{foldr } f (\text{foldr } f z ys) xs) (:) [] \quad \text{Definition of build} \\ &= \text{foldr } (:) (\text{foldr } (:) [] ys) xs \quad \beta - \text{reduction} \\ &= \text{foldr } (:) ys xs \quad \text{Fixed point for inner part} \\ &= xs \uplus ys \quad \text{By (5.3), list concatenation} \end{aligned}$$

□

For the rest examples, we only give the final result and leave the proof as exercises.

4. Concatenate multiple lists.

$$\text{concat } xss = \text{build } (f z \mapsto \text{foldr } (xs x \mapsto \text{foldr } f xxs) xss)$$

5. Map.

$$\text{map } f xs = \text{build } (\oplus z \mapsto \text{foldr } (y ys \mapsto (f y) \oplus ys) z xs)$$

6. Filter.

$$\text{filter } f xs = \text{build } (\oplus z \mapsto \text{foldr } (x xs' \mapsto \begin{cases} f(x) : & x \oplus xs' \\ \text{otherwise} : & xs' \end{cases}) z xs)$$

7. Generate an infinite long list of the same element.

$$\text{repeat } x = \text{build } (\oplus z \mapsto \text{let } r = x \oplus r \text{ in } r)$$

5.1.4 Reduction with the fusion law

Empowered by the fusion law, we are ready to simplify varies of computation. The first example is $\text{all}(p, xs) = \text{and}(\text{map}(p, xs))$, which was mentioned at the begining of this chapter.

Example 5.1.1. First, we express *and* in folding form, then turn *map* into its build form, and apply fusion law to do the reduction.

$$\begin{aligned} \text{all}(p, xs) &= \text{and}(\text{map}(p, xs)) && \text{definition} \\ &= \text{foldr } \wedge \text{True } \text{map}(p, xs) && \text{folding form of and} \\ &= \text{foldr } \wedge \text{True } \text{build } (\oplus z \mapsto \\ &\quad \text{foldr } (x ys \mapsto p(x) \oplus ys) z xs) && \text{build form of map} \\ &= (\oplus z \mapsto \text{foldr } (x ys \mapsto p(x) \oplus ys) z xs) \wedge \text{True} && \text{fusion law} \\ &= \text{foldr } (x ys \mapsto p(x) \wedge ys) \text{True } xs && \beta - \text{reduction} \end{aligned}$$

We can define a helper function *first*, which applies a given function *f* to the first element of a pair.

$$(\text{first } f) x y = f(x) y$$

Then we can further simplify *all* to:

$$\text{all } p = \text{foldr } (\wedge) \circ (\text{first } p) \text{True}$$

Example 5.1.2. The second example is to concatenate multiple words together and interpolate them with spaces, such that they form a sentence. This text manipulation process is often called *join*. For illustration purpose, we add an additional space at the end of the sentence. A typical definition is as below:

$$\text{join}(ws) = \text{concat}(\text{map}(w \mapsto w + [''], ws))$$

This definition is straightforward. It first uses *map* to append a space to every word, and output a new list of words; then concatenates the list to a string. However, its performance is poor. Appending at the end of a word is an expensive operation. It need move from the head to the tail of every word first, then do the string concatenation. How many words there are, how long the new list will be. This intermediate words list is thrown away finally. Let's optimize it with the fusion law.

$$\begin{aligned}
 & \text{join}(ws) \\
 & \quad \{ \text{definition} \} \\
 & = \text{concat}(\text{map}(w \mapsto w + [''], ws)) \\
 & \quad \{ \text{build form of } \text{concat} \} \\
 & = \text{build } (f z \mapsto \\
 & \quad \text{foldr } (x y \mapsto \text{foldr } f y x) z \text{ map}(w \mapsto w + [''], ws)) \\
 & \quad \{ \text{build form of } \text{map} \} \\
 & = \text{build } (f z \mapsto \\
 & \quad \text{foldr } (x y \mapsto \text{foldr } f y x) z (\mathbf{build } (f' z' \mapsto \\
 & \quad \text{foldr } (w b \mapsto f' (w \mapsto w + ['']) b) z' ws))) \\
 & \quad \{ \text{fusion law} \} \\
 & = \text{build } (f z \mapsto \\
 & \quad \text{foldr } (w b \mapsto (x y \mapsto \text{foldr } f y x) (w + ['']) b) z ws) \\
 & \quad \{ \beta - \text{reduction } x, y \} \\
 & = \text{build } (f z \mapsto \\
 & \quad \text{foldr } (w b \mapsto \text{foldr } f b (w + [''])) z ws) \\
 & \quad \{ \text{build form of } \text{++} \} \\
 & = \text{build } (f z \mapsto \\
 & \quad \text{foldr } (w b \mapsto \\
 & \quad \text{foldr } f b (\mathbf{build } (f' z' \mapsto \\
 & \quad \text{foldr } f' (\text{foldr } f' z' ['']) w))) z ws) \\
 & \quad \{ \text{fusion law} \} \\
 & = \text{build } (f z \mapsto \\
 & \quad \text{foldr } (w b \mapsto \\
 & \quad \text{foldr } (\text{foldr } f b ['']) w) z ws) \\
 & \quad \{ \text{substitute } (:) \text{ and } [] \text{ in the definition of } \text{build} \} \\
 & = \text{foldr } (w b \mapsto \text{foldr } (:) (\mathbf{foldr } (:) b ['']) w) [] ws \\
 & \quad \{ \text{evaluate the bold part} \} \\
 & = \text{foldr } (w b \mapsto \text{foldr } (:) ('' : b) w) [] ws
 \end{aligned}$$

The final reduced result is:

$$\text{join}(ws) = \text{foldr } (w b \mapsto \text{foldr } (:) ('' : b) w) [] ws$$

We can further expand the folding operation to obtain a definition with both good readability and performance.

$$\begin{cases} \text{join } [] = [] \\ \text{join } (w : ws) = h w \end{cases}$$

where :

$$\begin{cases} h [] = '' : \text{join } ws \\ h (x : xs) = x : h xs \end{cases}$$

Because $\text{concat} \circ \text{map}(f)$ is very common, many programming environments provide it in the optimized way as we deduced¹.

Although the second example demonstrates the power of fusion law, it also exposes a problem. The reduction process is complex and error prone, with plenty of repeated similar steps. It is exactly a typical case that machine performs better than human beings. Some programming environments have the fusion law built in the compiler[52]. What we need is to define the common list operations in the build...foldr form, then the rest boring work can be handled by machine. The compiler will help us reducing to optimized program, that avoid thrown away intermediate results and redundant recursions². As time goes on, more and more compilers will support this optimization tool.

5.1.5 Type constraint

Whenever we develop an abstraction tool, we should consider its application scope, understand when it will be invalid. For the fusion law, consider the below contradict results:

$$\begin{aligned} & \mathbf{foldr} \ f \ z \ (\mathbf{build} \ (c \ n \mapsto [0])) \\ = & \ (c \ n \mapsto [0]) \ f \ z && \text{fusion law} \\ = & \ [0] && \beta - \text{reduction} \end{aligned}$$

On the other hand:

$$\begin{aligned} & \mathbf{foldr} \ f \ z \ (\mathbf{build} \ (c \ n \mapsto [0])) \\ = & \ \mathbf{foldr} \ f \ z \ ((c \ n \mapsto [0]) \ (:) \ []) && \text{definition of build} \\ = & \ \mathbf{foldr} \ f \ z \ [0] && \beta - \text{reduction} \\ = & \ f(0, z) && \text{expand foldr} \end{aligned}$$

Obviously that $f(0, z)$ is not identical to $[0]$, even their types are not same³. The reason that leads to this contradiction is because $(c \ n \mapsto [0])$ is not a function that builds result of c and n . It tells us, the fusion law $\mathbf{foldr} \ f \ z \ (\mathbf{build} \ g) = g \ f \ z$, has type constraint for g . Its first argument can be c or f . Actually, it accepts a polymorphic binary operation $\forall A. \forall B. A \times B \rightarrow B$; The second argument is the start value of a polymorphic type B , the result type is also B . Write the binary operation in Curried form, we obtain the following type constraint for g .

$$g : \forall A. (\forall B. (A \rightarrow B \rightarrow B) \rightarrow B \rightarrow B)$$

In the above example that causes contradict results, the type is $\forall A. (\forall B. (A \rightarrow B \rightarrow B) \rightarrow B \rightarrow [Int])$. It does not satisfy the constraint. The corresponding type constraint for \mathbf{build} is:

$$\mathbf{build} : \forall A. (\forall B. (A \rightarrow B \rightarrow B) \rightarrow B \rightarrow B) \rightarrow \mathbf{List} \ A$$

Because there are two polymorphic types A and B , it is called rank-2 type polymorphic.

5.1.6 Fusion law in category theory

foldr/build fusion law can be deduced and extended from the category theory. Foldr/build is one of the fusion laws in category theory. They are named as *shortcut fusion* nowadays,

¹For example, `concatMap` in Haskell, and `flatMap` in Java and Scala.

²Haskell standard library for example, provides most list functions in build...foldr form.

³Unless the extreme case that $f = (:)$, $z = []$.

which play important roles in compiler and program optimization. We can't cover all of them in a chapter. Readers can refer to [54] to understand shortcut fusion theory and its practice in depth.

We introduced F-algebra, initial algebra, and catamorphism in previous chapter. Because initial algebra is the initial object, it has unique arrow to every other algebra, as shown in below diagram.

$$\begin{array}{ccc}
 \mathbf{F}I & \xrightarrow{i} & I \\
 \mathbf{F}(\mathbf{a}) \downarrow & & \downarrow (\mathbf{a}) \\
 \mathbf{F}A & \xrightarrow{a} & A
 \end{array}$$

The arrow from the initial algebra (I, i) to algebra (A, a) can be defined with catamorphism (\mathbf{a}) . If there exists another F-algebra (B, b) , and there is arrow from A to B as $A \xrightarrow{h} B$, then we can also draw (B, b) at the bottom:

$$\begin{array}{ccc}
 \mathbf{F}I & \xrightarrow{i} & I \\
 \mathbf{F}(\mathbf{a}) \downarrow & & \downarrow (\mathbf{a}) \\
 \mathbf{F}A & \xrightarrow{a} & A \\
 \mathbf{F}(h) \downarrow & & \downarrow h \\
 \mathbf{F}B & \xrightarrow{b} & B
 \end{array}$$

Since (I, i) is the initial algebra, it must have the unique arrow to (B, b) . Hence there must be arrow from I to B , which can be represented as catamorphism (\mathbf{b}) , as shown in below diagram.

$$\begin{array}{ccc}
 \mathbf{F}I & \xrightarrow{i} & I \\
 \mathbf{F}(\mathbf{a}) \downarrow & & \downarrow (\mathbf{a}) \\
 \mathbf{F}A & \xrightarrow{a} & A \\
 \mathbf{F}(h) \downarrow & & \downarrow h \\
 \mathbf{F}B & \xrightarrow{b} & B
 \end{array}$$

(\mathbf{b})

From this diagram we can find that, if and only if there is h , such that the square at the bottom commutes, then the path from I through A to B and the path directly from I to B also commutes. It is called the fusion law of initial algebra, denote as:

$$A \xrightarrow{h} B \Rightarrow h \circ (\mathbf{a}) = (\mathbf{b}) \iff h \circ a = b \circ \mathbf{F}(h) \quad (5.8)$$

What does the fusion law for initial algebra mean? In previous chapter, we explained that catamorphism can turn the non-recursive computation to folding on recursive structures. For example, when the functor \mathbf{F} is $\mathbf{List}FA$, where A is a given object, the arrow

$a = f + z$ (coproduct of f and z), the initial arrow is $i = (:) + []$, and the catamorphism is $\langle a \rangle = foldr(f, z)$. If denote $b = c + n$, then the fusion law can be written as:

$$h \circ foldr(f, z) = foldr(c, n)$$

It means the transformation after folding can be simplified to only folding. Takano and Meijer in 1995 further abstracted the catamorphism $\langle a \rangle$ to build some abstract algebra structure $g a$ from a , such that the fusion law can be extended to[55]:

$$A \xrightarrow{h} B \Rightarrow h \circ g a = g b \quad (5.9)$$

This extended fusion law is called *acid rain law*⁴. On the other hand, from the initial algebra I to A , there exists arrow $I \xrightarrow{\langle a \rangle} A$, hence substitute h on the left hand of acid rain law with $\langle a \rangle$, substitute a with i , and substitute b with a , we obtain:

$$I \xrightarrow{\langle a \rangle} A \Rightarrow \langle a \rangle \circ g i = g a \quad (5.10)$$

For the list example, the catamorphism $\langle a \rangle$ is $foldr(f, z)$; the initial algebra i for list is $(:) + []$; define $build(g) = g (:) []$, and substitute it into the left side of acid rain law, we obtain the $foldr/build$ fusion law:

$$\begin{aligned} & \langle a \rangle \circ g i = g a && \text{acid rain law} \\ \Rightarrow & foldr f z (g i) = g a && foldr \text{ is catamorphism for list} \\ \Rightarrow & foldr f z (g (:) []) = g a && \text{the initial algebra } i \text{ for list is } (:) [] \\ \Rightarrow & foldr f z (g (:) []) = g f z && \text{substitute } a \text{ with } f, z \\ \Rightarrow & \mathbf{foldr} f z (\mathbf{build} g) = g f z && \text{reverse of build} \end{aligned}$$

Hence we proved $foldr/build$ fusion law for list with category theory[54].

Exercise 5.1

1. Verify that folding from left can also be defined with $foldr$:

$$foldl f z xs = foldr (b g a \mapsto g (f a b)) id xs z$$

2. Prove the below build... $foldr$ forms hold:

$$\begin{aligned} concat xs &= build (f z \mapsto foldr (xs x \mapsto foldr f x xs) z xs) \\ map f xs &= build (\oplus z \mapsto foldr (y ys \mapsto (f y) \oplus ys) z xs) \\ filter f xs &= build (\oplus z \mapsto foldr (x xs' \mapsto \begin{cases} f(x) : & x \oplus xs' \\ \text{otherwise} : & xs' \end{cases}) z xs) \\ repeat x &= build (\oplus z \mapsto let r = x \oplus r \text{ in } r) \end{aligned}$$

3. Simplify the quick sort algorithm.

$$\begin{cases} qsort [] = [] \\ qsort (x : xs) = qsort [a | a \in xs, a \leq x] \uplus [x] \uplus qsort [a | a \in xs, a > x] \end{cases}$$

Hint: turn the ZF-expression⁵ into $filter$.

4. Verify the type constraint of fusion law with category theory. Hint: consider the type of the catamorphism.

⁴Because fusion law can help to eliminate intermediate results, it was named as *deforestation* before.

⁵Known as Zermelo-Fraenkel expression in set theory. In the form $\{f(x) | x \in X, p(x), q(x), \dots\}$ to build set. We'll meet it again in the next chapter about infinity and set theory.

5.2 Make a Century

Our second example is from Bird's *Pearls of Functional Algorithm Design* ([53], chapter 6). Knuth leaves an exercise in his *The Art of Computer Programming* ([56], Vol 4). Write down 1 2 3 4 5 6 7 8 9 in a row, only allow to insert + and \times symbol between these numbers, parentheses are not allowed. How to make the final calculation result be 100?

For instance:

$$12 + 34 + 5 \times 6 + 7 + 8 + 9 = 100$$

It looks like a mathematics puzzle for primary school students. It is also a good programming exercise if requests to find all the possible solutions. The straightforward way is to brute-force exhaustive search with machine. For every position between two digits, there are 3 possible options: (1) insert nothing; (2) insert +; or (3) insert \times . Since there are 8 spaces among 9 digits, there are $3^8 = 6561$ possible options in total. It's a piece of cake for modern computer to calculate these 6561 results, and figure out which are 100.

5.2.1 Exhaustive search

Let's first model the expression consist of digits, +, and \times . Consider the example:

$$12 + 34 + 5 \times 6 + 7 + 8 + 9$$

As multiplication is prior to addition, we can treat it as a list of sub-expressions separated by +. Hence the above example is identical to:

$$\text{sum}[12, 34, 5 \times 6, 7, 8, 9]$$

In summary, we define an expression as one or multiple sub-expressions separated by + in the form of $t_1 + t_2 + \dots + t_n$, or in the list form of $\text{expr} = [t_1, t_2, \dots, t_n]$. The formal definition is:

```
type Expr = [Term]
```

For every sub-expression, we treat it as one or multiple factors separated by \times . For example, $5 \times 6 = \text{product}[5, 6]$. This definition is also applicable for a single number, like 34. It can be treated as $34 = \text{product}[34]$. Hence we can define sub-expression as the product of factors $f_1 \times f_2 \times \dots \times f_m$, or in the list form: $\text{term} = [f_1, f_2, \dots, f_m]$:

```
type Term = [Factor]
```

Every factor is consist of digits. For example, 34 has two digits, while 5 has only 1. In summary $\text{factor} = [d_1, d_2, \dots, d_k]$.

```
type Factor = [Int]
```

It's a folding process to evaluate the decimal value from a list of digits. For example, $[1, 2, 3] \Rightarrow (((1 \times 10) + 2) \times 10) + 3$. We can define it as a function.

$$\text{dec} = \text{foldl} (n \ d \mapsto n \times 10 + d) \ 0$$

The exhaustive search need examine every possible expression evaluates to 100 or not. We need define a function to evaluate a given expression. From the expression definition, we need define the process to recursively evaluate every sub-expression (term), and sum them up; further, we also need recursively evaluate every factor, and multiply them up; in the end, we call dec function to evaluate each factor.

$$eval = sum \circ map (product \circ (map dec))$$

Obviously, this definition can be optimized with the fusion law. We leave the detailed deduction as the exercise, and directly give the result here:

$$eval = foldr (t ts \mapsto (foldr ((\times) \circ fork(dec, id)) 1 t) + ts) 0$$

Where $fork(f, g) x = (f x, g x)$. It applies two functions to a given variable separately, and forms a pair. From this result, we can write a definition with both good performance and readability:

$$\begin{cases} eval [] = 0 \\ eval(t : ts) = product (map dec t) + eval(ts) \end{cases}$$

According to this definition, the result is 0 if the expression is empty; otherwise, we take the first sub-expression, evaluate all its factors, and multiply them together. Then add it to the result of the rest sub-expressions. We repeatedly call *eval* for all possible expressions, and filter the candidates unless they evaluate to 100.

$$filter (e \mapsto eval(e) == 100) es$$

Where *es* is the set contains all the possible expressions of 1 to 9. How to generate this set? Starting from empty set, every time, we pick a number from 1 to 9 to expand the expressions. From the empty set and a given number *d*, we can only build one expression $[[[d]]]$. The inner most is the singleton factor consist of the only digit *d*, hence *fact* = $[d]$; then there is the sub-expression consist of this factor *term* = $[fact] = [[d]]$; this only sub-expression forms the final expression $[term] = [[fact]] = [[[d]]]$. We define this edge expression building process as:

$$expr(d) = [[[d]]]$$

For the common cases, we start from right, repeatedly pick numbers 9, 8, 7, ... to expand the expression. On top of a set of expressions $[e_1, e_2, \dots, e_n]$, how to expand all possible expressions from the next digit *d*? We've mentioned that there are 3 options between two numbers: 1) insert nothing; 2) insert +; 3) insert \times . Let's see what does each option mean for every expression *e_i* in the set. Write *e_i* as the sum of sub-expressions $e_i = t_1 + t_2 + \dots$, where the first sub-expression *t₁* can be further written as product of factors $t_1 = f_1 \times f_2 \times \dots$

1. Insert nothing means prepend digit *d* to the first factor in the first sub-expression in *e_i*. Hence *d* : *f₁* form a new factor. For example, let *e_i* be $8 + 9$, and *d* be 7. Writing 7 before $8 + 9$ without inserting any symbol, gives a new expression $78 + 9$;
2. Insert \times means to use *d* form a new factor $[d]$, then prepend it before the first sub-expression in *e_i*. Hence $[d] : t_1$ gives a new sub-expression. For the same example of $8 + 9$, we write 7 before it, and insert a \times symbol between 7 and 8. It gives the new expression $7 \times 8 + 9$;
3. Insert + means to use *d* form a new sub-expression $[[d]]$, then prepend it to *e_i* to form a new expression $[[d]] : e_i$. For the same example of $8 + 9$, we write 7 before it, and insert a + between 7 and 8. It gives the new expression $7 + 8 + 9$.

Below definition summarizes these three options:

```
add d ((ds:fs):ts) = [((d:ds):fs):ts,
                      ([d]:ds:fs):ts,
                      [[[d]]:(ds:fs):ts]
```

Where e_i is written in the form of $(ds : fs) : ts$. The first sub-expression in e_i is $ds : fs$, and the first factor in the first sub-expression is ds . From every expression, we can expand 3 new ones. Given the expression list $[e_1, e_2, \dots, e_n]$, we can expand for everyone, then concatenate the results together. It is exactly the *concatMap* function we mentioned in the previous section.

$$\begin{cases} \text{extend } d [] = [\text{expr}(d)] \\ \text{extend } d es = \text{concatMap} (\text{add } d) es \end{cases}$$

Therefore, we obtain the complete definition of the exhaustive search method.

$$\text{filter } (e \mapsto \text{eval}(e) == 100) (\text{foldr extend} [1..9]) \quad (5.11)$$

5.2.2 Improvement

How can we improve the exhaustive search? Observe the right to left expand process. We write down 9 first, then expand 3 new expressions with 8. They are: 89 , $8 \times 9 = 72$, and $8 + 9 = 17$. In the next step when expand with 7, expression 789 exceeds 100 obviously. We can drop it immediately. Any new expressions expanded to the left of 789 won't be 100. We can safe to drop them all to avoid unnecessary computation. Similarly, 7×89 exceeds 100, hence can be dropped to avoid further expansion on the left. Only $7 + 89$ is the possible expression. Next we expand 78×9 , it is bigger than 100, hence is dropped for further expansion...

In summary, we can evaluate the expression during the expanding process. Whenever exceeds 100, we immediately drop it to avoid further expansion. The whole process looks like a tree growing. When the expression in a branch exceeds 100, we cut that branch off.

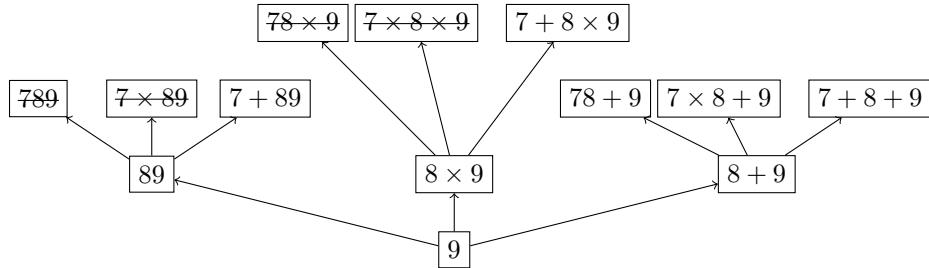


Figure 5.3: The process to expand expressions looks like a tree growing.

In order to evaluate the expression during expansion easily, we separate the value of the expression into 3 parts: the value of the first factor in the first sub-expression f ; the product of the rest factors in the first sub-expression v_{fs} ; and the sum of the rest sub-expressions v_{ts} . The overall value of the expression can be calculated out of them as $f \times v_{fs} + v_{ts}$.

When expand a new digit d on the left, corresponding to the 3 options, the expression and value are updated as the following:

1. Insert nothing, prepend d before the first factor as its most significant digit. The expression value is $(d \times 10^{n+1} + f) \times v_{fs} + v_{ts}$, where n is the number of digits of f . The 3 parts of the value update to: $f' = d \times 10^{n+1} + f$, v_{fs} and v_{ts} are unchanged;
2. Insert \times , put d as the first factor in the first sub-expression. The expression value is $d \times f \times v_{fs} + v_{ts}$. The 3 parts of the value update to: $f' = d$, $v_{fs'} = f \times v_{fs}$, while v_{ts} is unchanged;

3. Insert $+$, use d as the new first sub-expression. The expression value is $d + f \times v_{fs} + v_{ts}$. The 3 parts of the value update to: $f' = d, v_{fs'} = 1, v_{ts'} = f \times v_{fs} + v_{ts}$.

To calculate 10^{n+1} in the first option easily, we can record the exponent as the fourth part of the expression value. As such, the value is represented as a 4-tuple (e, f, v_{fs}, v_{ts}) . The function to calculate the value is defined as:

$$value(e, f, v_{fs}, v_{ts}) = f \times v_{fs} + v_{ts} \quad (5.12)$$

When expand expression, we also update the 4-tuple at the same time:

$$\begin{aligned} add \quad d (((ds : fs) : ts), (e, f, v_{fs}, v_{ts})) = \\ & [((d : ds) : fs) : ts, (10 \times e, d \times e + f, v_{fs}, v_{ts})), \\ & (([d] : ds : fs) : ts, (10, d, f \times v_{fs}, v_{ts})), \\ & ([[d]] : (ds : fs) : ts, (10, d, 1, f \times v_{fs} + v_{ts}))] \end{aligned} \quad (5.13)$$

For every expression and 4-tuple, we call *value* function to calculate the value, drop it if exceeds 100. Then concatenate the candidate expressions and the 4-tuples to a list. According to this idea, we re-define the previous *extend* function as below:

$$\begin{cases} expand \ d \ [] = [(expr(d), (10, d, 1, 0))] \\ expand \ d \ evs = concatMap ((filter ((\leq 100) \circ value \circ snd)) \circ (add \ d)) \ evs \end{cases} \quad (5.14)$$

We are ready to fold on *expand*, generate all candidate expressions and 4-tuples that do not exceed 100. Finally, we calculate the result from the 4-tuples, and leave only those equal to 100.

$$map \ fst \circ filter \ ((= 100) \circ value \circ snd) \ (foldr \ expand \ [] \ [1, 2, \dots, 9]) \quad (5.15)$$

The complete program based on this definition is given in the appendix of this chapter. There are total 7 expressions evaluate to 100.

$$\begin{aligned} 1 : & 1 \times 2 \times 3 + 4 + 5 + 6 + 7 + 8 \times 9 \\ 2 : & 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 \times 9 \\ 3 : & 1 \times 2 \times 3 \times 4 + 5 + 6 + 7 \times 8 + 9 \\ 4 : & 12 + 3 \times 4 + 5 + 6 + 7 \times 8 + 9 \\ 5 : & 1 + 2 \times 3 + 4 + 5 + 67 + 8 + 9 \\ 6 : & 1 \times 2 + 34 + 5 + 6 \times 7 + 8 + 9 \\ 7 : & 12 + 34 + 5 \times 6 + 7 + 8 + 9 \end{aligned}$$

Exercise 5.2

1. Use the fusion law to optimize the expression evaluation function:

$$eval = sum \circ map \ (product \circ (map \ dec))$$

2. How to expand all expressions from left?
 3. The following definition converts expression to string:

$$str = (join \ "+") \circ (map \ ((join \ "\times") \circ (map \ (show \circ dec))))$$

Where *show* converts number to string. Function *join*(*c*, *s*) concatenates multiple strings *s* with delimiter *c*. For example: *join*("#", ["abc", "def"]) = "abc#def". Use the fusion law to optimize *str*.

5.3 Further reading

Program deduction is a special mathematical deduction. With this tool, we can start from intuitive, raw, and unoptimized definition, through a formalized methods and theorems, step by step convert it to elegant and optimized result. Bird gives many such examples in his *Pearls of Functional Algorithm Design*[53].

The correctness of program deduction is based on mathematics. Instead of relying on human intuition, we need a dedicated theory, that can formalize varies of programs into strict mathematical form. The foldr/build fusion law is such an example. In the 1993 paper[52], people developed a tool to simplify program systematically. As the category theory being widely adopted in programming, a series of fusion laws have been developed[54], and applied to program deduction and optimization.

5.4 Appendix - example source code

The `build` and `concatMap` definition in Haskell.

```
build :: forall a. (forall b. (a -> b -> b) -> b -> b) -> [a]
build g = g (:)
[]

concatMap f xs = build (\c n -> foldr (\x b -> foldr c b (f x)) n xs)
```

Exhaustive search solution for ‘Making a Century’ puzzle:

```
type Expr = [Term]      -- | T1 + T2 + ... Tn
type Term = [Factor]    -- | F1 * F2 * ... Fm
type Factor = [Int]     -- | d1d2...dk

dec :: Factor -> Int
dec = foldl (\n d -> n * 10 + d) 0

expr d = [[[d]]]  -- | single digit expr

eval [] = 0
eval (t:ts) = product (map dec t) + eval ts

extend :: Int -> [Expr] -> [Expr]
extend d [] = [expr d]
extend d es = concatMap (add d) es where
  add :: Int -> Expr -> [Expr]
  add d ((ds:fs):ts) = [((d:ds):fs):ts,
                        ([d]:ds:fs):ts,
                        [[[d]]:(ds:fs):ts]

sol = filter ((==100) . eval) . foldr extend []
```

The improved exhaustive search solution:

```
value (_, f, fs, ts) = f * fs + ts

expand d [] = [(expr d, (10, d, 1, 0))]
expand d evs = concatMap ((filter ((<= 100) . value . snd)) . (add d)) evs
where
  add d (((ds:fs):ts), (e, f, vfs, vts)) =
    [(((d:ds):fs):ts, (10 * e, d * e + f, vfs, vts)),
     ([[d]]:(ds:fs):ts, (10, d, f * vfs, vts)),
     [[[d]]:(ds:fs):ts, (10, d, 1, f * vfs + vts))]

sol = map fst . filter ((==100) . value . snd) . foldr expand []
```


Chapter 6

Infinity

I see it, but I don't believe it.

— Georg Cantor, in a letter to
Dedekind in 1877

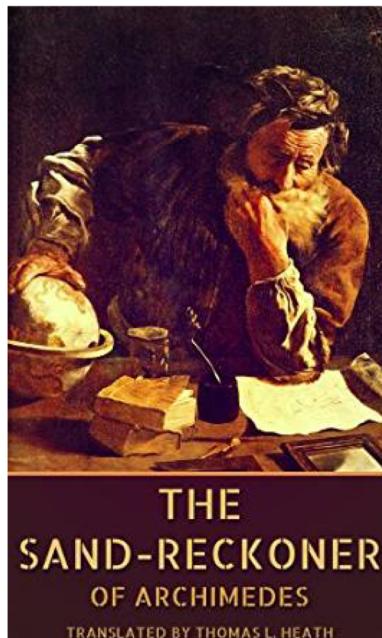
Long time ago, our ancestor looked up at the starry sky, facing the vast galaxy, and asked, how big is the world we live in? As the intelligence being, our mind exceeds ourselves, exceeds our planet and universe. We keep thinking about the concept of infinity. People first abstracted numbers from concrete things. From three goats hunted, three fruits collected, three jars made, abstracted number three to represent any three things. At early time, the numbers were not big. They were enough to satisfy everyday life, hunting, and work. As civilization evolved, people started trading things. Various numbering systems were developed to support the bigger and bigger numbers. At some time point, we came up with the question: what is the biggest number? There were two different opinions to it. Some people didn't think the question make sense. It's already big enough for numbers like thousands or millions in ancient time in everyday work and life. We needn't trouble ourselves with big numbers that would never being used. It's safe to consider for example, the number of sand-grains in the world is infinity. In ancient Greece, people thought ten thousands was a very big number, and named it 'murias'. It finally changed to 'myriad', means infinity[57]. In Buddhism, people also use 'the sand in Ganges River' to indicate the numbers that are too large to count. In the Mahayana Buddhist classic work *The Diamond Sutra*, it said: "If a virtuous man or woman filled a number of universes, as great as the number of sand-grains in all these rivers, with the seven treasures, and gave them all away in alms (dana), would his or her merit be great?" Other people had different opinion. Ancient Greek mathematician, Archimedes believed, even the sand-grains that filled the whole universe, can be represented with a definite number. In his book, *The Sand-Reckoner*,



Escher, Circle limit IV (Heaven and Hell), 1960

Archimedes said:

THERE are some, King Gelon, who think that the number of the sand is infinite in multitude; and I mean by the sand not only that which exists about Syracuse and the rest of Sicily but also that which is found in every region whether inhabited or uninhabited. Again there are some who, without regarding it as infinite, yet think that no number has been named which is great enough to exceed its multitude. And it is clear that they who hold this view, if they imagined a mass made up of sand in other respects as large as the mass of the Earth, including in it all the seas and the hollows of the Earth filled up to a height equal to that of the highest of the mountains, would be many times further still from recognising that any number could be expressed which exceeded the multitude of the sand so taken. But I will try to show you by means of geometrical proofs, which you will be able to follow, that, of the numbers named by me and given in the work which I sent to Zeuxippus, some exceed not only the number of the mass of sand equal in magnitude to the Earth filled up in the way described, but also that of a mass equal in magnitude to the universe.



The cover of *The Sand-Reckoner*. Archimedes Thoughtful by Domenico Fetti (1620)

Archimedes thought it *only* need 10^{63} sand-grains to fill the universe. The universe he meant was the sphere of the fixed star, which was about twenty thousands times the radius of the Earth. We know the observable universe is about 46.5 billion light-years nowadays, consist of around 3×10^{74} atoms¹. Archimedes was definitely genius in ancient Greek time, he demonstrated how to quantify the ‘infinite big’ things. There are many words in different languages serve as the unit for big numbers. The following table list these words in Chinese, they increase for every 10^4 ([58], pp31).

¹Also said to have 10^{80} to 10^{87} elementary particles.

京	10^{16}	载	10^{44}
垓 (gāi)	10^{20}	极	10^{48}
秭 (zǐ)	10^{24}	恒河沙	10^{52}
穰 (ráng)	10^{28}	阿僧祇 (zhī)	10^{56}
沟	10^{32}	那由他	10^{60}
涧	10^{36}	不可思议	10^{64}
正	10^{40}	无量大数	10^{68}

Many such words come from Buddhism, like ‘恒河沙’, means the sand-grain in Ganges River. It has 52 zeros after 1. Below table lists the unit words in English. Starting from one, there is a unit for every 1000 magnitude. Compare to 10000 magnitude step in Chinese, we can see the culture difference.

thousand	10^3	quattuordecillion	10^{45}	octovigintillion	10^{87}
million	10^6	quindecillion	10^{48}	novemvigintillion	10^{90}
billion	10^9	sexdecillion	10^{51}	trigintillion	10^{93}
trillion	10^{12}	septdecillion	10^{54}	untrigintillion	10^{96}
quadrillion	10^{15}	octodecillion	10^{57}	duotrigintillion	10^{99}
quintillion	10^{18}	novemdecillion	10^{60}	googol	10^{100}
sexillion	10^{21}	vigintillion	10^{63}		
septillion	10^{24}	unvigintillion	10^{66}		
octillion	10^{27}	duovigintillion	10^{69}		
noniliion	10^{30}	trevigintillion	10^{72}		
decillion	10^{33}	quattuorvigintillion	10^{75}		
undecillion	10^{36}	quinvigintillion	10^{78}		
duodecillion	10^{39}	sexvigintillion	10^{81}		
tredecillion	10^{42}	seprvigintillion	10^{84}		

The last unit, googol, was coined in 1920 by 9-year-old Milton Sirotta, nephew of U.S. mathematician Edward Kasner. It is written as the digit 1 followed by one hundred zeros. The internet company Google’s name came from this word[59].

6.1 The infinity concept

Whether there exists infinity that beyond all concrete numbers? It is not only a mathematical problem, but also a philosophical problem. Infinity also leads to the concept of infinitesimal (infinitely small). Ancient Greek philosopher, Zeno of Elea thought of a set of problems about infinity. Some of them are preserved in Aristotle’s *Physics*. Among them, four paradoxes are most famous.

The first one is the most popular, named Achilles and the tortoise paradox. Achilles In Greek mythology, was a hero of the Trojan War He is the greatest of all the Greek warriors, and is the central character of Homer’s *Iliad*. In this paradox, Achilles is in a footrace with the tortoise. Achilles allows the tortoise ahead start, for example 100 meters. Supposing that each racer starts running at some constant speed, one faster than the other. After some finite time, Achilles will have run 100 meters, bringing him to the tortoise’s starting point. During this time, the tortoise has run a much shorter distance, say 2 meters. It will then take Achilles some further time to run that distance, by which time the tortoise will have advanced farther; and then more time still to reach this third point, while the tortoise moves ahead. Thus, whenever Achilles arrives somewhere the tortoise has been, he still has some distance to go before he can even reach the tortoise. as shown in figure 6.3. Although it contradicts our common sense in real life, the argument

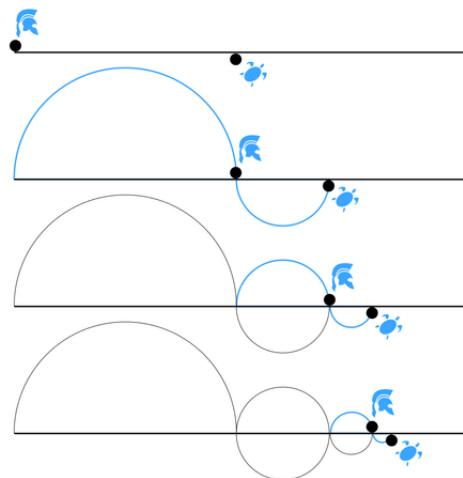


Figure 6.3: Achilles and the tortoise

is so convincing. This paradox attracted many great minds for thousands of years. Lewis Carroll, and Douglas Hofstadter even took Achilles and the tortoise as figures in their literary works.

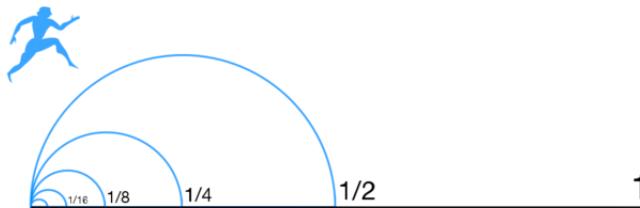


Figure 6.4: Dichotomy paradox

The second one is called Dichotomy paradox. Atalanta is a character in Greek mythology, a virgin huntress. Suppose Atalanta wishes to walk to the end of a path. Before she can get there, she must get halfway there. Before she can get halfway there, she must get a quarter of the way there. Before traveling a quarter, she must travel one-eighth; before an eighth, one-sixteenth; and so on. This description requires one to complete an infinite number of tasks, which Zeno maintains is an impossibility. The paradoxical conclusion then would be that travel over any finite distance can neither be completed nor begun, and so all motion must be an illusion.

The third one is called arrow paradox. Zeno states that for motion to occur, an object must change the position which it occupies. He gives an example of an arrow in flight. In any one (duration-less) instant of time, the arrow is neither moving to where it is, nor to where it is not. It cannot move to where it is not, because no time elapses for it to move there; it cannot move to where it is, because it is already there. In other words, at every instant of time there is no motion occurring. If everything is motionless at every instant, and time is entirely composed of instants, then motion is impossible. Whereas the first two paradoxes divide space, this paradox starts by dividing time—and not into segments,

but into points.



Figure 6.5: Arrow paradox

The fourth one is called the moving rows paradox, or stadium paradox. It's also about dividing time into atomic points. As shown in figure 6.6, there are three rows in the stadium. Each row being composed of an equal number of bodies. At the beginning, they are all aligned. At the smallest time duration, row A stays, row B moves to the right one space unit, while row Γ moves to the left one space unit. To row B, row Γ actually moves two space units. It means, there should be time duration that Γ moves one space unit relative to B. And it is the half time of the smallest duration. Since the smallest duration is atomic, it involves the conclusion that half a given time is equal to that time.

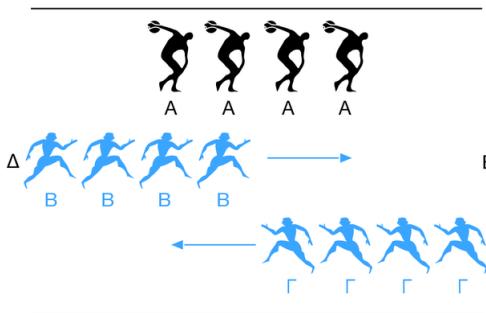


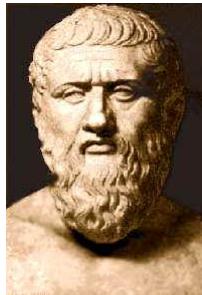
Figure 6.6: Moving rows paradox

Zeno's paradoxes are easy to understand. However, the conclusion is surprising. In our common sense, motion and time are so real. Achilles must be able to catch up the tortoise. However, it is hard to solve Zeno's paradox. From Aristotle to Bertrand Russel, from Archimedes to Herman Weyl, all proposed varies of solutions to Zeno's paradoxes[60].

Zeno was ancient Greek philosopher. He was born in Elea, which was a Greek colony located in present-day southern Italy. Little is known for certain about Zeno's life. In the dialogue of Parmenides, Plato describes a visit to Athens by Zeno and Parmenides, at a time when Parmenides is "about 65", Zeno is "nearly 40", and Socrates is "a very young man". Assuming an age for Socrates of around 20 and taking the date of Socrates' birth as 469 BC gives an approximate date of birth for Zeno of 490 BC. Some less reliable details of Zeno's life are given by Diogenes Laërtius in his *Lives and Opinions of Eminent Philosophers*. It said Zeno was the adopted son of Parmenides. He was skilled to argue both sides of any question, the universal critic. And that he was arrested and perhaps killed at the hands of a tyrant of Elea[8].

Zeno was the primary member of the Eleatics, which were a pre-Socratic school of philosophy founded by Parmenides in the early fifth century BC in the ancient town of Elea. It's another important school after Pythagoras. The Eleatics rejected the epistemological validity of sense experience, and instead took logical standards of clarity and

necessity to be the criteria of truth. Of the members, Parmenides and Melissus built arguments starting from sound premises. Zeno, on the other hand, primarily employed the *reductio ad absurdum*, attempting to destroy the arguments of others by showing that their premises led to contradictions. Zeno is best known for his paradoxes, which Bertrand Russell described as “immeasurably subtle and profound”.



Zeno of Elea, About 490BC - 425BC

Ancient Chinese philosophers developed equivalents to some of Zeno’s paradoxes. From the surviving Mohist School of Names book of logic which states, in the archaic ancient Chinese script, “a one-foot stick, every day take away half of it, in a myriad ages it will not be exhausted.”

Zeno’s paradoxes caused deep confusion to the ancient Greeks. The views about time, space, infinity, continuity, and movement are critical to the later philosophers and mathematicians even today. How to understand infinity, became a problem that must be solved by ancient Greek philosophers.

6.1.1 Potential infinity and actual infinity

Aristotle studied Zeno’s paradoxes deeply. One of the most important contributions that Aristotle had made to the study of infinity is identifying a dichotomy between what Aristotle calls the *potential infinite* and the *actual infinite*. This work fundamentally influenced the later development of mathematics[8]. Potential infinity is a process that never stops. It can be a group of “things” that continues without terminating, going on or repeating itself over and over again with no recognizable ending point. The obvious example is the group of natural numbers. No matter where you are while listing or counting out natural numbers, there always exists another number to proceed. Also, in Euclid geometry, a line with a starting point could extend on without end, but could still be potentially infinite because one can add on more length to a finite length to allow it to extend². The actual infinite involves never-ending sets or “things” within a space that has a beginning and end; it is a series that is technically “completed” but consists of an infinite number of members. According to Aristotle, actual infinities cannot exist because they are paradoxical. It is impossible to say that you can always “take another step” or “add another member” in a completed set with a beginning and end, unlike a potential infinite. It is ultimately Aristotle’s rejection of the actual infinite that allowed him to refute Zeno’s paradox.

Although Aristotle did disprove the existence of the actual infinite finally, and tended to reject a lot of major concepts in mathematics, the importance of mathematics was still never belittled in Aristotle’s eyes. Aristotle argued that actual infinity as it is

²Euclid avoided to use the term ‘infinitely extend’ in his work. instead he said a line can be extended any long as needed. This is a common treatment in ancient Greece.

not applicable to geometry and the universal, is not relevant to mathematics, making potential infinity all that actually is important.

Aristotle's viewpoint to infinity is typical in ancient Greece. The dichotomy and controversy about potential infinity and actual infinity have been influential till today. Despite of these arguments, the ancient Greek mathematicians achieved amazing result with the potential infinity concept. One success story was that Euclid proved there are infinitely many prime numbers. It is considered one of the most beautiful proofs in history.

Proposition 6.1.1 (Euclid's Elements, Book IX, Proposition 20). *Prime numbers are more than any assigned multitude of prime numbers*[1].

Euclid indented to avoid using term like 'infinitely many' when stated this proposition. Such treatment is very common in *Elements*. It's famous that Euclid uses reduction to absurdity in his proof. We explain it in modern language.

Proof. Suppose there are finite prime numbers p_1, p_2, \dots, p_n . Euclid makes a new number:

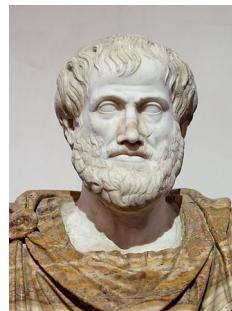
$$p_1 p_2 \dots p_n + 1$$

Which is the product of the n prime numbers plus one. It is either prime or not.

- If it is prime, definitely, it does not equal to any one from p_1 to p_n , hence it's a new prime not in the list;
- If it is not prime, then it must have a prime factor p . However, no one from p_1 to p_n divides this number. It means p is different from any one from p_1 to p_n , hence it's a new prime beyond those in the list.

In both cases, we obtain a new prime number. It contradicts the assumption that there are finite prime numbers. Therefore, there are infinitely many prime numbers. \square

From today's view point, Euclid obtained a kind of indirect 'proof of existence'. By using proof by contradiction, he proved there are infinitely many prime numbers, but did not give a way to list them. It is quite natural in mathematical proofs now. However, it led to hotly debating about the fundamentals of mathematics in the late 19th and early 20th Century. We'll return to this topic in next chapter.



Aristotle, 384BC - 322BC

Aristotle was a great philosopher and polymath in ancient Greece. Along with his teacher Plato, he has been called the 'Father of Western Philosophy'. Little is known about his life. Aristotle was born in the city of Stagira in Northern Greece in 384BC. His father died when Aristotle was a child, and he was brought up by a guardian. At the age of seventeen or eighteen, he joined Plato's Academy in Athens and remained there

for 20 years until Plato died. This period of study in Plato's Academy deeply influenced Aristotle's life. Socrates was Plato's teacher, and Aristotle was taught by Plato. He soon became an outstanding scholar, and Plato called him "the Spirit of the Academy". But Aristotle is not a man who only admires authority without his own opinions. He studied hard, and even established a library for himself.

Shortly after Plato died around 347BC, Aristotle left Athens. The traditional story about his departure records that he was disappointed with the Academy's direction after control passed to Plato's nephew. After that, he traveled around. In 343 BC, Aristotle was invited by Philip II of Macedon to become the tutor to his 13 years old son Alexander. Aristotle was appointed as the head of the royal academy of Macedon. During Aristotle's time in the Macedonian court, he gave lessons not only to Alexander, but also to two other future kings: Ptolemy and Cassander. It was under the influence of Aristotle that Alexander the Great cared about science and respected knowledge.

In 335BC, Philip II died. Aristotle returned to Athens, establishing his own school there known as the Lyceum. Aristotle conducted courses at the school for the next twelve years. In this period in Athens, between 335 and 323 BC, Aristotle composed many of his works. He studied and made significant contributions to logic, metaphysics, mathematics, physics, biology, botany, ethics, politics, agriculture, medicine, dance and theatre. There was legend that Aristotle had a habit of walking while lecturing along the walkways covered with colonnades. It was for this reason that his school was named "Peripatetic" (an ancient Greek word, which means "of walking" or "given to walking about"). Aristotle used the language that was much more obscure than Plato's Dialogue. Many of his works are based on lecture notes, and some are even the notes of his students. Aristotle was considered as the first author of textbooks in the western world.

Following Alexander's death, anti-Macedonian sentiment in Athens was rekindled. In 322 BC, his enemies reportedly denounced Aristotle for impiety, prompting him to flee to his mother's family estate in Euboea. He said: "I will not allow the Athenians to sin twice against philosophy" – a reference to Athens's trial and execution of Socrates. He died on Euboea of natural causes later that same year, at the age of 63.

More than 2300 years after his death, Aristotle remains one of the most influential people who ever lived. He contributed to almost every field of human knowledge in ancient time, and he was the founder of many new fields. Among countless achievements, Aristotle was the founder of formal logic, pioneered the study of zoology, and left every future scientist and philosopher in his debt through his contributions to the scientific method.

6.1.2 Method of exhaustion and calculus

Some other ancient Greek mathematicians took the practical approach about infinity. They developed the method of exhaustion and made surprising achievements.

The idea of exhaustion originated in the late 5th Century BC with Antiphon (About 480BC - 410BC), when he tried to solve one of the three classic geometric problems, square the circle³. To approximate the area of a circle, Antiphon started from an inscribed square, then repeatedly double the number of the sides to obtain octagons, hexagons... As the area of the circle gradually "exhausts", the side length of inscribed polygons gets smaller and smaller. Antiphon thought the polygon would eventually coincide with the circle. This is the idea of exhaustion. The method was made rigorous a few decades later by Eudoxus of Cnidus, who used it to calculate areas and volumes. The correctness of

³The other two are trisecting the angle, and doubling the cube. Given a circle, ancient Greeks attempted to seek the solution to draw a square that has the same area with only straightedge and compass. Many mathematicians attempted to solve this problem, but all failed until Galois developed theory to prove they were all impossible in 19th Century.

this method relies on the famous axiom of Eudoxus-Archimedes (or simply called axiom of Archimedes).

Axiom 6.1.1. Axiom of Archimedes Given two magnitudes a and b , there exists some natural number n , such that $a \leq nb$.

Axiom of Archimedes is fundamental. We introduced Euclid algorithm to compute the greatest common measurement in chapter 2, however, we did not show if this algorithm always terminates. With axiom of Archimedes, we can prove that Euclid algorithm always terminates. Eudoxus stated “Given two different magnitudes, for the larger one, subtract a magnitude larger than its half, then for the remaining, subtract another magnitude larger than its half, repeat this process, there must be some remaining less than the smaller one.” This is the logic behind the method of exhaustion.

By using the method of exhaustion, Eudoxus proved that: the volume of a pyramid is one-third the volume of a prism with the same base and altitude, and the volume of a cone is one-third that of the corresponding cylinder. These statements are summarized as propositions in the book 12 of Euclid’s *Elements*[8].

Archimedes greatly developed the method of exhaustion, and achieved the highest level amazing result. He calculated π , proved the formulas of circular area, surface area and volume of sphere, cone, and even found the method to calculate the area under the parabola curve. He was said to be the god of the mathematics in ancient Greece.

Archimedes (287BC - 212 BC) was a Greek mathematician, physicist, engineer, inventor, and astronomer. He was born in the seaport city of Syracuse, at that time a self-governing colony in Magna Graecia. Archimedes might have studied in Alexandria, Egypt in his youth. During his lifetime, Archimedes made his work known through correspondence with the mathematicians in Alexandria. Although few details of his life are known, he is considered the greatest mathematician of antiquity and one of the greatest of all time. various popular stories about him are widely circulated.

The most widely known anecdote about Archimedes tells of how he uncovered a fraud in the manufacture of a golden crown commissioned by King Hiero II of Syracuse. The king had supplied the pure gold to be used, and Archimedes was asked to determine whether some silver had been substituted by the dishonest goldsmith. Archimedes had to solve the problem without damaging the crown, so he could not melt it down into a regularly shaped body in order to calculate its density. While taking a bath, he noticed that the level of the water in the tub rose as he got in, and realized that this effect could be used to determine the volume of the crown. Archimedes then took to the streets naked, so excited by his discovery that he had forgotten to dress, yelling “Eureka!” (Greek word meaning “I have found [it]!”). The test was conducted successfully, proving that silver had indeed been mixed in. His discovery is the “Archimedes’ Principle” that every middle school student must learn. Eureka was later used to describe the moment when inspiration was found.

In 214BC, the Second Punic War Broke out. Legend has it that Archimedes created a giant parabolic mirror to deflect the powerful Mediterranean sun onto the ship’s sails, setting fire to them. Archimedes also created a huge crane operated hook –the Claw of Archimedes –that was used to lift the enemy ships out of the sea before dropping them to their doom. After two-year-long siege, In 212 BC, the Romans captured Syracuse. The Roman force commander, Marcellus had ordered that Archimedes, the well-known



The Fields Medal carries a portrait of Archimedes.

mathematician should not be killed. Archimedes, who was now around 78 years of age, continued his studies after the breach by the Romans and while at home, his work was disturbed by a Roman soldier. The last words attributed to Archimedes are “Do not disturb my circles!” The soldier killed Archimedes despite orders that Archimedes should not be harmed. 137 years after his death, the Roman orator Cicero described visiting the tomb of Archimedes. It was surmounted by a sphere and a cylinder, which Archimedes had requested be placed on his tomb to represent his mathematical discoveries⁴.

As an example, let us see how Archimedes calculated π with the method of exhaustion around 250BC. Symbol π represents the ratio of a circle’s circumference to its diameter, sometimes it’s referred to as Archimedes’ constant.

As shown in figure 6.10, Archimedes drew two regular polygons inside and outside a circle with diameter of 1. For a side of the inscribed polygon and the corresponding arc, the length of the arc is greater than the side because the straight line is the shortest between two points. Hence the circumference of the circle is greater than the inscribed polygon. Similarly, we can reason that the circumference of the circle is less than the circumscribed polygon. Since the diameter is 1, the circle’s circumference equals to π . Hence the below relation holds:

$$C_i < \pi < C_o$$

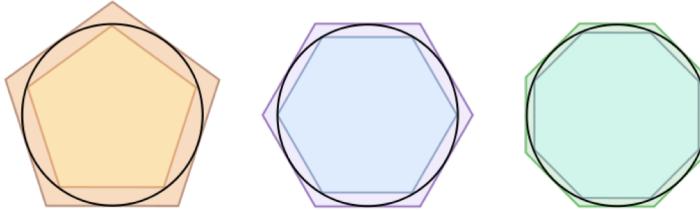


Figure 6.10: π can be estimated by computing the perimeters of circumscribed and inscribed polygons.

Where C_i and C_o are the circumferences of the inscribed and circumscribed polygons respectively. Successively increasing the number of sides approximates the range of π . Archimedes calculated the 96-sided regular polygon, he proved that $\frac{223}{71} < \pi < \frac{22}{7}$ (that is $3.1408 < \pi < 3.1429$). This upper bound of $\frac{22}{7}$ is widely used in western. The Chinese mathematician Zu Chongzhi, around 480AD, calculated that $3.1415926 < \pi < 3.1415927$ and suggested the approximations $\pi \approx \frac{355}{113} = 3.14159292035\dots$ by applying to a 12,288-sided polygon. This value remained the most accurate approximation of π for the next 800 years.

The method of exhaustion, developed in ancient time has limitation. Although rigorous, it demands specific approach for different problems. As a precursor to the methods of calculus, it’s complex. Partially because ancient Greeks rejected irrational numbers, they had to make difference between geometric magnitudes and numbers. Another reason was because they attempted to avoid using infinity and infinitesimal.

Ptolemy, the ancient Greek mathematician, astronomer, and geographer also made great achievement with the method of exhaustion. He developed a geocentric model to calculate the celestial motions. It was almost universally accepted until the scientific

⁴A sphere has $2/3$ the volume and surface area of its circumscribing cylinder including its bases.

revolution. His Planetary Hypotheses presented a physical realization of the universe as a set of nested spheres, in which he used the epicycles of his planetary model to compute the dimensions of the universe. He estimated the Sun was at an average distance of 1,210 Earth radii, while the radius of the sphere of the fixed stars was 20,000 times the radius of the Earth.

After Hellenistic period, the Greek civilization was destroyed by several forces. The Romans conquered Greece, Egypt, and the Near East. In 47BC, the Romans set fire to the Egyptian ships in the harbor of Alexandria; the fire spread and burned the library – the most expensive ancient libraries. The emperor Theodosius (ruled 379 - 396) proscribed the pagan religions and in 392 ordered that their temples be destroyed, including the temple of Serapis in Alexandria, which housed the only remaining sizable collection of Greek works. Thousands of Greek books were burned by the Romans. Many other works written on parchment were expunged to rewrite Christianity works.

The final blow to the Greek civilization was the conquest of Egypt by the uprising Arab empire in 640. The remaining books were destroyed on the ground that as Omar, the conqueror, put it, “Either the books contain what we also have, in which case we don’t have to read them, or they contain the opposite of what we believe, in which case we must not read them.” And so for six months the baths of Alexandria were heated by burning rolls of parchment[4].

When read about such history, it always makes people sad and sigh. The tragedy of burning books in South America, in the Qin Empire, has been performed ever since ancient times. After capture of Egypt, the majority of the scholars migrated to Constantinople, which had become the capital of the Eastern Roman Empire. The Arabs absorbed the Greek works, translated and commented extensively to Greek knowledge. The ‘House of Wisdom’ in Baghdad gradually became the academy centre in the world. After Medieval, Europeans translated the ancient Greek works from Arabic to Latin. Along with the Renaissance in Europe, not only arts and culture, but also mathematics and philosophy recovered and were greatly developed.

German astronomer Johannes Kepler took the next important step after Archimedes atop method of exhaustion. When Nicolaus Copernicus began to think astronomy, the Ptolemaic theory had become somewhat more complicated. To explain the variations in speed and direction of the apparent motions, Ptolemy added epicycles, and other complex geometric tricks in his model. In Copernicus’ time, the theory required a total of 78 circles to describe the motion of Sun, Moon, and the five planets. By moving the Sun to the centre, Copernicus was able to reduce the total number of circles (differents and epicycles) to 34. It was greatly simplified from the geocentric model. Kepler made more remarkable achievement. He inherited valuable observation data from the famous astronomer Tycho Brahe. He spent 8 years to analyze the observed data and false trails. Kepler’s most famous and important results are known today as Kepler’s three laws of planetary motion. According to his first law, Kepler broke with the tradition of two thousand years that circle or sphere must be used to describe celestial motions. It states that each planet moves on an ellipse and that the sun is at one (common) focus of each of these elliptical paths. The other radical step Kepler made was he discovered that the planet does not move at a constant velocity. A line segment joining a planet and the Sun sweeps out equal areas during equal intervals of time. This is his second law. It explains that why a planet sometimes moves fast (close to the Sun) while sometimes moves slowly (far from the Sun). The third law states that, the square of the orbital period of a planet is directly proportional to the cube of the semi-major axis of its orbit. Such complex models required more powerful mathematical tool, the method of exhaustion is not convenient. Kepler then made simplification, and he used the new method on measuring the volume of containers such as wine barrels.

The next important step was made by Descartes and Fermat. Through analytical

geometry, numbers and geometry were bridged, and finally evolved to calculus by Issac Newton and Gottfried Wilhelm Leibniz. Infinitesimal is the central concept in calculus, and the integration involved sum of infinitely many such quantities. As a side word, John Wallis, the important contributor to calculus, introduced ∞ symbol for infinity in 1665.

Although the logic foundation of calculus caused hotly debating, this new tool, representing the modern spirit of the western, broke the waves in its sail in the 18th Century. This was an era of heroes. The Bernoulli family, Euler, and Lagrange greatly developed calculus and infinite series, solved many hard problems in astronomy, mechanics, and fluid that people never imagined before.

6.2 Potential infinity and programming

Mathematicians came back to consider actual infinity when debating about how to make calculus rigorous. Before this topic, let us see how the idea of infinity is realized in programming. Computers can only use limited resources. Numbers are represented in binary forms suitable for computer. There are finite many binary bits, hence the numbers represented in computer are also bounded. A binary number of m bits can represent numbers at maximum of $2^m - 1$, which is 11...1 of length m . The biggest 16 bits number is $2^{16} - 1 = 65535$. For this reason, if the number of elements in a set is also represented in binary, then the set can only contain finite many elements. In early days of programming, arrays were often used to hold multiple elements. To effectively use computer memories, the size of array need be determined before using. For example below statement in C programming language, declares an array that can hold 10 integers:

```
int a[10];
```

There are two different concepts of numbers, ordinal number and cardinal number. In short, ordinal number is used to describe a way to arrange a collection of objects in order, one after another; while cardinal numbers, are used to measure the size of collections. We'll provide the formal definitions for them later. Both ordinals and cardinals are finite in traditional programming. They can't represent infinity directly. It was reasonable in the early days of computer science. The computer devices were very expensive, people never thought to deal with practice problems with infinity. As time goes on, the cost of computation resources keep decreasing. We are not satisfied with the way to predict the size of the collection before using it in programming. New tools, like dynamic array, were developed in some programming environments. They were known as containers, the size can be easily adjusted on-demand. However, even for dynamic containers, the elements are still finite many. It can not exceed the number representation limit. People developed the linked-list, as explained in chapter 1, elements are stored in node, and chained together. The last element points to a special empty node indicating the end. Given such a linked-list, we can start from its head, move to any node in it without need of knowing its cardinal. As far as the storage allows, a linked-list can be arbitrary long. It brings the chance to represent potential infinity.

However, there is eventually a gap between linked-list and potential infinity. We consider natural numbers as potential infinity without terminating or 'end point'. But when use linked-list to represent natural numbers, no matter how long the list is, for instance n , we have to store all numbers from 0 to n in it. It only represents sequence of 0, 1, ..., n , but not the natural numbers, 0, 1, ..., n , ...

To model the potential infinity, people introduced concept of lazy evaluation. Instead of evaluate the value of an expression or variable, this evaluation strategy delays it until the value is needed. For the natural number example, any number n has a successor $n+1$

according to Peano's axiom introduced in chapter 1, and the first natural number is 0. Hence we can define natural numbers as below:

$$N = \text{iterate}(n \mapsto n + 1, 0)$$

Where *iterate* is defined as:

$$\text{iterate}(f, x) = x : \text{iterate}(f, f(x))$$

Let us see the first several steps when generate natural numbers. To make it simple, denote $\text{succ}(n) = n \mapsto n + 1$

$$\begin{aligned} \text{iterate}(\text{succ}, 0) &= 0 : \text{iterate}(\text{succ}, \text{succ}(0)) && \text{definition of } \text{iterate} \\ &= 0 : \text{iterate}(\text{succ}, 1) && \text{succ}(0) = 0 + 1 = 1 \\ &= 0 : 1 : \text{iterate}(\text{succ}, \text{succ}(1)) \\ &= 0 : 1 : \text{iterate}(\text{succ}, 2) \\ &= 0 : 1 : 2 : \text{iterate}(\text{succ}, 3) \\ &= \dots \end{aligned}$$

Without lazy evaluation, this process will repeat endlessly forever. It can not be used to solve practical problems. We must change the link operation to lazy evaluation. One method is to leverage the λ calculus introduced in chapter 2.

$$x : xs = \text{cons}(x, () \mapsto xs)$$

We often call the expression $() \mapsto \text{exp}$ as *delay(exp)*. It builds a function without argument, when evaluates (the function), gives the result *exp*.

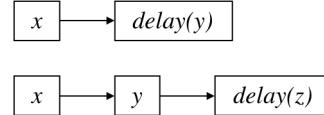


Figure 6.11: The next node pointed is a λ expression. It will create a new node when force evaluation.

Therefore, when link *x* and *xs* together, the value of *xs* won't be evaluated, but *xs* itself will be wrapped in a λ expression, and the evaluation is delayed to future time. With this modification, generation of natural numbers changes to:

$$\begin{aligned} \text{iterate}(\text{succ}, 0) &= 0 : \text{iterate}(\text{succ}, \text{succ}(0)) && \text{definition of } \text{iterate} \\ &= \text{cons}(0, () \mapsto \text{iterate}(\text{succ}, \text{succ}(0))) && \text{lazy link} \end{aligned}$$

The computation stops here. It won't go on. The result is a list. The first element is 0, while the next element is a λ expression. If we want to obtain the successive element, we have to force the list evaluation.

$$\text{next}(\text{cons}(x, e)) = e()$$

When apply *next* to $\text{cons}(0, () \mapsto \text{iterate}(\text{succ}, \text{succ}(0)))$, we obtain:

$$\begin{aligned} &\text{next}(\text{cons}(0, () \mapsto \text{iterate}(\text{succ}, \text{succ}(0)))) \\ &= \text{iterate}(\text{succ}, \text{succ}(0)) && \text{definition of } \text{next} \\ &= \text{iterate}(\text{succ}, 1) && \text{definition of } \text{succ} \\ &= 1 : \text{iterate}(\text{succ}, \text{succ}(1)) && \text{definition of } \text{iterate} \\ &= \text{cons}(1, () \mapsto \text{iterate}(\text{succ}, \text{succ}(1))) && \text{lazy link} \end{aligned}$$

The computation stops again. By repeatedly applying *next* to N , we generate natural numbers one by one. People call this kind of model *stream*, and use it to represent potential infinity. We can easily define a function to fetch the first m natural numbers from this potential infinite stream.

$$\begin{aligned} \text{take } 0 _ &= [] \\ \text{take } n \text{ cons}(x, e) &= \text{cons}(x, \text{take}(n-1, e())) \end{aligned}$$

For example, $\text{take } 8 \ N = [0, 1, 2, 3, 4, 5, 6, 7]$. There are examples in the appendix of this chapter about how to define natural numbers with potential infinity in different programming languages.

Exercise 6.1

1. In chapter 1, we realized Fibonacci numbers by folding. How to define Fibonacci numbers as potential infinity with *iterate*?
2. Define *iterate* by folding.

6.2.1 Coalgebra and infinite stream★

To model the stream of potential infinity, we need the coalgebra concept in category theory introduced in chapter 4. Readers are safe to skip this section in the following 2 pages, and directly read from the next section **Explore the actual infinity**. Let us first revisit coalgebra and F-morphism.

Definition 6.2.1. Let \mathbf{C} be a category, $\mathbf{C} \xrightarrow{\mathbf{F}} \mathbf{C}$ is an endo-functor of category \mathbf{C} . For the object A and morphism α in this category, arrow:

$$A \xrightarrow{\alpha} \mathbf{F}A$$

forms a pair (A, α) . It is called F-coalgebra, where A is the carrier object.

We can treat F-coalgebra as object. When the context is clear, we denote the object as a pair (A, α) . The arrows between such objects are defined as the following:

Definition 6.2.2. F-morphism is the arrow between F-coalgebra objects:

$$(A, \alpha) \longrightarrow (B, \beta)$$

If the arrow $A \xrightarrow{f} B$ between the carrier objects make the below diagram commutes:

$$\begin{array}{ccc} A & \xrightarrow{\alpha} & \mathbf{F}A \\ f \downarrow & & \downarrow \mathbf{F}(f) \\ B & \xrightarrow{\beta} & \mathbf{F}B \end{array}$$

Which means $\beta \circ f = \mathbf{F}(f) \circ \alpha$

F-coalgebra and F-morphism form F-coalgebra category $\mathbf{CoAlg}(\mathbf{F})$. For F-algebra, we care about the initial algebra; symmetrically, for F-coalgebra, we care about the *final coalgebra*. Where the final coalgebra is the final object in F-coalgebra category, denoted as (T, μ) . For any other algebra (A, f) , there exists unique morphism m , such that the below diagram commutes:

$$\begin{array}{ccc}
 & \mathbf{F}(m) & \\
 \mathbf{FT} & \xleftarrow{\quad} & \mathbf{FA} \\
 \mu \uparrow & & \uparrow f \\
 T & \xleftarrow{m} & A
 \end{array}$$

From Lambek theorem, the final coalgebra is the fixed point for functor. The morphism $T \xrightarrow{\mu} \mathbf{FT}$ is an isomorphism, such that \mathbf{FT} is isomorphic to T . The final coalgebra can be used to build infinite data structures.

We use catamorphism to evaluate the initial algebra. Symmetrically, we use anamorphism (prefix ana- means upward) to coevaluate the final coalgebra. For any coalgebra (A, f) , the unique arrow to the final coalgebra (T, μ) can be represented with the anamorphism as $\llbracket f \rrbracket$. The brackets do not look like bananas any more, but like a pair of lenses in optics. They are known as **lens brackets**. As shown in below diagram:

$$\begin{array}{ccc}
 T & \xrightarrow{\mu} & \mathbf{FT} \\
 \uparrow \llbracket f \rrbracket & & \uparrow \mathbf{F} \llbracket f \rrbracket \\
 A & \xrightarrow{f} & \mathbf{FA}
 \end{array}$$

$$m = \llbracket f \rrbracket, \text{ if and only if } \mu \circ m = \mathbf{F}(m) \circ f$$

Let us see how anamorphism builds infinite stream. Anamorphism takes a coalgebra $A \xrightarrow{f} \mathbf{FA}$ and a carrier object A , it generates the fixed point of functor \mathbf{F} , which is $\mathbf{Fix F}$. This fixed point is the final coalgebra. It has the form of infinite stream.

$$\llbracket f \rrbracket = \mathbf{Fix} \circ \mathbf{F} \llbracket f \rrbracket \circ f$$

We can also define the anamorphism as the function that returns the fixed point:

$$\begin{aligned}
 (A \rightarrow \mathbf{FA}) & \xrightarrow{\text{ana}} (A \rightarrow \mathbf{Fix F}) \\
 \text{ana } f & = \text{Fix} \circ \text{fmap} (\text{ana } f) \circ f
 \end{aligned}$$

As a concrete example, functor \mathbf{F} is defined as:

```
data StreamF E A = StreamF E A
```

Its fixed point is:

```
data Stream E = Stream E (Stream E)
```

StreamF E is a normal functor, we intend to give it name of ‘stream’. The coalgebra of this functor is such a function, it transforms a ‘seed’ a of type A to a pair, containing a , and the next seed.

Coalgebra can generates varies of infinite streams. Here are two examples. The first example is Fibonacci numbers. We use $(0, 1)$ as the starting seed. To generate the next seed, we take the second value 1 in the pair as the first value in the new pair, and use $0 + 1$ as the second value in the new pair to form the new seed $(1, 0 + 1)$. Repeat this process, for seed (m, n) , we generate the next seed $(n, m + n)$. Written in coalgebra, we have the following definition:

$$(Int, Int) \xrightarrow{fib} \mathbf{StreamF} \ Int \ (Int, Int)$$

$$fib(m, n) = \mathbf{StreamF} \ m \ (n, m + n)$$

In this definition, the carrier object A is a pair of integers. With coalgebra, we can feed it into anamorphism to build the infinite stream of Fibonacci numbers. For functor $\mathbf{StreamF} E$, the type of the anamorphism is:

$$(A \rightarrow \mathbf{StreamF} E \ A) \xrightarrow{ana} (A \rightarrow \mathbf{Stream} E)$$

We can realize it as:

$$ana \ f = fix \circ f$$

where: $fix \ (StreamF \ e \ a) = Stream \ e \ (ana \ f \ a)$

Apply the anamorphism to coalgebra fib and the start pair $(0, 1)$ gives infinite stream that generates Fibonacci numbers:

$$ana \ fib \ (0, 1)$$

We can define a auxiliary function to take the first n elements from the infinite stream:

```
take 0 _ = []
take n (Stream e s) = e : take (n - 1) s
```

The next example demonstrates how to generate infinite stream of prime numbers with the sieve of Eratosthenes method. The start seed is ‘all’ the natural numbers with 1 being removed: 2, 3, 4, ... From this seed, we remove all the multiples of 2 to obtain the next seed, which is an infinite list start from 3 as 3, 5, 7, ... Next, we remove all the multiples of 3, and repeated this process. This method is defined as below coalgebra:

$$[Int] \xrightarrow{era} \mathbf{StreamF} \ Int \ [Int]$$

$$era(p : ns) = \mathbf{StreamF} \ p \ \{n \mid p \nmid n, n \in ns\}$$

Then feed it to anamorphism, we obtain the infinite stream that generates all prime numbers:

```
primes = ana era [2...]
```

For list particularly, anamorphism is called unfold. Anamorphism and catamorphism are mutually inverse. We can turn the infinite stream back to list through catamorphism.

Exercise 6.2

1. Use the definition of the fixed point in chapter 4, prove $Stream$ is the fixed point of $StreamF$.
2. Define *unfold*.
3. The fundamental theorem of arithmetic states that, any integer greater than 1 can be unique represented as the product of prime numbers. Given a text T , and a string W , does any permutation of W exist in T ? Solve this programming puzzle with the fundamental theorem and the stream of prime numbers.

6.3 Explore the actual infinity

Aristotle's influence was profound. For more than two thousand years, mathematicians and philosophers had been thinking about concept of infinity. Most of them accepted the potential infinity. However, there were discordant views about actual infinity. For a long time, people believed the actual infinity was God, or only God mastered actual infinity. Many attempts to reason the actual infinity led to confusion and contradict result. Let all the natural numbers be actual infinity for example. Because natural numbers are separated by even numbers and odd numbers. They alternate in turns. It's natural to think that all even numbers are half of all natural numbers. However, doubling every natural number gives an even number; and dividing every even number by 2 gives a natural number vise versa. There is one to one correspondence between all the natural numbers and even numbers. Are these two actual infinities same? If not, which one has more? natural numbers or even numbers?

Father of the modern science, Galileo Galilei made a similar paradox in his final scientific work *Two New Sciences* in 1636. Some numbers are squares, while others are not; therefore, all the numbers, including both squares and non-squares, must be more numerous than just the squares. And yet, for every number there is exactly one square, which forms the sequence 1, 4, 9, 16, 25, ... hence, there cannot be more of one than of the other. This paradox is known as Galileo's paradox.

Not only numbers, people found similar paradox in Geometry. As shown in figure 6.12, For two circles with the same centre, every radius connects two points in each circle respectively, hence, there is one to one correspondence between the points in the bigger circle and the smaller circle. It indicates there are same many points in each circle. However, our common sense tells there must be more points in the bigger one.

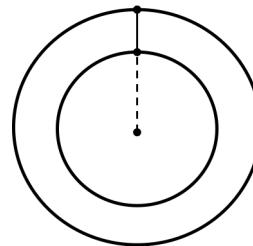
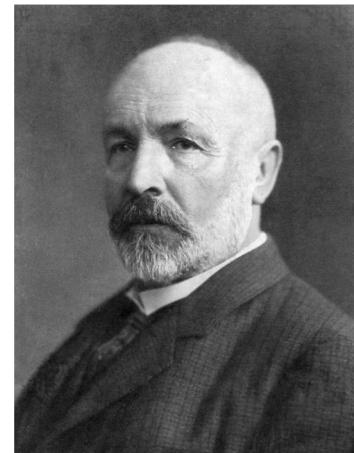


Figure 6.12: Every point in the big circle is corresponding to a point in the small circle.

Because of these paradoxes, people accepted Aristotle's approach to avoid using actual infinity. Galileo concluded that the ideas of less, equal, and greater couldn't apply to infinite sets like natural numbers. People rejected terms of actual infinity, like "all the natural numbers".

About two hundred years after Galileo, German mathematician Cantor led people broke into the Kingdom of infinity. From these paradoxes, Cantor thought the key problem was our common sense assumption, that the whole must be larger than the part. Is it necessarily right? The development of modern science taught us our common view to things could be incomplete or wrong. The theory of relativity challenges our understanding to space and time – the space we are living in is not necessarily the Euclidean space; Quantum mechanics challenges our common sense that the world is causal – randomness rules the quantum world. When think out of the box, it will open up a new world we've never seen, resulting in fundamental development. This was exactly what Cantor did. If we accept the counter-intuitive result, that 'the part can equal to the whole', then the door to the infinity is open. He established the importance of one-to-one correspondence between the members of two sets, and defined (actual) infinite sets.

To compare two sets, Cantor defines if there is one to one correspondence between them, that every element in set M has the unique corresponding element in set N , then the two sets are equinumerous. They have the same size or have the same number of elements⁵, denoting $M \cong N$. For finite sets, it is true obviously; when extends to infinite sets, it means there are same infinite many even numbers as the natural numbers! there are same infinite many square numbers as natural numbers; the points in smaller circle and the bigger circle of the same centre are equinumerous... Cantor's friend, Richard Dedekind even gave such a definition⁶ of infinite set in 1888: if some proper subset of a set is equinumerous to this set, then it is a infinite set.



Georg Cantor, 1845-1918

6.3.1 Paradise of infinite kingdom

Let us have a glance view of the garden in infinite kingdom. David Hilbert told an interesting story in a lecture in 1924 (published in 2013) to help people understanding Cantor's infinite sets. It was popularized through George Gamow's 1947 book *One Two Three... Infinity*.

In this story, there is a Grand hotel with infinite many rooms. It is fully occupied during the hot season. One evening, a tired driver has passed many "No vacany" hotels before reaching to this one. He goes to see if there might nonetheless be a room for him. The clerk, Hilbert said: "No problem, we can make a room for you." He moved the guest in room 1 to room 2, then moved the guest in room 2 to room 3, and moved the guest in room 3 to room 4, ... He moved every one from current room to the next room. That freed up the first room for this new guest.

The story goes on. On the second day, there comes a tour group of infinite many guests. Hilbert said: "No problem, we can make rooms for every one." He moved the new guest who was in room 1 last night to room 2, then moved the guest in room 2 to room 4, and moved the guest in room 3 to room 6, ... He moved every one in room i to room $2i$. Since the hotel has infinite many rooms, after moving, room 2, 4, 6, ... these even number rooms are occupied with the guests accommodated yesterday, while the room 1, 3, 5, ... these odd number rooms are freed up. There are infinite many odd numbers rooms that can accommodate every member in the tour group.

The story does not end. On the third day, there comes infinite many tour groups of infinite many guests each. Can the magic Hilbert's hotel accommodate them all? Before seeing the answer, let us first revisit the story on the first two days.

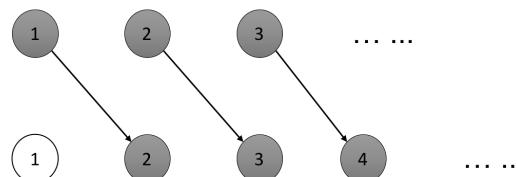


Figure 6.14: First day of Hilbert's Grand hotel

⁵Remind the 'isomorphic' concept introduced in chapter 3.

⁶known as Dedekind infinite set.

As shown in figure 6.14, on the first day, we move every guest to the next room to free up room 1. Essentially, we establish a 1-to-1 correspondence for shadowed circles between the two rows as $n \leftrightarrow n + 1$. It reveals a counterintuitive fact that infinity plus one is infinity again. Hilbert's grand hotel can accommodate not only this one more guest, but also finite many k new guests by repeating this arrangement k times. It means:

$$\infty + 1 = \infty$$

$$\infty + k = \infty$$

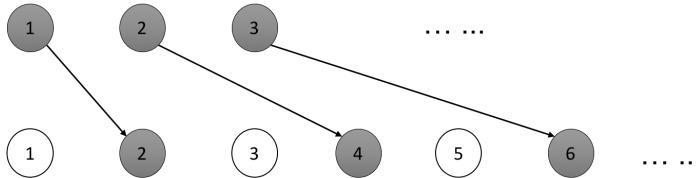


Figure 6.15: Second day of Hilbert's Grand hotel

The solution for the second day is shown in figure 6.15. We essentially establish a 1-to-1 correspondence between natural numbers and even numbers, hence free up infinite many odd number rooms. Then we establish another 1-to-1 correspondence between these empty rooms and the infinite many guests in the tour group, which is exactly the 1-to-1 correspondence between odd numbers and natural numbers. The second day story tells us, infinity plus infinity is infinity again.

$$\infty + \infty = \infty$$

It's natural to ask, although the symbols are same, does the infinity on the left hand equal to the infinity on the right hand? Can we compare between infinities for size? We'll see later, it was exactly this question, led Cantor to study infinity in depth. In Hilbert's Grand hotel story, we can establish 1-to-1 correspondence between all these infinities, hence they are all equinumerous. Such infinity that has 1-to-1 correspondence with natural numbers are called 'countable infinity'.

To solve the problem on the third day of Hilbert's Grand hotel, we need think about how to establish the 1-to-1 correspondence between the infinite many tour groups of infinity many guests and the infinity many rooms. One may come to the idea to arrange the guests in the first tour group to room 1, 2, 3, ... then arrange the guests in the second group to room $\infty + 1, \infty + 2, \dots$. However, this method does not work. We don't know which one are more numerous, the rooms or the guests before the arrangement. Consider how this process executes. The first guest in the second group will never know when the first group finishes accommodating, this guest has no way to determine which room should move to. Compare with the first day story, the new guest could immediately move to room 1 when the original guest moved to room 2, although the whole infinity accommodating process is endless. Same situation happened on the second day. When the original guest in room 1 moved to room 2, the first guest in the tour group could move to the freed up room 1 immediately. Next the original guest in room 2 moved to room 4, and the guest in room 3 moved to room 6, at this time, the second guest in the tour group could move to room 3...

Figure 6.16 gives a numbering solution. For convenient purpose, we label the first guest 0, the second guest 1, the third guest 2, ... We label the guests already lived in the

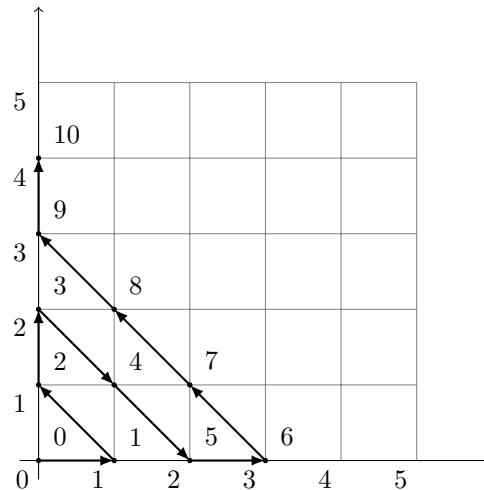


Figure 6.16: One solution to number the infinity of infinity

hotel as group 0, the first tour group 1, the second group 2, ... In this figure, every guest corresponds to a grid point. We also label the hotel rooms from 0.

Now we arrange rooms in this order: assign the guest 0 in group 0 to room 0, the guest 1 in group 0 to room 1, the guest 0 in group 1 to room 2, the guest 0 in group 2 to room 3, the guest 1 in group 1 to room 4, ... Along the zig-zag path, we can assign room one by one, without missing any guests. Hence we establish a 1-to-1 correspondence between every guest in these infinite many groups and the infinity many rooms. Hilbert's Grand hotel surprisingly holds 'two-dimensional' infinity⁷.

Exercise 6.3

1. We establish the 1-to-1 correspondence between the rooms and guests with the numbering scheme shown in 6.16. For guest i in group j , which room number should be assigned? Which guest in which group will live in room k ?
2. For Hilbert's Grand hotel, there are multiple solutions for the problem on the third day. Figure 6.17 is the cover page of the book *Proof without word*. Can you give a numbering scheme based on this figure?

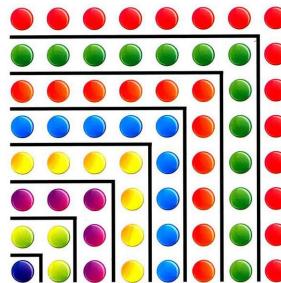


Figure 6.17: From cover page of *Proof without word*

⁷The traditional solution uses the Euclid's theorem, that there are infinite many prime numbers. Empty the odd numbered rooms by sending the guest in room i to room 2^i , then put the first group's guests in rooms 3^n , the second group's guests in rooms 5^n , ... put the k -th group's guests in rooms p^n , where p is the k -th prime number. This solution leaves certain rooms empty, specifically, all odd numbers that are not prime powers, such as 15 or 847, will no longer be occupied.

6.3.2 One-to-one correspondence and infinite set

From Hilbert's Grand hotel story, we see the importance of the 1-to-1 correspondence in studying infinity. If there exists 1-to-1 correspondence between two sets, then they have the same cardinality. We can further use 1-to-1 correspondence to classify sets. As explained in chapter 3, for two sets A and B , we establish a map $A \xrightarrow{f} B$, such that every element x in A is corresponding to an element y in B through $x \mapsto y = f(x)$. For sets, we call f function. y is the image of x , and x is preimage. If there is exactly one preimage, such map is called *injective function*; if every element y in B has a preimage, then the map is called *surjective* or onto. If a map is both injective and surjective, it is called *bijective* or one-to-one correspondence. Figure 6.18 illustrates a bijection between two sets.

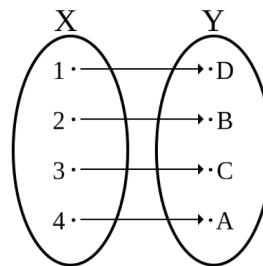


Figure 6.18: 1-to-1 correspondence between two sets.

Hilbert's Grand hotel gives surprising, counterintuitive properties of infinity. The set of natural numbers is equinumerous with its proper subsets like even and odd numbers. And the one dimensional natural number n and two dimensional pair (m, n) are also equinumerous. Starting from natural numbers, Cantor extended a series of infinite sets through 1-to-1 correspondence. for example:

1. **Integers.** We can establish the following 1-to-1 correspondence:

$$\begin{array}{ccccccc}
 0 & 1 & -1 & 2 & -2 & \dots & n & -n & \dots \\
 \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & & \uparrow & \uparrow \\
 0 & 1 & 2 & 3 & 4 & \dots & 2n-1 & 2n & \dots
 \end{array}$$

Hence extend natural numbers to integers. From another view point, we essentially correspond odd numbers to positive integers, and even numbers to negative integers and zero. It tells that the integers and natural numbers are equinumerous.

2. **Rationals.** A rational number can be expressed as the quotient or fraction p/q of two integers, a numerator p and a non-zero denominator q . On the third day of Hilbert's Grand hotel story, we established 1-to-1 correspondence between a pair (p, q) and a natural number. We can adjust it a bit for rational number p/q . Put negative numbers aside for now, we skip whenever the second number q is zero, or p/q is reducible fraction (with common divisor). Then we re-use the method for integers, to cover the negative rational numbers as well. In this way, we construct rational numbers from natural numbers. Below table illustrates how the first several natural numbers correspond to rational numbers:

$$\begin{array}{ccccccccccc}
 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & \dots \\
 \uparrow & \uparrow \\
 0 & 1 & \frac{1}{2} & -\frac{1}{2} & -1 & -2 & -\frac{2}{3} & -\frac{1}{3} & \frac{1}{3} & \dots
 \end{array}$$

It tells us natural numbers and rational numbers are equinumerous.

3. **Algebraic numbers.** An algebraic number is a root of a non-zero polynomial in one variable with rational coefficients. Equivalently, by clearing denominators, we can loose the coefficients to integers. For example, $\sqrt{2}$ and $1 \pm \sqrt{3}i$ are algebraic numbers, while π and e are not.

Given an algebraic equation

$$a_0x^n + a_1x^{n-1} + \dots + a_n = 0$$

Where a_0, a_1, \dots, a_n are integers, and $a_0 \neq 0$. All its roots are algebraic numbers. Define a positive integer:

$$h = n + |a_0| + |a_1| + \dots + |a_n|$$

Which is the sum of the degree and coefficients of the equation. We name it as the height h of the equation. For any algebraic equation, h is a unique natural number. On the other hand, given a height h , there could be multiple equations. For example, the height of equations $x - 3 = 0$, $x^3 + 1 = 0$, $x^3 - 1 = 0$, $x^2 + x + 1 = 0$, $x^2 - x + 1 = 0$ are all 4. However, there are finite many equations for every h . Therefore, we can enumerate all algebraic equations: first list all equations of height $h = 1$, then list the equations of height $h = 2, \dots$ and repeated this process. The equations of the same heights can be list in arbitrary order. According to the fundamental theorem of algebra proved by Gauss, the number of roots equals to the equation degree n . Taking the multiplicity into consideration, the number of different roots is not greater than n . Hence there are finite many roots for equation of height h . Now we can enumerate all algebraic numbers.

First, we enumerate all roots for equations of height $h = 1$ (only one such equation $x = 0$), which is 0; then enumerate all roots for equations of height 2. Because different equations may have some same roots, we need skip any root if it has been enumerated before. In this way, we establish the 1-to-1 correspondence between algebraic numbers and natural numbers. Hence they are equinumerous. In other words, we can extend to all algebraic numbers from natural numbers.

We'll meet a problem next. Can we extend natural numbers to real numbers? not only for normal irrationals, but also cover transcendental numbers like π and e ? Cantor and Dedekind made great breakthrough when studied this problem.

Cantor and Dedekind

Georg Cantor was a German mathematician, He created set theory, which has became a fundamental theory in mathematics. Cantor was born in 1845 in the western merchant colony of Saint Petersburg, Russia, and brought up in the city until he was eleven. His father was a successful merchant, and member of the Saint Petersburg stock exchange; his mother came from a family well-known of music. When his father became ill, the family moved to Germany Frankfurt in 1856. Cantor demonstrated exceptional skill in mathematics in school. But his father wanted Cantor to became "a shining star in the engineering firmament." However, at the age of 17, Cantor had sought his father's permission to study mathematics at university and he was overjoyed when eventually his father consented[8].

He entered the Polytechnic of Zürich in 1862, then moved to the University of Berlin in 1863. He attended lectures by Leopold Kronecker, Karl Weierstrass and Ernst Kummer. He spent the summer of 1866 at the University of Göttingen. Cantor was a good student, and he received his doctorate degree in 1867 with the dissertation on number theory. Cantor later took up a position at the University of Halle, where he spent his entire career.

At that time, many mathematicians were trying to rebuild the rigorous logical foundation of analysis led by Weierstrass. Cantor was also influenced by this movement. He soon realized the importance to study real numbers as the basis of calculus, which became the beginning of set theory research.

In 1872, he met and began a friendship with the young mathematician Richard Dedekind while on holiday in Switzerland. Even in Cantor's honeymoon in Harz mountains, Cantor spent much time in mathematical discussions with Dedekind. They started a long time correspondence between each other.

In 1874, Cantor published his first revolutionary paper about set theory at the age of 29. It marked as the beginning of set theory as a branch of mathematics. With the extraordinary ingenuity, Cantor established set theory in the following ten years almost alone, leading the revolution of infinity in mathematics. However, he was not well recognized during his most creative period. He desired, but was not able to obtain a professor chair at the University of Berlin. He spent his career at the University of Halle, which was a infamous university with a meager salary. Cantor's theory was originally regarded as so counterintuitive – even shocking. People found paradoxes hidden in infinity sets (we'll introduce Russell's paradox in next chapter). Cantor's work encountered resistance from mathematical contemporaries. Among them, some were famous mathematicians, including his teacher, the leading mathematician in Berlin, Leopold Kronecker. He had a famous saying: "God made the integers, all else is the work of man." He objected to Cantor's theory about infinity and transfinite numbers, said it was not mathematics but mysticism. Mathematics was headed for the madhouse under Cantor. Henri Poincaré, the famous French mathematician, known as "The Last Universalist", referred to Cantor's work as a "disease" from which mathematics would eventually be cured. Poincaré said, "There is no actual infinite; the Cantorians have forgotten this, and that is why they have fallen into contradiction." Later Hermann Weyl, the great German mathematician criticized Cantor's hierarchy of infinities as "fog on fog." Hermann Schwartz was originally a friend of Cantor, but he stopped the correspondence with Cantor as opposition to Cantor's ideas continued to grow. Mathematicians split to schools of empiricism, intuitionism, and constructivism in different ways, and fell into the controversy about the foundations of mathematics against Cantor.

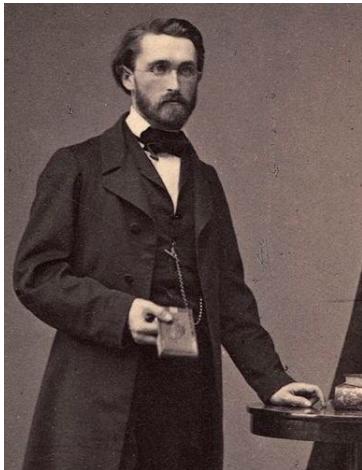
The tragedy was not the set theory, but Cantor went to the madhouse. Cantor suffered his first known bout of depression in May 1884. During the rest of his life, the depression recurred several times with different intensity, driving him from the community into the mental hospital refuge. In 1904 he was agitated by a paper presented by Julius König at the Third International Congress of Mathematicians. The paper was read in front of his daughters and colleagues. Cantor was shaken, and sent to hospital again. Cantor retired in 1913, living in poverty. In June 1917, he entered the sanatorium for the last time and continually wrote to his wife asking to be allowed to go home. Cantor died on January 6, 1918, in the sanatorium where he had spent the last year of his life.

Cantor's set theory was publicly acknowledged and praised at the first International Congress of Mathematicians, held in Zurich in 1897. Adolf Hurwitz (1859 - 1919) openly expressed his great admiration of Cantor and proclaimed him as one by whom the theory of functions has been enriched. Jacques Hadamard expressed his opinion that the notions of the theory of sets were known and indispensable instruments. Over time, people



Cantor, around 1870

gradually realized the importance of set theory. David Hilbert praised Cantor's work as "the finest product of mathematical genius and one of the supreme achievements of purely intellectual human activity." The Continuum hypothesis, introduced by Cantor, was presented by Hilbert as the first of his twenty-three open problems in his address at the 1900 International Congress of Mathematicians in Paris. When Brouwer, the founder of intuitionism criticizing the paradoxes in set theory, Hilbert defended it by declaring, "No one shall expel us from the paradise that Cantor has created."



Richard Dedekind, 1831-1916

Richard Dedekind was a German mathematician. He was born on October 6, 1831 in Brunswick Germany, where is the hometown of Gauss. His father was a professor, his mother was a daughter of a professor at the same university in Brunswick. When Dedekind was in school, mathematics was not his main interest. He studied science, in particular physics and chemistry. However, they became less than satisfactory to Dedekind with what he considered an imprecise logical structure and his attention turned towards mathematics. He entered the University of Göttingen in the spring of 1850 with a solid grounding in mathematics. He learned number theory from M A Stern, and physics from Wilhelm Weber. Gauss was still teaching, although mostly at an elementary level, and Dedekind became his last student. Dedekind did his doctoral work in four semesters under Gauss's supervision and submitted a thesis on the theory of Eulerian

integrals. He received his doctorate from Göttingen in 1852.

After that, Dedekind went to Berlin for two years of study, where he and Bernhard Riemann were contemporaries; they were both awarded the habilitation degrees in 1854. Dedekind was then qualified as a university teacher and he began teaching at Göttingen giving courses on probability and geometry. He studied for a while with Dirichlet, and they became good friends. About this time, Dedekind studied the work of Galois and he was the first to lecture on Galois theory. He also became one of the first people to understand the importance of the notion of groups for algebra and arithmetic.

Dedekind was humble. Many of his achievements were unknown to the people at the time. For example, after Dirichlet's death, Dedekind wrote and published the famous book *Lectures on Number Theory* based on his notes from Dirichlet's lectures. Dedekind was so modest that he published the book under Dirichlet's name, even after adding many additional results of his own in later editions. Unfortunately, Dedekind's modesty hurt his career; he failed to get tenure at Göttingen and ended up on the faculty of a minor technical university([10] pp.140). – Institute of Technology Brunswick in his hometown.

Dedekind remained in Brunswick for the rest of his life, retiring on April 1, 1894. He lived his life as a professor in Brunswick. "In close association with his brother and sister, ignoring all possibilities of change or attainment of a higher sphere of activity. The small, familiar world in which he lived completely satisfied his demands: ... there he found sufficient leisure and freedom for scientific work in basic mathematical research. He did not feel pressed to have a more marked effect in the outside world: such confirmation of himself was unnecessary."

Dedekind made a number of highly significant contributions to mathematics and his work would change the style of mathematics into what is familiar to us today. While teaching calculus for the first time at Brunswick, Dedekind developed the notion now known as a Dedekind cut, now a standard definition of the real numbers. As well as his

analysis of the nature of number, his work on mathematical induction, including the definition of finite and infinite sets, and his work in number theory, particularly in algebraic number fields, is of major importance. He introduced the notion of an ideal which is fundamental to ring theory (later introduced and extended by Hilbert and Emmy Noether). He also proposed an axiomatic foundation for the natural numbers, whose primitive notions were the number one and the successor function. The next year, Giuseppe Peano, citing Dedekind, formulated an equivalent but simpler set of axioms.

Dedekind died on February 12, 1916. About his death, there was an interesting story. One day, Dedekind discovered in a “Biography of Mathematicians”, that wrote: Dedekind died on September 4, 1897. To correct this error, he wrote a letter to the editor of the biography: “According to my diary, I was very healthy on this day and talking with my lunch guest, dear friend Cantor, some interesting things, very enjoyable.” [8]

Even today, there are still different views regarding to Cantor and Dedekind’s work. Dieudonné still considered Dedekind’s work caused unnecessary confusions in 1980s. Not to mention the fierce divisions and debates at the beginning of the 20th Century. Most biographies and comments we see today are often too critical to Kronecker, Brouwer, and the mathematical philosophy of intuitionism they represented. They sympathize with Cantor, and enthusiastically praise the revolution of infinite sets and transfinite numbers. We recommend the rational readers have your own thoughts, but not be completely influenced by one-sided view or the other. Kronecker had a strong belief in mathematical philosophy. He emphasized that mathematics should deal only with finite numbers and with a finite number of operations. He was the first to doubt the significance of non-constructive existence proofs. We should not think that Kronecker’s views of mathematics were totally eccentric. Although it was true that most mathematicians of his day would not agree with those views, and indeed most mathematicians today would not agree with them, they were not put aside. Kronecker’s ideas were further developed by Poincaré and Brouwer, who placed particular emphasis upon intuition. Intuitionism stresses that mathematics has priority over logic, the objects of mathematics are constructed and operated upon in the mind by the mathematician, and it is impossible to define the properties of mathematical objects simply by establishing a number of axioms. Poincaré in his popular book *Science and Hypothesis* stated that convention plays an important role in physics. His view came to be known as “conventionalism”. He also believed that the geometry of physical space is conventional. His idea inspired Einstein when developed his theory of relativity.

Fibonacci numbers and Hamming numbers

Some programming environments support lazy evaluation by default. With them, we can perform complex computation directly on infinite streams. Below is a definition of natural numbers.

$$N = 0 : map(succ, N)$$

In this definition, N is a infinite set of natural numbers. The first number is zero, from the second one, every number is the successor of the previous natural number, as described in the following table:

	$N:$	0	1	2	...
	$map(succ, N):$	$succ(0)$	$succ(1)$	$succ(2)$...
$0 : map(succ, N):$	0	1	2	3	...

Based on this idea, below example code firstly defines the infinite set of natural numbers, then takes the first 10 numbers:

```
nat = 0 : map (+1) nat
take 10 nat
[0,1,2,3,4,5,6,7,8,9]
```

Similarly, we can define Fibonacci numbers as an infinite set. Let F be the set of all Fibonacci numbers. The first element is 0, the second is 1, every Fibonacci number after them are the sum of the previous two. We can make a table with the same method as we do for natural numbers:

	$F:$	0	1	1	2	3	5	8	...
	$F':$	1	1	2	3	5	8	13	...
0	1	1	2	3	5	8	13	21	...

Where the first row lists all Fibonacci numbers; the second row removes the first one, and lists the rest. We can also consider it is the result by shifting the first row to left by one cell; in the third row, every cell is the sum of the above two numbers in the same column. It also lists all Fibonacci numbers except for the first two: 0 and 1. We can prepend them to the left of the third row to obtain the definition of infinite Fibonacci numbers:

$$F = \{0, 1\} \cup \{x + y \mid x \in F, y \in F'\}$$

Here is the corresponding example code:

```
fib = 0 : 1 : zipWith (+) fib (tail fib)
```

For another example, in mathematics, regular numbers are defined as those numbers whose only prime divisors are 2, 3, or 5. In computer science, regular numbers are often called Hamming numbers, after Richard Hamming (American mathematician, ACM Turing award receiver, 1915-1998), who proposed the problem of finding computer algorithms for generating these numbers in ascending order. Here are the first several Hamming numbers.

1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 15, 16, 18, 20, 24, 25, 27, 30, 32, 36, 40, 45, 48, 50, 54, 60, ...

It's not trivial to write a program to generate Hamming numbers. While there exists an intuitive and efficient method by using infinite stream. Let H be the infinite set of Hamming numbers. The first Hamming number is 1. For every number x in H , $2x$ is still a Hamming number, define $H_2 = \{2x \mid x \in H\}$. It is also true for factor 3 and 5. Let us define $H_3 = \{3x \mid x \in H\}$ and $H_5 = \{5x \mid x \in H\}$ respectively. If we merge these three sets, H_2 , H_3 , and H_5 together, remove those duplicated numbers, and prepend 1, then we obtain the set of Hamming numbers again.

$$\begin{aligned} H &= \{1\} \cup H_2 \cup H_3 \cup H_5 \\ &= \{1\} \cup \{2x \mid x \in H\} \cup \{3x \mid x \in H\} \cup \{5x \mid x \in H\} \end{aligned}$$

In programming, \cup means merge two series $X = \{x_1, x_2, \dots\}$ and $Y = \{y_1, y_2, \dots\}$ in order, and drop the duplicated elements.

$$X \cup Y = \begin{cases} x_1 < y_1 : & \{x_1, X' \cup Y\} \\ x_1 = y_1 : & \{x_1, X' \cup Y'\} \\ x_1 > y_1 : & \{y_1, X \cup Y'\} \end{cases}$$

Here is an example code that summarizes above definition, and returns the 1,000,000th Hamming number.

6.3.3 Countable and uncountable infinity

From natural numbers, we've seen how to construct integers, rationals, and the algebraic numbers (including part of irrational numbers like radicals). We are able to establish 1-to-1 correspondence between all of them and natural numbers, hence they are all equinumerous. A set is called *countable* infinite set if it has the same cardinality as natural numbers. Are all infinite set countable? Are there any larger infinities? On November 29, 1873, Cantor wrote to his friend Dedekind a letter([2], pp. 198).

May I ask you a question, which has a certain theoretical interest for me. but which I can't answer; may be you can answer it and would be so kind as to write to me about it. It goes as follows: take the set of all natural numbers n and denote it N . Further, consider, say, the set of all positive real numbers x and denote it R . Then the question is simply this: can N be paired with R in such a way that to every individual of one set corresponds to one and only one individual of the other? At first glance, one says to oneself, "No, this is impossible, for N consists of discrete parts and R is a continuum." But nothing is proved by this objection. And much as I too feel that N and R do not permit such a pairing, I still cannot find the reason. And it is this reason that bothers me; maybe it is very simple.

One week after on December 7, Cantor wrote again to Dedekind.

Recently I had time to follow up a little more fully the conjecture which I mentioned to you; only today I believe I have finished the matter. Should I have been deceived, I would not find a more lenient judge than you. I thus take the liberty of submitting to your judgement what I have written, in all the incompleteness of a first draft.

Cantor proved it impossible to establish 1-to-1 correspondence between natural numbers and real numbers, hence real numbers are uncountable. December 7, 1873 was the day that set theory born. Cantor had given two proofs. The second one is the popular Cantor's *diagonal argument*.

Cantor used reduction to absurdity method in his proof. Suppose the real numbers in open interval $(0, 1)$ are countable. There exists 1-to-1 correspondence to natural numbers. Then we can list all real numbers in this interval as a sequence $a_0, a_1, a_2, \dots, a_n, \dots$ in decimals. For any irrational number, its decimal format is endless non-repeating; For rational number, its decimal format can be infinitely repeating finite sequence of digits, for example $\frac{1}{3} = 0.333\dots$; for decimals with finite digits, we can append infinite many zeros, for example $\frac{1}{2} = 0.5000\dots$ All the real numbers in interval $(0, 1)$ can be listed as below:

$$\begin{aligned}
a_0 &= 0.a_{00}a_{01}a_{02}a_{03}\dots \\
a_1 &= 0.a_{10}a_{11}a_{12}a_{13}\dots \\
a_2 &= 0.a_{20}a_{21}a_{22}a_{23}\dots \\
a_3 &= 0.a_{30}a_{31}a_{32}a_{33}\dots \\
&\dots \\
a_n &= 0.a_{n0}a_{n1}a_{n2}a_{n3}\dots \\
&\dots
\end{aligned}$$

Here is an important note: a_0, a_1, a_2, \dots are not necessarily ordered from small to big. Next we construct a number $b = 0.b_0b_1b_2b_3\dots b_n\dots$, where the n -th digit $b_n \neq a_{nn}$. To achieve this, we can simply make a rule, if $a_{nn} \neq 5$, then let $b_n = 5$, else let $b_n = 6$.

$$b_n = \begin{cases} 5 & : a_{nn} \neq 5 \\ 6 & : a_{nn} = 5 \end{cases}$$

The constructed number b must not equal to any number we list above. This is because at least the n -th digit is different. That the diagonal digits are different. We highlight them in bold font in the table.

$$\begin{aligned}
a_0 &= 0.\mathbf{a_{00}}a_{01}a_{02}a_{03}\dots \\
a_1 &= 0.a_{10}\mathbf{a_{11}}a_{12}a_{13}\dots \\
a_2 &= 0.a_{20}a_{21}\mathbf{a_{22}}a_{23}\dots \\
a_3 &= 0.a_{30}a_{31}a_{32}\mathbf{a_{33}}\dots \\
&\dots \\
a_n &= 0.a_{n0}a_{n1}a_{n2}a_{n3}\dots\mathbf{a_{nn}}\dots \\
&\dots
\end{aligned}$$

Because we assume all numbers in interval $(0, 1)$ are enumerated without one missing, b is obviously in this interval, but it does not equal to any a_i . The 1-to-1 correspondence missed b , hence lead to contradiction. As the result, we are not able to establish 1-to-1 correspondence between real numbers and natural numbers. This proof is called Cantor's diagonal argument.

One may argue why can't add b to the list of a_0, a_1, a_2, \dots ? Suppose after adding b to the list, its position is the m -th number, we can construct another new number c , where its m -th digit does not equal to b_m . Hence we get another number not being included.

This proof is simple and easy. It tells a surprising result: the set of real numbers in interval $(0, 1)$ is uncountable! It is the first infinite set that people found more numerous than natural numbers⁸. As the next step, we establish a 1-to-1 correspondence: $y = \pi x - \frac{\pi}{2}$. It sends every real number in $(0, 1)$ to interval $(-\frac{\pi}{2}, \frac{\pi}{2})$ without any missing. We immediately conclude that the real numbers in this new interval are uncountable. As the final attack, we establish another 1-to-1 correspondence: $y = \tan(x)$. It sends every real number between $-\frac{\pi}{2}$ and $\frac{\pi}{2}$ to the infinite set of all real numbers without any missing⁹.

⁸Courant and Robbins give another intuitive geometric proof in his popular book *What is mathematics*. Suppose the points in the unit line segment $(0, 1)$ can be enumerated as a_1, a_2, a_3, \dots . We cover point a_1 with an interval of length $1/10$, cover point a_2 with an interval of length $1/100$, ... cover point a_n with an interval of length $1/10^n$, and so on. Then the unit line segment $(0, 1)$ will be completely covered (there can be overlaps) by these sub-intervals with lengths $1/10, 1/100, 1/1000, \dots$. However, the total length of these sub-intervals, which is the sum of a geometric series, is $1/10 + 1/100 + 1/1000 + \dots = 1/9 < 1$. It is impossible to cover the line segment of length 1 by an interval of total length $1/9$, hence our assumption cannot hold, the points in the line segment are uncountable[62].

⁹There is another geometric method to establish the 1-to-1 correspondence between the unit line segment to all real numbers. We bend the line segment to a semicircle of length 1, then draw an infinite line L outside the circle. From any point P in L , connect it with the centre of the circle, it must intersect with the arc at a point Q , as in figure 6.21.

With this proof, Cantor made his most important result: The set of real numbers is not countable. It is a higher level of infinity than countable sets. Cantor named it uncountable set, denoted as C .

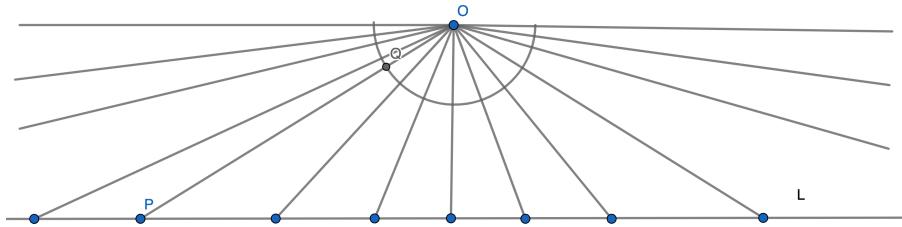


Figure 6.21: map a semicircle of length 1 to all real numbers.

It reminds us the point in the line. In Euclid's *Elements*, a point is defined as "A point is that which has no part.", and line is consist of points. According to Hippasus's finding, there are irrational numbers in the line. In other words, rationals can't fully cover a line, while real numbers can. We'll give Dedekind cut in the next section to define a real number rigorously. The above proof tells us, the points in the unit line segment, the points in a line segment of any length, and the points in an infinite long line, which is the number axis, are all equinumerous. They are all uncountable sets. So as the points in circles of the same centre.

It is also surprised when compare the rationals and irrationals. Given any two rational numbers, there are infinite many irrational numbers between them; given any two irrational numbers, there are also infinite many rational numbers between them. It seems they are equinumerous. While according to Cantor's finding, rational numbers are countable, but irrational numbers are uncountable. Irrationals are more numerous than rationals. Further, we've shown that algebraic numbers are countable, but the transcendental numbers like π and e , are uncountable, they are more numerous than algebraic numbers.

On the third day in Hilbert's Grand hotel story, we established 1-to-1 correspondence between one-dimension countable infinite numbers and two-dimension infinite grids, thus proved rational numbers are countable. Consider the points of real numbers in one-dimension line segment, and points in two-dimension plane, which are more numerous? or equinumerous? Cantor raised this question in a letter to Dedekind in January, 1874. He seemed sure the latter, the two-dimension square had more numerous points than one-dimension line segment. But was not able to prove it. Four years after that, Cantor surprised to find he was wrong. He managed to figure out an interesting 1-to-1 correspondence. He sent the proof to Dedekind, asking for review in June, 1877. In that letter, Cantor said "I see it, but I don't believe it!", the famous statement we quoted at the beginning of this chapter.

Let us see how this 1-to-1 correspondence makes the world in a piece of sand. We are facing two infinite set of points. One is a unit square:

$$E = \{(x, y) | 0 < x < 1, 0 < y < 1\}$$

The other is a unit line segment $(0, 1)$. Take an arbitrary point (x, y) in the unit square, represent both x and y in decimals (for finite decimal, like 0.5, write it in 0.4999... refer to the exercise of this section). Then group the fractional part after the decimal point, every group ends at the first none zero digits, for example:

$$\begin{array}{ccccccc} x & = & 0.3 & 02 & 4 & 005 & 6 \dots \\ y & = & 0.01 & 7 & 06 & 8 & 04 \dots \end{array}$$

Next, we construct a number $z = 0.3\ 01\ 02\ 7\ 4\ 06\ 005\ 8\ 6\ 04\ \dots$ by taking the group of digits from x and y in turns. For this example, first write down 0 and decimal point, then take the first group from x , which is 3, then take the first group from y , which is 01, next take the second group from x , which is 02, next take the second group from y , which is 7, ... number z definitely belongs to the unit line segment. For every two different points in the square, their decimals of x and y must have different digits. Hence the corresponding z are different. It means $(x, y) \mapsto z$ is an injection. On the other hand, for any point z in the unit line segment, we can group the fractional part, then append all the odd groups after 0 and decimal point to form x , and use all the even groups to form y . Pair (x, y) is a point in the unit square. It means $(x, y) \mapsto z$ is also a surjection, hence a bijection (1-to-1 correspondence). We prove that the points in the unit line and square have the same cardinality. Both are uncountable.

Similarly, we can next prove that, not only line and plan have equinumerous points, but also they are equinumerous as the points in the three-dimension space, and even equinumerous as the points in n -dimension space.

Before Cantor, there were only finite sets and “the infinite”, which was considered a topic for philosophical, rather than mathematical, discussion. It was Cantor, that first time told us, there exist infinite sets of different sizes. Cantor did not stopped after differentiating the countable and uncountable infinities. He went on considering if there exist more numerous infinities. Along the ‘infinity of infinity’ path, could we reach to the end point? Before answering these question, let us first see how Dedekind define real numbers with his genius idea.

Exercise 6.4

1. Let $x = 0.9999\dots$, then $10x = 9.9999\dots$, subtract them gives $10x - x = 9$. Solving this equation gives $x = 1$. Hence $1 = 0.9999\dots$ Is this proof correct?

6.3.4 Dedekind cut

In order to make the foundation of calculus rigorous, mathematicians in the 19th Century went back to inspect the confusion concepts, like infinitesimal and infinite series, which were developed and used by Newton, Leibniz, Jacobi, and Euler. Through the work of Cauchy, Weierstrass and so on, the standard of rigour, including limit and convergence were setup. However, there was still a critical problem remaining, the concept of real numbers. The whole calculus is built on top of the continuity of real numbers, while it was lack of a satisfied definition of real number. People thought rationals could present line, but later found there were ‘gaps’ between rational numbers. It was not completeness or continuous. While we demand line be completeness, continuous, without any gaps. What is the exactly meaning of continuity of line?

When Dedekind was thinking how to teach differential and integral calculus, the idea of Dedekind cut came to him on November 24, 1858. He kept developing this idea, and published the result in 1872. Dedekind found although rationals were dense – for any two rational numbers, no matter how they close to each other, there are other rational numbers in between – they were not continuous. Consider a continuous number line, let us use an infinitely thin knife, the knife of thought, to cut the line into two parts([8] pp. 196).

Because the line is continuous without any gaps, no matter how thin the knife, it must cut at a point, but not pass through a gap. (for the line of rationals, but not real numbers, then the knife may cut at a point, or may cut through a gap between two rational numbers, for example, cutting at position $\sqrt{2}$.) Suppose cut at point A , then A is either on the left, or on the right. It can not be on the both sides, or not be on

neither side. This point can not be divided or disappeared. In other words, since line is continuous, wherever it is cut into two parts, one must have an end point, while the other not.

Dedekind defined a cut (A_1, A_2) , A_1 is called ‘closed downwards’, and A_2 is ‘closed upwards’. Where all numbers in downwards A_1 is less than every number in upwards A_2 . Such that A_1 represents the left half line of the cut, and A_2 represents the right half. For any such a cut, either A_1 has a greatest number, or A_2 has a smallest number. There must be one and only one case.

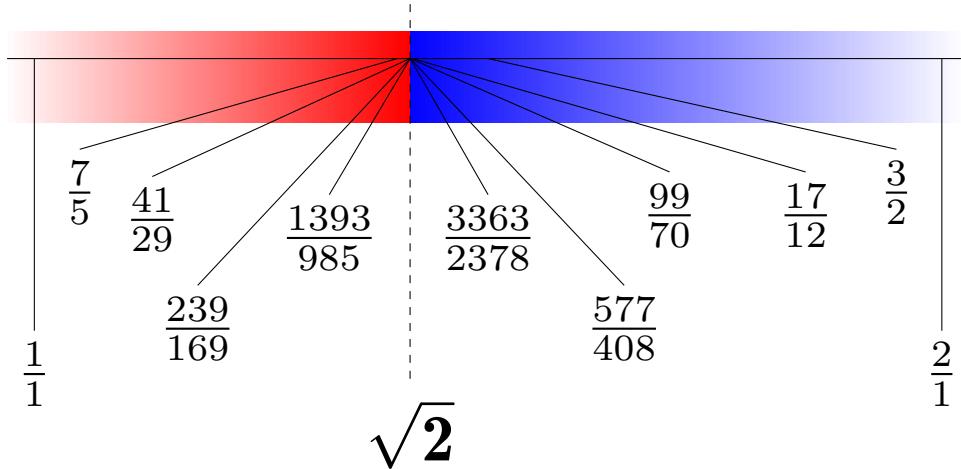


Figure 6.22: Define $\sqrt{2}$ with Dedekind cut

When apply Dedekind cut to all rational numbers, we can find that rationals are not continuous. For example, A_1 contains all rational numbers less than or equal to 2, and A_2 contains all rational numbers greater than 2, then this cut defines rational number 2. However for the negative example: the downwards A_1 contains all rational numbers that are negative, and the non-negative ones, but with their square less than or equal to 2; The upwards A_2 contains the rest rational numbers. We can find in this cut, there is no greatest number in the downwards, while there is no smallest number in the upwards too. It means there is a gap in rational numbers. When cut at this point, the knife will pass through. This cut actually defines a new number $\sqrt{2}$, and it is not a rational number, as shown in figure 6.22.

Dedekind came to the idea that every real number r divides the rational numbers into two subsets, namely those greater than r and those less than r . Dedekind’s brilliant idea was to represent the real numbers by such divisions of the rationals. Every cut of a rationals defines a real number. The cut through a gap (no greatest in A_1 , and no smallest in A_2) is an irrational number; the cut at a point (A_1 has the greatest, or A_2 has the smallest) is a rational number. And real numbers contain both rationals and irrationals. Hence Dedekind cut defines real numbers. Every point in the number line is a real number. It also gives the foundation of the continuity of real numbers.

From Hippasus found the irrational number, till Dedekind finally defined real numbers, It takes people two thousands years¹⁰. In the method of Dedekind cut, we always divide numbers into two finalized infinite parts, which are two infinite sets. This is the development and practice of actual infinity concept.

¹⁰In the same year of 1872, Weierstrass defined irrational numbers as limits of convergent series; Cantor also defined irrational numbers as convergent sequences of rational numbers. The theory of real numbers hence were established through these different paths.

6.3.5 Transfinite numbers and continuum hypothesis

On the way to find more numerous infinity, Cantor firstly considered power set. For a given set A , power set is the set of all possible subsets of A . For example $A = \{a, b\}$, then its power set contains $\{\emptyset, \{a\}, \{b\}, \{a, b\}\}$, total four elements. For a set of 3 elements, its power set contains 8 subsets. Generally, for a set of n elements, because every element can be select or skip when building a subset, the size of its power set is 2^n . It's obvious that power set has a greater cardinality than the original finite set.

Cantor proved in 1891, that even for any infinite set, the power set has a strict greater cardinality than the original set. This result is called *Cantor's theorem* nowadays. The proof is not hard, we put it in the appendix of this chapter. This theorem is the key to open the door to infinite of infinite world. Cantor introduced notation \aleph_0 for the cardinality of countable infinite set, like natural numbers. \aleph is the first Hebrew letter. The cardinality of the power set for countable infinite set is 2^{\aleph_0} . According to Cantor's theorem, $\aleph_0 < 2^{\aleph_0}$, on top of that, we can repeatedly use power set to generate greater and greater infinities.

$$\aleph_0, 2^{\aleph_0}, 2^{2^{\aleph_0}}, \dots \quad (6.1)$$

Transfinite numbers

Cantor named the series of leveled infinite cardinal numbers as *transfinite cardinal numbers*. The story of Hilbert Grand hotel, actually demonstrates the arithmetic rules of transfinite numbers like $\aleph_0 + 1 = \aleph_0$, $\aleph_0 + k = \aleph_0$, $\aleph_0 + \aleph_0 = \aleph_0$, ...

Besides power set, Cantor found another method to generate greater infinities. To understand this method, we need introduce the concept of ordinal number. It is defined recursively as below:

1. 0 is an ordinal number;
2. If a is an ordinal number, then $a \cup \{a\}$ is also an ordinal number, denoted $a + 1$. It is called the successor of a ;
3. If S is a set of ordinals (its elements are all ordinal numbers), then $\cup S$ is an ordinal number;
4. Any ordinal number is obtained from the above 3 steps.

From this definition, we can list the first several ordinal numbers from 0 as the following:

$$\begin{aligned} 0 \\ 1 &= 0 \cup \{0\} \\ 2 &= 1 \cup \{1\} = 0 \cup \{0\} \cup \{0 \cup \{0\}\} \\ 3 &= 2 \cup \{2\} = 1 \cup \{1\} \cup \{1 \cup \{1\}\} = \dots \\ &\dots \end{aligned}$$

Where $\cup S$ is the union of all its elements, sometimes called as *infinitary union*. According to the first two items in the definition of ordinals, the natural numbers 0, 1, 2, 3, ..., n , ... are all ordinals. Let ω be the set of natural numbers, because all natural numbers are ordinals, hence ω is a set of ordinals. Consider its infinitary union:

$$\cup \omega = \{0, 1, 2, \dots\} = \omega$$

According to the third item in ordinal definition, ω is also an ordinal. It is a *limit ordinal*¹¹, and is the smallest infinite ordinal. We can append it to the end of natural numbers to form a new series:

$$0, 1, 2, \dots, \omega$$

Start from ω , repeatedly applying the second item in ordinal definition, gives a new ordinal series:

$$\omega + 1, \omega + 2, \omega + 3, \dots, \omega + n, \dots$$

Combine the above two series into one set, denoted as $\omega \cdot 2$. Its infinitary union is $\cup \omega \cdot 2 = \omega \cdot 2$, hence $\omega \cdot 2$ is also an ordinal, and it is a limit ordinal. From $\omega \cdot 2$, repeat the above process, we obtain an infinite of infinite ordinal series:

$$\begin{aligned} & 0, 1, 2, \dots, n, \dots \\ & \omega, \omega + 1, \omega + 2, \dots, \omega + n, \dots \\ & \omega \cdot 2, \omega \cdot 2 + 1, \omega \cdot 2 + 2, \dots, \omega \cdot 2 + n, \dots \\ & \dots \\ & \omega \cdot k, \omega \cdot k + 1, \omega \cdot k + 2, \dots, \omega \cdot k + n, \dots \\ & \dots \\ & \omega^2, \omega^2 + 1, \omega^2 + 2, \dots, \omega^2 + n, \dots \\ & \dots \\ & \omega^3, \omega^3 + 1, \omega^3 + 2, \dots, \omega^3 + n, \dots \\ & \dots \\ & \omega^\omega, \omega^\omega + 1, \omega^\omega + 2, \dots, \omega^\omega + n, \dots \\ & \dots \end{aligned} \tag{6.2}$$

Except the first row is natural numbers, all the others are infinite ordinals, and the first one in every row is the limit ordinal. For the ordinals obtained by this method, they are far from what people could imagined before. It extends the natural numbers to a kingdom of infinite ordinals. What's more surprising, these ordinals are all countable! As a set, it has 1-to-1 correspondence with natural numbers. We'll soon see, there exist uncountable ordinals, further, there exit greater and greater infinite ordinal series one by one.

Among these infinite ordinals, which one is the best as the cardinal number for infinite countable set? It's natural to select the smallest limit ordinal ω . Let's give the formal definition for cardinal number:

Definition 6.3.1. *An ordinal a is a cardinal if there is no ordinal $b < a$ with $a \cong b$.*

In other words, ordinal a is a cardinal, if for any ordinal $b < a$, the cardinality of b is less than the cardinality of a . This definition tell us every natural numbers n is a cardinal, and ω is also a cardinal. When use the ordinal ω as cardinal, we write it as \aleph_0 , hence $\aleph_0 = \omega$. We've shown to use \aleph_0 as the cardinal for all infinite countable sets.

Except ω , although all rest ordinals in series 6.2 are greater than ω , their cardinals are same as ω (all equal to countable infinity). Hence they are not cardinals.

In order to obtain greater cardinals, we form a new set contains all ordinals in series 6.2, denote it as ω_1 .

$$\omega_1 = \{a \mid a \text{ is ordinal, and } |a| \leq \aleph_0\}$$

¹¹A nonzero ordinal that is not a successor is called a limit ordinal.

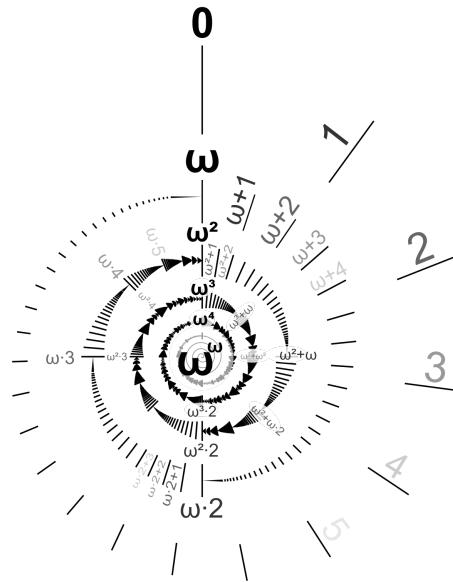


Figure 6.23: Infinite ordinals

Where $|a|$ is the cardinal of a ¹². We can prove that ω_1 is a cardinal, and it is the first uncountable cardinal. Then, we repeat this method, to expand a new infinite ordinal series from ω_1 :

$$\omega_1, \omega_1 + 1, \dots, \omega_1 \cdot 2, \dots, \omega_1^2, \dots, \omega_1^\omega, \dots$$

All elements in this series have the same cardinality. The smallest one is ω_1 , which also satisfies the cardinal definition. It is the second infinite cardinal $\aleph_1 = \omega_1$. Using the similar process to construct ω_1 , we can form another set:

$$\omega_2 = \{a \mid a \text{ is ordinal, and } |a| \leq \aleph_1\}$$

It gives the third infinite cardinal $\aleph_2 = \omega_2$. Repeat this method, we can obtain a series of infinite cardinals. In summary, for any ordinal a , we can define the infinite cardinal \aleph_a , then form a set:

$$\omega_{a+1} = \{b \mid b \text{ is ordinal, and } |b| \leq \aleph_a\}$$

It gives a cardinal greater than \aleph_a as $\aleph_{a+1} = \omega_{a+1}$. For any ordinal a , there is a infinite cardinal \aleph_a . All these cardinals also form a series:

$$\aleph_0, \aleph_1, \aleph_2, \dots, \aleph_n, \dots, \aleph_\omega, \dots \quad (6.3)$$

From left to right, these cardinals become greater and greater, and there are not any other infinite cardinals between any two next \aleph s. The infinite ordinals and infinite cardinals are also called transfinite ordinals and transfinite cardinals, or transfinite numbers as a whole. Where will these more and more numerous transfinite numbers lead to? Cantor thought it must be god.

People were surprised at transfinite numbers. Some praised Cantor's amazing innovation opened up a new world we've never seen before; Others criticized transfinite

¹²To be accurate, we should use other notation for the cardinal of A , like $\overline{\overline{A}}$, or $\#A, card(A), n(A)$ etc.

numbers were “fog on fog”, Cantor was building a disease of mathematics need to be cured. Although there was hotly debating, transfinite number was one of the most amazing achievement of thought in the 19th Century.

Continuum hypothesis

Cantor found two types of infinite cardinal series, one is power sets, the other is transfinite cardinals:

$$\aleph_0, 2^{\aleph_0}, 2^{2^{\aleph_0}}, \dots$$

and

$$\aleph_0, \aleph_1, \aleph_2, \dots$$

\aleph_0 is the cardinal of the infinite countable set. In previous section, we reasoned that, \aleph_1 is the next transfinite cardinals next to \aleph_0 . However, according to Cantor’s theorem of power set, we only know that 2^{\aleph_0} is more numerous than \aleph_0 , but we do not know if it is more or less numerous than \aleph_1 . Cantor conjectured $2^{\aleph_0} = \aleph_1$, that there wasn’t any other transfinite cardinals between \aleph_0 and 2^{\aleph_0} . Hence 2^{\aleph_0} is the first transfinite cardinal more numerous than infinite countable set.

In 1847, Cantor proved $2^{\aleph_0} = C$. It means all subsets of natural numbers and real numbers have the same cardinality. Therefore, Cantor’s conjecture essentially states that, there exists no set whose cardinality is strictly greater than that of natural numbers \aleph_0 and less than that of real numbers C . Because real numbers are often called continuum, this conjecture is called *Continuum Hypothesis*, abbreviated as CH.

Continuum Hypothesis can be further extended. For any ordinal a , whether $2^{\aleph_a} = \aleph_{a+1}$ holds. This conjecture is called generalized continuum hypothesis, abbreviated as GCH.

Cantor raised continuum hypothesis in a paper in 1878. He believed it to be true and tried for many years to prove it. Sometimes he thought he had proved it false, then the next day found his mistake. Again he thought he had proved it true only again to quickly find his error. His inability to prove the continuum hypothesis caused him considerable anxiety till his death in 1918.

The problem, whether we can prove continuum hypothesis true or false became the first on David Hilbert’s list of 23 important open questions that was presented at the International Congress of Mathematicians in 1900 in Paris. The continuum hypothesis came from the practical problems from geometry, mechanics, and physics. Hilbert expressed this view:

But even this creative activity of pure thought is going on, the external world once again reasserts its validity, and by thrusting new questions upon us through the phenomena that occur, it opens up new domains of mathematical knowledge.

Because the continuum hypothesis is the most central open problem at the foundation of mathematical logic and axiomatic set theory, it has been studied by many great mathematicians for over hundred years. Although many significant progresses were made, it is not completely resolved. Kurt Gödel proved in 1938 that the negation of the continuum hypothesis, the existence of a set with intermediate cardinality, could not be proved in standard set theory (also known as Zermelo-Fraenkel axioms for set theory together with the axiom of choice or AC. Informally, AC says that given any collection of non-empty bins (even infinite), it is possible to make a selection of exactly one object from each bin. we’ll introduce the details in next chapter). The second half of the independence of the

continuum hypothesis, unprovability of the nonexistence of an intermediate-sized set, was proved in 1963 by Paul Cohen with a new powerful technique called *forcing*. There was an interesting story said that Cohen, the young US mathematician was not sure about his proof([8], pp. 280). He came to Princeton and knocked on Gödel's house. Gödel was struggling with paranoia at the time. He slightly open the door, such that Cohen could pass his proof through. Then Cohen was shut out. Two days later, Gödel invited Cohen came in to drink tea, and finally accept his proof. Cohen was awarded a Fields Medal the next year.

Gödel and Cohen proved, the continuum hypothesis is undecidable from ZFC set theory. We'll explain more about undecidable statement in next chapter. Continuum hypothesis is independence from the axioms in ZF system. Similar result also happens to the axiom of choice. Gödel and Cohen's result also tells that AC is undecidable from ZF system. Accepting AC gives the consistent mathematical system; while rejecting AC gives another consistent mathematical system. With the axiom of choice, accepting or rejecting continuum hypothesis all gives consistent mathematics respectively. Continuum hypothesis and the axiom of choice is independent to ZF set theory[61].

6.4 Infinity in art and music



Vincent van Gogh, *The Starry Night*, Museum of Modern Art, New York

Along with the thought and exploration, infinity also inspired art and music when people facing the vast galaxy, the sea, and the mystery nature.

Among the countless art about sky, the *Starry Night* by Dutch post-impressionist artist, Vincent van Gogh is impressive. In his art work, Van Gogh's night sky is a field of roiling energy. Below the exploding stars, the village is a place of quiet order. Connecting Earth and sky is the flame like cypress, a tree traditionally associated with graveyards and mourning. Van Gogh said "Looking at the stars always makes me dream." He painted in June, 1889, in the mental hospital at Saint-Rémy-de-Provence in France, just before sunrise, with the addition of an idealized village. Van Gogh stayed there for 108 days. During this period, he pictured about 150 oil canvas, and hundreds of sketches.

Snow Storm: Steam-Boat off a Harbour's Mouth is a painting by English Romanticism artist J.M.W Turner. The picture may recall a particularly bad storm in January, 1842. In order to feel the power of sea, Turner got the sailors to tie him to the mast to observe. "I was lashed for four hours. and I did not expect to escape, but I felt bound to record it if I did." However, the critical response to the painting was largely negative at the time, with one critic calling it "soapsuds and whitewash". Turner said: "I did not paint it to



Joseph Mallord William Turner, *Snow Storm*, 1842, Tate Modern, London UK

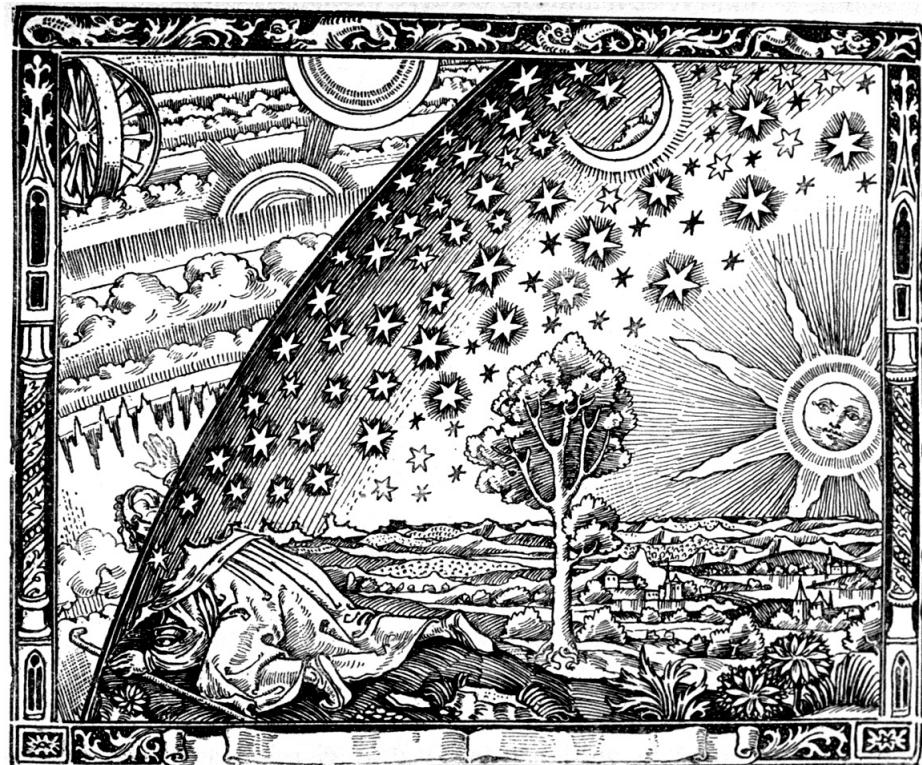
be understood, but I wished to show what such a scene was like.” As time going, people finally realized, as John Ruskin, the leading English art critic of the Victorian era said: “It is one of the very grandest statements of sea-motion, mist and light, that has ever been put on canvas.”

The thought and study of infinity soon became a problem in philosophy and theology from the concrete things in nature. In Ptolemy’s model, the universe is realized as a set of nested spheres, with planets moving along them. The out most boundary is a sphere of fixed stars. By the medieval, the church had largely accepted this model developed by Aristotle and Ptolemy. People believed the Earth, created by God, was the centre of the universe, and the stellar sphere was bounded.

A woodcut published in Paris in 1888, reflected how people understood the bounded universe at that time. The original caption bellow the picture translated to: “A medieval missionary (Bruno) tells that he has found the point where heaven and Earth meet...” If someone stands at the boundary of stellar sphere, can he raise his stick out to across the boundary of universe? It’s hard to prevent him from doing that; but if he can, what is the space out side the physical world? This was the paradox of the universe boundary. To solve it, scholars in medieval refactored Aristotle’s theory, and proposed an idea of progressive boundary. Others believed, if throw a spear outside the world boundary, it would enlarge the universe. The world of matter is bounded, but the boundary is surrounded by endless void.

During the Renaissance, artists adopted mathematics and science into their works. Leonardo da Vinci had wide interest including architecture, anatomy, mathematics, and engineering. He intended to use perspective disciplines, and experiment different aesthetic proportions in his works. German Renaissance artist Albrecht Dürer studied human proportions and the use of transformations to a coordinate grid to demonstrate facial variation. His book *Four Books on Measurement* introduced both painting theory and research on geometric and perspective principles. Johannes Kepler and Gérard Desargues independently developed the concept of the “point at infinity” in projective geometry. Desargues developed an alternative way of constructing perspective drawings by generalizing the use of vanishing points to include the case when these are infinitely far away. He made Euclidean geometry, where parallel lines are truly parallel, into a special case of an all-encompassing geometric system.

It’s the non-Euclidean geometry that brings a new view point to artists about infinity. Euclidean geometry is named after the ancient Greek mathematician Euclid. It has been the perfect example of axiom system and rigours reasoning for two thousand years. However, mathematicians addict to perfection continued questioning Euclid’s fifth postulate.



Un missionnaire du moyen âge raconte qu'il avait trouvé le point
où le ciel et la Terre se touchent...

Camille Flammarion, L'Atmosphère: Météorologie Populaire (Paris, 1888), pp. 163.

The first four are intuitive and obvious, for example: to draw a line from any point to any point; all right angles are equal to one another. However the fifth postulate is disparate complex. In Euclid's original formulation: "If a straight line falls on two straight lines in such a manner that the interior angles on the same side are together less than two right angles, then the straight lines, if produced indefinitely, meet on that side on which are the angles less than the two right angles." This postulate is also known as "parallel postulate", because it is essentially equivalent to a simpler statement: for any given line and a point not on it, there is exactly one line through the point that does not intersect the line. People doubted if this postulate could be deduced from the first four. Actually, the fifth postulate hasn't been used in a large portion in Euclid's Elements. Many mathematicians attempted to prove the fifth postulate in the past two thousand years, but all failed. Italian mathematician Saccheri then tried to prove by contradiction. He assumed the fifth postulate is false, but he obtained a series of obscure results. Saccheri believed they were too strange, and Euclid's fifth postulate must be true.

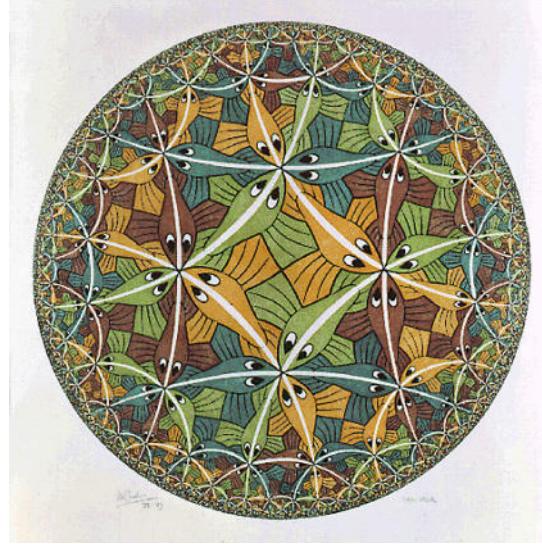
In 19th Century, German mathematician Carl Friedrich Gauss, Hungarian mathematician János Bolyai and the Russian mathematician Nikolai Ivanovich Lobachevsky separately realized it's not possible to establish such proof. Parallel postulate is independent to other postulates. And it can be replaced with other different "parallel postulate". Gauss did not publish his result, people found his finding after Gauss died in 1885. Bolyai and Lobachevsky independently published treatises on non-Euclidean geometry around 1830. In this new geometry (known as hyperbolic geometry nowadays), Lobachevsky replaced Euclid's postulate: in a plane, given a line and a point not on it, there are multiple lines through a point that do not intersect the line. Then he developed a whole set of consistent results. Many of them are different from Euclidean geometry. For example, in the new geometry, the inner angle of a triangle is less than two right angles.

German mathematician Bernhard Riemann, in 1854 constructed another new geometry that was different from both Euclid and Lobachevsky. He founded the field of Riemannian geometry, and the simplest of these is called elliptic geometry. It's non-Euclidean due to its lack of parallel lines. It means every two lines must intersect in a plane. In elliptic geometry, the inner angle of a triangle is greater than two right angles.

Eugenio Beltrami, in 1868, showed that Euclidean geometry and hyperbolic geometry were equiconsistent so that hyperbolic geometry was logically consistent if and only if Euclidean geometry was. In 1871, Felix Klein defined a metric method to describe the non-Euclidean geometries. Klein's influence has led to the current usage of the term "non-Euclidean geometry" to mean either "hyperbolic" or "elliptic" geometry.

Non-Euclidean geometry brings us such possibility, that infinite space can be bounded. Poincaré, in his popular science book *Since and Hypothesis*, described a interesting world. The world is encapsulated in a infinite bounded sphere. The temperature at the centre is very high. Along the distance away from the centre, the temperature decrease in proportion. When arrive at the sphere surface, it decrease to absolute zero K. Let the radii of the world sphere be R , for a point with the distance to the centre as r , then the temperature is proportion to $R^2 - r^2$. In this special world, due to thermal expansion, object size is proportion to the temperature, the closer to the sphere boundary, the smaller. Because of this, if a citizen moving towards the sphere, the temperature decreases lower and lower. He also becomes smaller and smaller. The pace keeps slowing down. He will never reach to the boundary of this world, although this world is bounded. What Poincaré described is exactly a world ruled by a kind of hyperbolic geometries.

Inspired by Poincaré, the Dutch artist M.C Escher painted a series of woodcuts, called circle limit to illustrate the bounded, infinite world. The angles, devils, and fishes all become smaller and smaller when close to the edge of the circle, hence they will never exceed this bounded, while infinite world.



M.C. Escher. Circle limit III, 1959

Not only in art, there are musics about infinity. In May, 1747, Johann Sebastian Bach visited Sanssouci Palace in Postdam. King Frederick II invited Bach to try out his new Silbermann pianos. Bach asked the King to give him a subject of Fugue, then immediately executed it without any preparation to the astonishment of all present. After his return to Leipzig, he composed the subject, which he had received from the King, in three and six parts, added several artificial passages in strict canon to it, and had it engraved, under the title of "Musical Offering". He sent a copy to the King on July 7. This collection of music is catalogued as BWV1079 nowadays. Among them there is a special piece with title "Canon per Tonos". It is called endlessly rising canon. It pits a variant of the king's theme against a two-voice canon at the fifth. However, it modulates and finishes one whole tone higher than it started out at. It thus has no final cadence. Douglas Hofstadter, in his Pulitzer Prize book *Gödel, Escher, Bach: An Eternal Golden Braid*, wrote:



Part of the endlessly rising canon

It has three voices. The uppermost voice sings a variant of the Royal Theme, while underneath it, two voices provide a canonic harmonization based on a second theme. The lower of this pair sings its theme in C minor (which is the key of the canon as a whole), and the upper of the pair sings the same theme displaced upwards in pitch by an interval of a fifth. What makes this canon different from any other, however, is that when it concludes—or, rather, seems to conclude—it is no longer in the key of C minor, but now is in D minor.

Somehow Bach has contrived to modulate (change keys) right under the listener's nose. And it is so constructed that this "ending" ties smoothly onto the beginning again; thus one can repeat the process and return in the key of E, only to join again to the beginning. These successive modulations lead the ear to increasingly remote provinces of tonality, so that after several of them, one would expect to be hopelessly far away from the starting key. And yet magically, after exactly six such modulations, the original key of C minor has been restored! All the voices are exactly one octave higher than they were at the beginning, and here the piece may be broken off in a musically agreeable way. Such, one imagines, was Bach's intention; but Bach indubitably also relished the implication that this process could go on ad infinitum, which is perhaps why he wrote in the margin "As the modulation rises, so may the King's Glory." To emphasize its potentially infinite aspect[5].

Exercise 6.5

1. Light a candle between two opposite mirrors, what image can you see? Is it potential or actual infinity?

6.5 Appendix - Example programs

Define natural numbers with stream, take the first 15 numbers. Example program in Java 1.8

```
IntStream.iterate(1, i -> i + 1);

IntStream.iterate(1, i -> i + 1)
    .limit(15).forEach(System.out::println);
```

Example in Python 3

```
def naturals():
    yield 0
    for n in naturals():
        yield n + 1
```

Define the infinite set of natural numbers recursively in Haskell.

```
nat = 1 : (map (+1) nat)

take 15 nat
```

Define the infinite set of Fibonacci numbers recursively in Haskell, then fetch the 1500th Fibonacci number.

```
fib = 0 : 1 : zipWith (+) fib (tail fib)

take 15 fib
[0,1,1,2,3,5,8,13,21,34,55,89,144,233,377]

fib !! 1500
13551125668563101951636936867148408377786010712418497242133543153221487310
87352875061225935403571726530037377881434732025769925708235655004534991410
2924249595997483982286992875272419318113250950996424476212422002092544399
20196960465321438498305345893378932585393381539093549479296194800838145996
187122583354898000
```

Define the infinite stream of prime numbers with coalgebra in Haskell.

```

data StreamF e a = StreamF e a
data Stream e = Stream e (Stream e)

ana :: (a -> StreamF e a) -> (a -> Stream e)
ana f = fix . f where
  fix (StreamF e a) = Stream e (ana f a)

takeStream 0 _ = []
takeStream n (Stream e s) = e : takeStream (n - 1) s

era (p:ns) = StreamF p (filter (p `notdiv`) ns)
  where notdiv p n = n `mod` p /= 0

primes = ana era [2..]

takeStream 15 primes
[2,3,5,7,11,13,17,19,23,29,31,37,41,43,47]

```

6.6 Appendix - Proof of Cantor's theorem

Theorem 6.6.1. Cantor's theorem: For every set, $|S| < |2^S|$ holds. Where $|S|$ is the cardinality of S ; 2^S is the power set of S , which contains all subsets of S .

Proof. Our proof has two steps. The first step is to prove $|S| \leq |2^S|$. For any x , let $f(x) = \{x\}$, which is the singleton set of x . It's obvious that for different elements $x_1 \neq x_2$, the singleton sets are not identical $\{x_1\} \neq \{x_2\}$, which means $f(x_1) \neq f(x_2)$. Hence map $S \xrightarrow{f} 2^S$ is injective, we have:

$$|S| \leq |2^S|$$

The second step is to prove $|S| \neq |2^S|$. Using the reduction to absurdity, suppose they are equal. Then there exists a 1-to-1 correspondence $S \xrightarrow{\phi} 2^S$, such that for every $x \in S$, its image $\phi(x) \in 2^S$ holds. It means $\phi(x)$ is some subset of S , hence $\phi(x) \subseteq S$. Now we ask whether x belongs to $\phi(x)$? Either $x \in \phi(x)$, or $x \notin \phi(x)$. We put all x that not belonging to $\phi(x)$ to form a new set S_0 :

$$S_0 = \{x | x \in S, \text{ and } x \notin \phi(x)\} \quad (6.4)$$

Obviously, S_0 is a subset of S , which means $S_0 \subseteq S$. Hence $S_0 \in 2^S$. Because ϕ is bijection, there must exist some x_0 , such that $\phi(x_0) = S_0$. According to the logical law of excluded middle, either $x_0 \in S_0$, or $x_0 \notin S_0$. Both must be one and only one is true.

Let's see each case next. If $x_0 \in S_0$ holds, according to the definition of S_0 in (6.4), we have $x_0 \notin \phi(x_0)$. But since $\phi(x_0) = S_0$, hence $x_0 \notin S_0$.

If $x_0 \notin S_0$, because $S_0 = \phi(x_0)$, we have $x_0 \notin \phi(x_0)$. But according to the definition of S_0 in (6.4), $x_0 \in S_0$ should hold.

Hence no matter x_0 belongs to S_0 or not, both lead to contradiction. There cannot be 1-to-1 correspondence between S and 2^S established. Therefore, $|S| \neq |2^S|$.

Summarize the result in above two steps: $|S| \leq |2^S|$, and $|S| \neq |2^S|$, we prove the Cantor's theorem:

$$|S| < |2^S|$$

□

It reminds us the popular Russel's paradox from the second part of the proof: Let S be a set containing all sets that each not belong to itself, then does S belong to itself? We'll introduce Russel's paradox and Gödel's incompleteness theorems in next chapter.

6.7 Appendix - Canon per tonos, The Music Offering by J.S. Bach

Canon a 2. (Per tonos.)

Musikalisches Opfer BWV 1079

Johann Sebastian Bach

1

2

3

Chapter 7

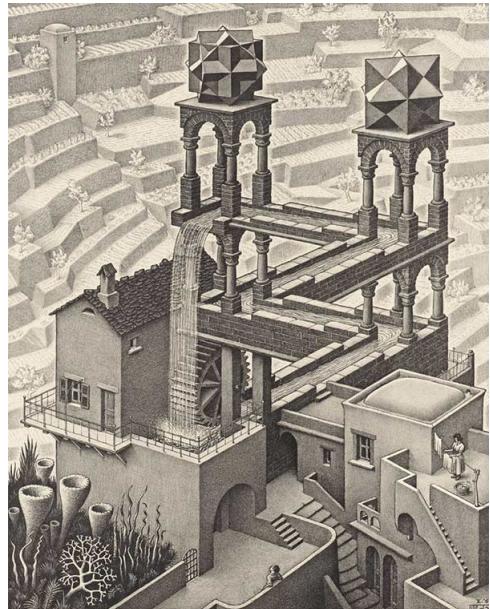
Paradox

I know that I know nothing

—Socrates

In 1996, the 26th Summer Olympic game was held in Atlanta, U.S. More than 10 thousand athletes from 197 nations challenged human limit of speed, strength, and team work in 26 sports. At the same time, there was another interesting match on-going. An IBM computer, called Deep Blue challenged the world chess champion Garry Kasparov in a six-game match. Deep Blue won the first game, but Kasparov won three and drew two, defeating Deep Blue by a score of 4:2. The next year, heavily upgraded Deep Blue challenged again to human world champion Kasparov. On May 11, computer defected human: two wins for Deep Blue, one for the champion, and three draws. Deep Blue is a super computer of 1270 kilogram weight, with 32 processors. It can explore 200 million possible moves in a second. The design team input 2 million grandmaster games in the past 100 years as the knowledge base for Deep Blue. The machine created by human intelligence, defected human at the first time in the field of intelligence. This result led to attention, fear, and hotly debate among mass media.

Most people believed this was a significant progress in artificial intelligence at that time. Although computer could defeat human for chess, but there was still a big gap in board game Go. There are 8 rows and 8 columns in chess board, and 32 pieces. Computer need search among a big game tree containing about 10^{123} possible moves. Even Deep Blue could explore 2 million moves per second, it would take about 10^{107} years to exhaust the tree. The design team optimized the program to narrow down the search space, such that Deep Blue only need explore 12 moves ahead from current game,



Escher, Waterfall, 1961

while human grandmasters can only evaluate about 10 moves ahead. However for Go game, there are 19 rows and 19 columns, two players can put black and white pieces in 361 grids. The scale of the game tree is about 10^{360} , which is far bigger than chess. For a long time after Deep Blue, people did not believe computer could defeat human in Go.



Deep Blue versus Kasparov. from *Scientific American*

20 years later in 2016, a computer program ‘AlphaGo’ challenged top human Go master. Korean professional 9-dan Go player, Lee Sedol, lost the game in a 1:4 series matches. One year after, the successor program ‘AlphaGo master’ beat Chinese professional Ke Jie, the world number one ranked player, in a three-game match. Go had previously been regarded as a hard problem in artificial intelligence that was expected to be out of reach for the technology of the time. It was considered the end of an era. Facing the emotionless machine, Ke Jie was unwilling and burst into tears. As human beings, our feelings are mixed. Even the programmer community doing intellectual work is feeling the pressure from machine: will machine replace us eventually?

Traditionally, we thought the areas with culture background, inner emotions, and human characters, like art, literature, and music could not be dominated by machine. In 2015, three researchers Gatys, Ecker, and Bethge from University of Tübingen, a small town 30 km south of Stuttgart, Germany, applied machine learning to art style. By using deep convolutional neural network, they transformed a landscape photo of Tübingen into art painting of different styles[64]. No matter the exaggerated emotion of the post-impressionist Van Gogh, or Turner’s romantic turbid light and shadow effect, all vivid imitated by machine, as if the artists painted by themselves (figure 7.3).

In the following years, artificial intelligence and machine learning conquered varies of areas in accelerated speed. Machines generated different styles of music, and played them with moods and rhythms of tension, relaxing, and so on. It is not the monotonous electronic sound anymore. Machine batch translated news and academic papers, which is comparable to human professional translators. Machine processed X-ray photos, CT, and MRI medical images to diagonal diseases, and the accuracy exceeded human doctors. Self-driven cars, powered by artificial intelligence traveled on streets, successfully overtaking other vehicles and avoid pedestrians. Automated groceries suddenly appeared on the street, people can pick the products and walk out without being checked out by a cashier... As humans we can’t stop asking: Are we eliminating jobs faster than creating? Will human be replaced by machine completely? Will machine rules people in the future?

All these lead to a critical question: does there exist boundary of computation? if yes, where is it?

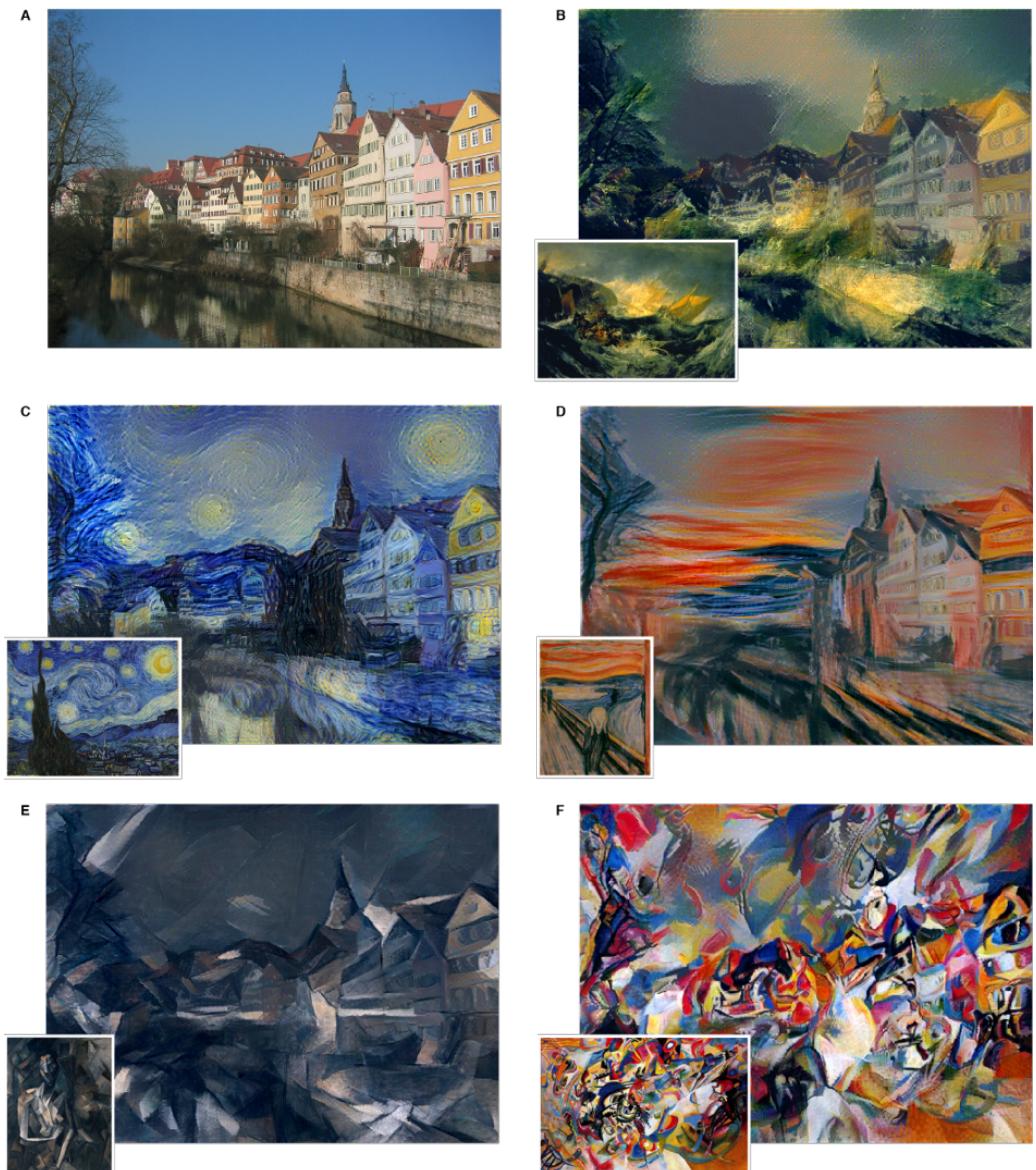


Figure 7.3: Artworks in different styles generated by machine learning: **A**, Landscape photo of Tübingen; **B**, *The Shipwreck of the Minotaur* by J.M.W. Turner, 1805; **C**, *The Starry Night* by Vincent van Gogh, 1889; **D**, *Der Schrei* by Edvard Munch, 1893; **E**, *Femme nue assise* by Pablo Picasso, 1910; **F**, *Composition VII* by Wassily Kandinsky, 1913.

7.1 Boundary of computation

Gu Sen described the hesitated feelings when facing a long-running program in his popular book *Fun of thinking*: Will this program finish? Shall I wait or kill it? Is there a compiler could tell if my program will run endlessly ([65] pp.228)?

Why not possible? It seems more realistic than time machine. We may see such a scene in a scientific film: a programmer typed something in the dark screen, then hit enter. A highlighted, bold warning popped up immediately “Warn: the program with the given input will run forever. Continue? (Y/N)” If this became true one day, what fantastic cool things will you do? Do you believe that I can make big money with it? I’ll firstly use it to prove the Goldbach’s conjecture. I can write a program, enumerate all even numbers one by one, examine if it is the sum of two prime numbers. If yes, then check the next even number, otherwise output the negative example and quit. The next thing is to compile my program. Can’t the compiler determine my program terminates or not in advance? If the magic compiler warns me that my program will run endlessly, haven’t I proved Goldbach’s conjecture? Or if the compiler tells me the program will terminate, doesn’t it mean Goldbach’s conjecture is falsehood? Either case, I’ll be the first one that solve the Goldbach’s conjecture, and leave my name in mathematical history. What’s the next? I will modify that program to explore the twin primes, then compile it to see if there are really infinite many twin primes. And next, are there infinite many Mersenne primes? This is also an open question in number theory for a long time. I can easily solve it in this way. The $3x+1$ conjecture? It’s a piece of cake to write a “proof program” in a few minutes, then win the 500 dollars prize offered by Paul Erdős. There are enough mathematical open questions, I’ll never worry about nothing to do. Martin LaBar in 1984 asked if a 3×3 magic square can be constructed with nine distinct square numbers. The award has accumulated to 100 dollars, 100 euros, and a bottle of champagne. Search “Unsolved problems in mathematics” from internet, filter in those about discrete things, and with award, then write a few programs to solve them all...

In 1936, the pioneer of computer science and artificial intelligence, Alan Turing proved that, a general algorithm to determine an arbitrary computer program will finish running, or continue to run forever, cannot exist. A key part of the proof was a mathematical definition of a computer and program, which became known as a Turing machine. This problem is called *halting problem* today.

We can use reduction to absurdity method to prove the Turing’s halting problem. Suppose there exists a algorithm *halts*(*p*), that can determine an arbitrary program *p* terminates or not. First, we construct a never halting program:

$$\text{forever}() = \text{forever}()$$

This is a infinitely recursive call. Then define a special program *G* as below¹:

$$G() = \begin{cases} \text{halts}(G) = \text{True} : & \text{forever}() \\ \text{otherwise} : & \text{halt} \end{cases}$$

In program *G*, we utilize *halts*(*G*) to examine whether *G* itself will halt or not. If it halts, then we call *forever*() to let it run forever. It exactly means *G* will not halt

¹We intend to use *G* for special meaning. It’s Gödel’s initial letter, exactly the same name for non-deterministic proposition in Gödel’s incompleteness theorem.

in this case, hence $halts(G)$ should be false. However, according to the second clause, it will halt. Therefore $halts(G)$ should be true. whether $halts(G)$ is true or false, we obtain conflicted result. Hence our assumption can't hold. There does not exit a general algorithm to solve the halting problem for all possible program input.

There is another method to prove the halting problem in two steps([66] pp.268). Most are same except that G accepts another argument p , it applies p to itself, then passes to $halts$:

```
G(p) = if halts(p(p)) then forever() else 'Halted'
```

Let's see what will happen when pass G to itself $G(G)$. If $halts(G(G))$ returns true, then it calls $forever()$, hence $G(G)$ never finishes. While it exactly means $halts(G(G))$ should returns false, hence the program enters the `else` branch, and halts. But it again means $halts(G(G))$ should return true. Whether halts or not, it leads to absurdity.

The halting problem clearly provides a incomputable problem, breaks the bubble of all the above magic ideas. It reminds us the proof of Cantor's theorem in previous chapter, where we used quite similar method to prove that for all sets, including infinite sets, the cardinals are strict less than their power sets. Actually, halting problem is closely related to many interesting logic paradoxes.

7.2 Russel's paradox

The history of paradox came back to ancient Greece. We've introduced Zeno's paradoxes about infinity and continuity. Logic paradox is often an interesting problem, with strict reasoning but deduced to conflicted result. About the fourth Century BC, the ancient Greek philosopher Eubulides of Miletus raised a proposition: "I am lying." How to determine if this declaration is true or false?

If this declaration is false, then what it states (lying) should be true, it conflicts; however if this declaration is true, since it states I am laying, it should be false and lead to conflict again. Whether what Eubulides said is true or not, all falls into contradiction. This confusing problem is called 'liar paradox'.

There is a variance of liar paradox, appeared as two separated statements:

Achilles: The tortoise is a liar, he always lies. Do not trust him.

Tortoise: Dear Achilles, you are honest, you always speaks truth.

Is it true or false for what the tortoise said? If the tortoise tells the truth, then what Achilles states is true. However, Achilles claims the tortoise is laying, it leads to contradiction. On the contrary, if the tortoise lies, then what Achilles says is wrong, hence the tortoise should be true. We end up with a wired-loop: whether the tortoise speaks truth or not, all leads to absurdity.

This two-segment liar paradox sometimes appears as a joke. You receive a piece of note: "The other side is nonsense." while when you flip to the other side, it writes: "The other side is true." Which side is true? In a similar way, it reduces to contradiction.

Such paradox also appears in children's story. A lion caught a rabbit. He is so happy, that he promise the rabbit: If you can guess what I am going to do, I'll let you go; otherwise I'll eat you. The clever rabbit then answers: I guess you are going to eat me.

If the lion eats the rabbit, then the rabbit guesses correct, the lion should keep his promise to let the rabbit go. However, if he let the rabbit go, it means the rabbit guesses wrong. Hence the lion should eat the rabbit. The lion falls into the dilemma, he should neither eat the rabbit, nor let the rabbit go. We can imagine the rabbit silently runs away when the lion keeps deep thinking.

According to the legend, after ancient Greek army defected Persian, the king decided to do something kind to the captives – let them chose the way to be killed. According to

what the captive said, if it is true, then cut head off, otherwise hang. A clever captive said: "I think you are going to hang me." If the king hangs him, then what the captive said is true. Hence he should be cut head according to the rule. But if cut his head off, then it does not follow what the captive said. Hence he spoke falsehood, and should be hanged. Whether cut head or hang, the king's rule will not be conducted correctly. Facing such struggled situation, the king did not only let this clever man go, but also released all captives.



E. O. Plauen *Father and Son*, 1930s

In Cervantes' novel *Don Quixote*, there is an interesting paradox in Part II, Chapter 51:

A deep river divides a certain lord's estate into two parts... over this river is a bridge, and at one end a gallows and a sort of courthouse, in which four judges sit to administer the law imposed by the owner of the river, the bridge and the estate. It runs like this: "Before anyone crosses this bridge, he must first state on oath where he is going and for what purpose. If he swears truly, he may be allowed to pass; but if he tells a lie, he shall suffer death by hanging on the gallows there displayed, without any hope of mercy." ...

Now it happened that they once put a man on his oath, and he swore that he was going to die on the gallows there – and that was all. After due deliberation the judges pronounced as follows: "If we let this man pass freely he will have sworn a false oath and, according to the law, he must die; but he swore that he was going to die on the gallows, and if we hang him that will be the truth, so by the same law he should go free."

Besides the liar paradox, the barber paradox is another popular puzzle. It was told by British mathematician and logician, Bertrand Russel in 1919. In a small village, the barber sets up a rule for himself: "He only shaves all those, and those only, who do not shave themselves." Then the question is, does the barber shave himself? If he shaves himself, the according to his rule, he should not shave himself; but if he does not, then he should serve and shave himself. The barber falls into his own trap.

Russel discovered the paradox in set theory early in 1901. He collected and summarized a series of paradoxes, and formalized them as a fundamental problem in set theory. People called this kind of paradoxes as *Russell's paradox*. In Cantor's naive set theory, Russell considered the problem about if any set belongs to itself. Some sets do, while others not. For example the set of all spoons is obviously not another spoon; while the set of anything that is not a spoon, is definitely not a spoon. Russell considered the latter, and extended it to all such cases. He constructed a set R , which contains all sets that are not members of themselves. Symbolically:

$$R = \{x | x \notin x\}$$

Russell next asked, is R a member of R ? According to logical law of excluded middle, an element either belongs to a set, or does not. For a given set, it makes sense to ask whether the set belongs itself. But this well defined, reasonable question falls into contradiction.

If R is a member of R , then according to its definition, R only contains the sets that are not members of themselves, hence R should not belong to R ; On the contrary, if R is not a member of R , again, from its definition, any set does not belong to itself should be contained, hence R is a member of R . Whether it is a member or not, gives contradiction. Formalized as:

$$R \in R \iff R \notin R$$

Russell explicitly gave the paradox in Cantor's set theory.

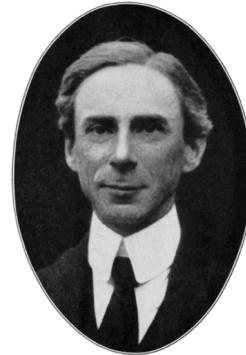
Bertrand Russell was born in 1872 in Monmouthshire into a family of the British aristocracy. Both his parents died before he was three, and his grandfather died in 1878. His grandmother, the countess was the dominant family figure for the rest of Russell's childhood and youth. Her favourite Bible verse, "Thou shalt not follow a multitude to do evil", became his motto.

Russell was educated at home by a series of tutors. When Russell was eleven years old, his brother Frank introduced him to the work of Euclid, which he described in his autobiography as "one of the great events of my life, as dazzling as first love." During these years, he read about the poems of Percy Bysshe Shelley, and thought about religious and philosophy. In 1890, Russell won a scholarship to read for the Mathematical Tripos at Trinity College, Cambridge, where he became acquainted with Alfred North Whitehead. He quickly distinguished himself in mathematics and philosophy, graduating as seventh Wrangler in the former in 1893 and becoming a fellow in the latter in 1895.

Russell started an intensive study of the foundations of mathematics. He discovered Russell's paradox. In 1903 he published *The Principles of Mathematics*, a work on foundations of mathematics. It advanced a thesis of logicism, that mathematics and logic are one and the same. The three-volume *Principia Mathematica*, written with Whitehead, was published between 1910 and 1913. This, along with the earlier *The Principles of Mathematics*, soon made Russell world-famous in his field.

After the 1950s, Russell turned from mathematics and philosophy to international politics. He opposed nuclear war. The Russell–Einstein Manifesto was a document calling for nuclear disarmament and was signed by eleven of the most prominent nuclear physicists and intellectuals of the time. Russell was arrested and imprisoned twice. The second time he was in jail was at the age of 89, for "breach of peace" after taking part in an anti-nuclear demonstration in London. The magistrate offered to exempt him from jail if he pledged himself to "good behaviour", to which Russell replied: "No, I won't." In 1950 Russell won the Nobel Prize for Literature. The committee described him as "in recognition of his varied and significant writings in which he champions humanitarian ideals and freedom of thought."

Russell died of influenza on February 2nd, 1970 at his home in Penrhyneddraeth. In accordance with his will, there was no religious ceremony; his ashes were scattered over the Welsh mountains later that year.



Bertrand Russell, 1872 - 1970

7.2.1 Impact of Russell's paradox

Russell was sad after discovering the paradox in the central of set theory. “What makes it vital, what makes it fruitful, is the absolute unbridled Titanic passion that I have put into it. It is passion that has made my intellect clear, ... it is passion that enabled me to sit for years before a blank page, thinking the whole time about one probably trivial point which I could not get right ...”([8] pp.231) Russell wrote to mathematician and logician Gottlob Frege about his paradox. Frege was about to build the foundation of arithmetic. The second volume of his *Basic Laws of Arithmetic* was about to go to press. Frege was surprised, he wrote: “Hardly anything more unwelcome can befall a scientific writer than that one of the foundations of his edifice be shaken after the work is finished. This is the position into which I was put by a letter from Mr Bertrand Russell as the printing of this volume was nearing completion...” Russell’s paradox in axiomatic set theory was disastrous. Further, since set theory was seen as the basis for axiomatic development of all other branches of mathematics, Russell’s paradox threatened the foundation. This motivated a great deal of research around the turn of the 20th century to develop a consistent (contradiction free) mathematics.

Exercise 7.1

1. We can define numbers in natural language. For example “the maximum of two digits number” defines 99. Define a set containing all numbers that cannot be described within 20 words. Consider such an element: “The minimum number that cannot be described within 20 words”. Is it a member of this set?
2. “The only constant is change” said by Heraclitus. Is this Russell’s paradox?
3. Is the quote saying by Socrates (the beginning of this chapter) Russell’s paradox?

7.3 Philosophy of mathematics

To solve Russell’s paradox that affects the foundation of mathematics and logic, many mathematicians continued discussing, debating, and proposed varies of solutions from 1900 to 1930. For thousands of years, mathematics had long been regarded as the truth with non-doubtful absoluteness and uniqueness in rational thinking. In this hot discussion, people finally realized that different mathematics can coexist under different philosophical views.

7.3.1 Logicism

Gottlob Frege was a German philosopher, logician, and mathematician. He is understood by many to be the father of analytic philosophy. Frege was the early representative of logicism. His goal was to show that mathematics grows out of logic, and in so doing, he devised techniques that took him far beyond the traditional logic. Frege treated the naive set theory as a part of logic. In order to do that, he defined natural numbers with logic. We know that numbers are abstraction from concrete things. For example 3 can represents three persons, three eggs, three angles in a figure and so on. All these collections are a *class*² containing 3 elements. Which one should be used to represent natural number 3? Frege’s idea is ‘all’. All such classes that 1-to-1 correspondence can be established. This is a infinite, abstract class that defines natural number 3. Although a bit complex, it’s a great definition that free from culture limitation. No matter what language, what symbol you are using, there won’t be any ambiguities to understand number 3 through

²Frege’s work was prior to Cantor’s, he used the term ‘class’, while Cantor later used ‘set’ in German

Frege's method. This is because no symbol is needed in Frege's definition. As such, Frege managed to define number – which is the class of all classes. On top of this definition and logical laws, Frege developed his theory of natural numbers, hence established logical arithmetic. As the next step, he was going to develop all mathematics except for geometry from logic. This is what Frege wanted to achieve in his book *Basic Laws of Arithmetic*. Frege believed logical axioms were reliable and widely accepted. Once his work completed, mathematics would be “fixed on an eternal foundation”.



Gottlob Frege, 1848-1925

We know what happened next. Just during the preparation of press for *Basic Laws of Arithmetic*, Russell's letter arrived ‘in time’. Frege fell into confusions about Russell's paradox. His corner stone – using logic to define the concept of numbers – is exactly about class of all classes. Such definition directly leads to logical paradox. Frege was shocked, and finally gave up his logicism viewpoint.

Russell took over the torch of logicism. He then tried to develop mathematics from logic in another way. Russell believed all mathematics is symbolic logic. His logicism was largely influenced by Italian mathematician Giuseppe Peano. In 1900, Russell attended the International Congress of Philosophy in Paris. He wrote: “The Congress was the turning point of my intellectual life, because there I met Peano... In discussions at the Congress I observed that he was always more precise than anyone else, and that he invariably got

the better of any argument on which he embarked. As the days went by, I decided that this must be owing to his mathematical logic... It became clear to me that his notation afforded an instrument of logical analysis such as I had been seeking for years...” After he came back, Russell and Whitehead discussed the basic concepts of mathematics every day. After hard work, they finally wrote the famous *Principia Mathematica*³. The three volumes classic work about mathematical logic were published from 1910 to 1913. To solve the paradox, Russell pointed that: “An analysis of the paradoxes to be avoided shows that they all result from a kind of *vicious circle*. The vicious circles in question arise from supposing that a collection of objects may contain members which can only be defined by means of the collection as a whole.” He suggested: “Whatever involves all of a collection must not be one of the collection” and call this the “vicious-circle principle”. To carry out this restriction, Russell and Whitehead introduced ‘theory of types’.

The theory of types classified sets into levels. Individual elements, such as a person, a number, or a particular book are of type 0; The sets of elements in type 0 are of type 1; The set of elements in type 1, which are sets of sets are of type 2... Every set is of a well defined type. The objects in a proposition must belong to its type. Thus if one says a belongs to b , then b must be of higher type than a . Also one cannot speak of a set belonging to itself. Although this approach can avoid paradox, it is exceedingly complex in practice. It took 363 pages till the definition of number 1 in *Principia Mathematica*. Poincaré remarked: “eminently suitable to give an idea of the number 1 to people who have never heard it spoken of before.” The theory of types requires all works at their proper type levels, propositions about integers have to be at the level of integers; propositions about rationals have to be at the level of rationals. $n/1$ and n are at different levels, hence should not be handled in one proposition at the same time. And the common statements like “all the real numbers...” are not valid any more, as multiple types of sets are involved.

The most questionable part is about *axiom of reducibility*, *axiom of choice*, and *axiom of infinity*. In order to handle natural numbers, real numbers, and transfinite numbers,

³Russell and Whitehead gave this Latin name in honor of Isaac Newton's *Philosophiae Naturalis Principia Mathematica*.

Russel and Whitehead accepted the axiom of infinity to support the concept of infinite classes. They also accept one can chose elements from non-empty set or even infinite set to form new set. Such two arguable axioms exist in set theory too. Many people opposed to the axiom of reducibility particularly. To support mathematical induction, this axiom says any proposition at a higher level is coextensive with a proposition at type 0 level. Poincaré pointed out it was disguised form of mathematical induction. But mathematical induction is part of mathematics and is needed to establish mathematics, hence we cannot prove consistency.

Later Russell himself became more concerned: “Viewed from this strictly logical point of view, I do not see any reason to believe that the axiom of reducibility is logically necessary, which is what would be meant by saying that it is true in all possible worlds. The admission of this axiom into a system of logic is therefore a defect, even if the axiom is empirically true.”⁴

7.3.2 Intuitionism

Some mathematicians took opposite approach to build the foundation of mathematics called intuitionism. They thought mathematics was purely the result of the constructive mental activity of humans rather than the discovery of fundamental principles claimed to exist in an objective reality.

Intuitionism can be backtracked to Blaise Pascal. Leopold Kronecker was the pioneer mathematician hold intuitionism philosophy. Many world class mathematicians, including János Bolyai, Henri Lebesgue, Henri Poincaré, and Hermann Weyl support intuitionism. The founder is the Dutch mathematician Luitzen Egbertus Jan Brouwer. Brouwer was bore in 1881 in Overschie near Rotterdam, Netherlands. He entered University of Amsterdam in 1897, and soon demonstrated good mathematics capability. While still an undergraduate Brouwer proved original results on continuous motions in four dimensional space and published his result in the Royal Academy of Science in Amsterdam in 1904. Other topics which interested Brouwer were topology and the foundations of mathematics.

Influenced by Hilbert’s list of problems proposed at the Paris International Congress of Mathematicians in 1900, Brouwer put a very large effort to study typology from 1907 to 1913. The best known is his fixed point theorem, usually referred to now as the Brouwer Fixed Point Theorem. This theorem states that in the plan every continuous function from a closed disk to itself has at least one fixed point. He also extended this theorem to arbitrary finite dimension. Specially, every continuous function from a closed ball of a Euclidean space into itself has a fixed point. In 1910, Brouwer proved topological invariance of degree, then gave the rigours definition of topological dimension. Because of the outstanding contribution to topology, he was elected a member of the Royal Netherlands Academy of Arts and Sciences.

When Brouwer was a post graduate student, he was interested in the on-going debate between Russell and Poincaré on the logical foundations of mathematics⁴. His doctoral thesis in 1907 attacked the logical foundations of mathematics and marks the beginning



Alfred North Whitehead, 1861-1947

⁴Poincaré distinguished three kinds of intuition: an appeal to sense and to imagination, generalization by induction, and intuition of pure number—whence comes the axiom of induction in mathematics. The first two kinds cannot give us certainty, but, he says, “who would seriously doubt the third, who would doubt arithmetic?”^[67]

of the Intuitionist School. His views had more in common with those of Poincaré and if one asks which side of the debate he came down on then it would have with the latter. Brouwer was killed in 1966 at the age of 85, struck by a vehicle while crossing the street in front of his house.



L. E. J. Brouwer, 1881-1966

Brouwer's intuitionism came from his philosophy: mathematics is a intellectual human activity. It does not exist outside our mind. Therefore, it is independent from the real physical world. The mind recognizes basic and clear intuitions. These intuitions are not perceptual or empirical, but directly admit certain mathematical concepts, like integers. Brouwer believed that mathematical thinking is a process of intellectual construction. It builds its own world, that is independent of experience, and is limited only by the basic mathematical intuition. The basic intuitive concepts should not be understood as undefined in axiomatic theory, but should be conceived as something, as long as they are indeed useful in mathematical thinking, they can be used to understand various undefined concepts in a mathematics.

In his 1908 paper, Brouwer rejected in mathematical proofs the principle of the excluded middle, which states that any mathematical statement is either true or false, no other possibility is allowed. Brouwer denied that this dichotomy applied to infinite sets. In 1918 he published a set theory, the following year a theory of measure, and by 1923 a theory of functions, all developed without using the principle of the excluded middle.

Brouwer's constructive theories were not easy to set up since the notion of a set could not be taken as a basic concept but had to be built up using more basic notions. Because of this, Intuitionism rejected non-constructive existence proofs. For example, Euclid's proof about the existence of infinite many prime numbers was not acceptable according to Brouwer because it does not give a way to construct the prime number.

In general, intuitionism was more critical than construction in the first decades of the 20th Century. Intuitionism denied a large number of mathematical achievements, including irrational numbers, function theory, and Cantor's transfinite numbers. Many reasoning methods, like the principle of the excluded middle, were rejected. Therefore, it was strongly opposed by other mathematicians. Hilbert said: "For, compared with the immense expense of modern mathematics, what would wretched remnants mean, the few isolated results incomplete and unrelated, that the intuitionists have obtained. "

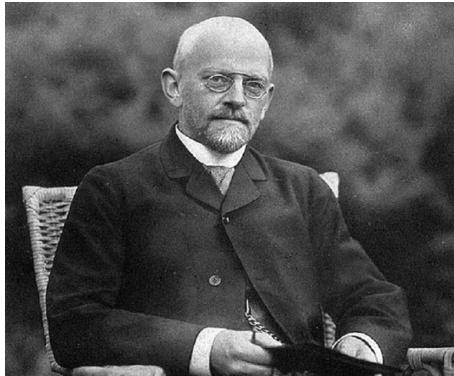


Henri Poincaré, 1854-1912

7.3.3 Formalism

The third mathematical school of thought is the Formalism led by the German mathematician David Hilbert. Hilbert was one of the most influential and universal mathematicians of the 19th and early 20th Centuries. He was born in 1862 in Königsberg, Eastern Prussia. He didn't shine at school at first, but later received the top grade for mathematics.

In 1880, Hilbert enrolled at the University of Königsberg, where he met and developed lifelong friendship with Hermann Minkowski (two years younger than Hilbert), and associate professor Adolf Hurwitz (three years elder than Hilbert). Hilbert wrote: “During innumerable walks, at times undertaken day after day, we roamed in these eight years through all the corners of mathematical science.”



David Hilbert, 1862-1943

mathematics as a world. which it is, then Hilbert was a world conqueror.” When he died, *Nature* remarked that there was scarcely a mathematician in the world whose work did not derive from that of Hilbert.[68]

Hilbert put forth a most influential list of 23 unsolved problems at the International Congress of Mathematicians in Paris in 1900. This is generally reckoned as the most successful and deeply considered compilation of open problems ever to be produced by an individual mathematician. These were the problems which he considered most significant in mathematics at that time; not isolated questions but problems of such a general character that their solution was bound to have an enormous influence on the shape of future mathematics.

Among Hilbert’s students were Hermann Weyl, the famous world chess champion Emanuel Lasker, and Ernst Zermelo. But the list includes many other famous names including Wilhelm Ackermann, Felix Bernstein, Otto Blumenthal, Richard Courant, Haskell Curry, Max Dehn, Rudolf Fueter, Alfred Haar, Georg Hamel, Erich Hecke, Earle Hedrick, Ernst Hellinger, Edward Kasner, Oliver Kellogg, Hellmuth Kneser, Otto Neugebauer, Erhard Schmidt, Hugo Steinhaus, and Teiji Takagi. From 1933, the Nazis purged many of the prominent faculty members, included Hermann Weyl, Emmy Noether and Edmund Landau. About a year later, the new Nazi Minister of Education, Bernhard Rust asked whether “the Mathematical Institute really suffered so much because of the departure of the Jews”. Hilbert replied, “Suffered? It doesn’t exist any longer, does it!” Hilbert died in 1943 at age of 81. The epitaph on his tombstone in Göttingen consists of the famous lines he spoke: “We must know. We will know.”

Hilbert’s *Foundations of Geometry* published in 1899 proposes a formal set, called Hilbert’s axioms, substituting for the traditional axioms of Euclid. It is the representative work of axiomatization. Hilbert’s approach signaled the shift to the modern axiomatic method. From 1904, Hilbert started studying the foundation of mathematics. In 1920 he proposed explicitly a research project that became known as Hilbert’s program. He wanted mathematics to be formulated on a solid and complete logical foundation. It opened the way for the development of the formalist school, one of three major schools of mathematics of the 20th century. According to the formalist, mathematics is manipulation of symbols according to agreed upon formal rules. It is therefore an autonomous activity

In 1895, invited by Felix Klein, he obtained the position of Professor of Mathematics at the University of Göttingen. He remained there 48 years for the rest of his life. During the Klein and Hilbert years, Göttingen became the preeminent institution in the mathematical world. Students and young mathematicians viewed Göttingen as the Mecca of mathematics: “Packed a bag and go to Göttingen!”

Hilbert contributed to many branches of mathematics. There are too many terms, theorems named after him, that even Hilbert himself did not know. He once asked other colleagues in Göttingen what ‘Hilbert space’ was. “If you think of

of thought.

The main goal of Hilbert's program was to provide secure foundations for all mathematics. In particular this should include:

1. **A formulation of all mathematics.** In other words all mathematical statements should be written in a precise formal language, and manipulated according to well defined rules.
2. **Completeness:** a proof that all true mathematical statements can be proved in the formalism.
3. **Consistency:** a proof that no contradiction can be obtained in the formalism of mathematics. This consistency proof should preferably use only “finitistic” reasoning about finite mathematical objects.
4. **Conservation:** a proof that any result about “real objects” obtained using reasoning about “ideal objects” (such as uncountable sets) can be proved without using ideal objects.
5. **Decidability:** there should be an algorithm for deciding the truth or falsity of any mathematical statement.

To execute his program, Hilbert initiated metamathematics, to study the mathematics itself using mathematical methods. This study produces metatheories, which are mathematical theories about other mathematical theories. Such approach actually differentiates three different mathematical systems:

1. Mathematics that is not formalized G : It's the normal mathematics, that allows classic logical reasoning. For example, applying principle of excluded middle on infinite set.
2. Formalized mathematics H : All symbols, formulas, axioms, and propositions are formalized. They are undefined concepts without any concrete meanings before explanation. Once explained, they are the concepts in G . In other words, G is the model of H , and H is formalized G . As Hilbert described: “One must be able to say at all times —instead of points, straight lines, and planes—tables, chairs, and beer mugs.” With this approach, the specific meanings and background in Euclid geometry are put aside, we only focus on the relations between the undefined concepts, which are reflected through a collection of axioms.
3. Metamathematics K : This is the metatheories to study H . All reasoning in K should be admitted intuitively. For example, without applying principle of excluded middle on infinite set.

While Hilbert and the mathematicians who worked with him in his enterprise were committed to the project, a young mathematician Gödel proved incompleteness theorems, which showed that most of the goals of Hilbert's program were impossible to achieve. We'll explain the details in later sections.

7.3.4 Axiomatic set theory

Different from the mathematical schools of logicism, intuitionism, and formalism, the members of the set-theoretic school did not formulate their distinct philosophy at the beginning, but they gradually gained adherents, and a program. This school today earns as much as supporters as the other three we introduced.

The set-theoretic school can be traced back to Cantor and Dedekind's work. Although both were primarily concerned with infinite sets, they found by establishing the concept of natural numbers on basis of set, all of mathematics could then be derived. When Russell's paradox was found at the centre of Cantor's set theory, some mathematicians believed that the paradox was due to the informal introduction of sets. Cantor's set theory is often described today as 'naive set theory'. Hence the set theoretic thought that a carefully selected axiomatic foundation would remove the paradoxes of set theory. Just as the axiomatization of geometry and of the number system had resolved logical problems in those areas. German mathematician Ernst Zermelo first took the axiomatization approach in set theory in 1908.

Zermelo also believed the paradoxes arose because Cantor had not restricted the concept of a set. He therefore stressed with clear and explicit axioms to clarify what is meant by a set, and what properties sets should have. In particular, he wanted to limit the size of possible sets. He had no philosophical basis but sought only to avoid the contradictions. His axiom system contained the undefined fundamental concepts of set and the relation of one set being included in another. These and the defined concepts were to satisfy the statements in the axioms. No properties of sets were to be used unless granted by the axioms. In his system, the existence of infinite sets, the operations as the union of sets, and the formation of subsets were provided as axioms. Zermelo also used the axiom of choice[4].



(a) Ernst Zermelo, 1871-1953



(b) Abraham Fraenkel, 1891-1965

Zermelo's system of axioms was improved by Abraham Fraenkel in 1922. Zermelo did not distinguish a set property and the set itself, they were used as synonymous. The distinction was made by Fraenkel. The system of axioms used mostly common by set theorists is known as Zermelo-Fraenkel system, abbreviated as ZF system. They both saw the possibility of refined and sharper mathematical logic available in their time, but did not specify the logical principles, which they thought were outside of mathematics, and could be confidently applied as before[4].

Zermelo provided 7 axioms in his 1908 paper. Then in 1930, Fraenkel, Skolem, and Von Neumann suggested to add another two axioms. These axioms are as below:

1. **Axiom of extensionality:** Two sets are equal if they have the same elements. For set A and B , if $A \subseteq B$ and $B \subseteq A$, then $A = B$.
2. **Empty set:** The empty set exists.

3. **Axiom schema of separation:** Also known as axiom schema of specification. Any property that can be formalized in the language of the theory can be used to define a set. For set S , if proposition $p(x)$ is defined, then there exists set $T = \{x|x \in S, p(x)\}$.
4. **Axiom of power set:** One can form the power set (the collection of all subsets of a given set) of any set. This process can be repeated infinitely.
5. **Axiom of union:** The union over the elements of a set exists.
6. **Axiom of choice:** abbreviated as AC.
7. **Axiom of infinity:** There exists a set Z , containing empty set. For any $a \in Z$, then $\{a\} \in Z$. This axiom ensures infinite set exists.
8. **Axiom schema of replacement:** This axiom is introduced by Fraenkel in 1922. For any function $f(x)$ and set T , if $x \in T$, and $f(x)$ is defined, there exists a set S , that for all $x \in T$, there is a $y \in S$, such that $y = f(x)$. It says that the image of a set under any definable function will also fall inside a set.
9. **Axiom of regularity:** Also known as axiom of foundation. It was introduced by Von Neumann in 1925. x does not belong to x .

As such, set theory was abstracted to a axiomatic system. Set turned to be an undefined concept that satisfies these axioms. They do not permit ‘all inclusive set’, hence avoid the paradox, and fixed the defects in naive set theory. However, there were still debate about which axioms were acceptable, particularly the axiom of choice is arguable.



Figure 7.12: Banach-Tarski paradox: a solid ball can be decomposed and put back together into two copies of the original ball.

In 1924, Polish mathematicians Stefan Banach and Alfred Tarski proved a theorem called Banach-Tarski paradox⁵. This theorem states that, if accept axiom of choice, then for a solid ball in 3 dimensional space, there exists a decomposition of the ball into a finite number of disjoint subsets, which can then be put back together in a different way to yield two identical copies of the original ball. Indeed, the reassembly process involves only moving the pieces around and rotating them without changing their shape. Banach and Tarski was going to reject axiom of choice through this theorem. However, their proof looked so natural, that mathematicians tend to consider it only reflects the counter intuitive fact about axiom of choice. Some set-theorists insist not to including axiom of choice, such axiomatic set theory is called ZF system, while the one included axiom of choice is called ZFC system. We introduced the interesting relation between axiom of choice and continuum hypothesis in previous chapter.

⁵Also known as Hausdorff-Banach-Tarski theorem, or ‘Doubling sphere paradox’.

7.4 Gödel's incompleteness theorems

By 1930, there had been four separated, distinct, and more or less conflicting approaches about mathematics foundation. Their supporters adherence to their own mathematical schools. One could not say a theorem is correctly proven, because by 1930, he had to add by whose standard, it was proven correct. The consistency of mathematics, which motivated these new approaches was not settled at all except if one argue that it's the human intuition guarantees consistency[4]. Hilbert was still planning his project optimistically to prove the completeness and consistency of mathematics. All these were ended up by a young mathematician and logician, Gödel.



Kurt Gödel, 1906-1978

Kurt Gödel was born in 1906 in Brünn, Austria-Hungary Empire (now Brno, Czech Republic) into a German family. He had quite a happy childhood. He had rheumatic fever and recovered at age 6. However, 2 years later when read medical books about the illness, he learnt that a weak heart was a possible complication. Although there is no evidence that he did have a weak heart, Kurt became convinced that he did, and concern for his health became an everyday worry for him. In his family, young Kurt was known as "Mr. Why" because of his insatiable curiosity.

In 1924, Gödel entered the University of Vienna. he hadn't decided whether to study mathematics or theoretic physics until he learnt number theory. He decided to take mathematics as his main subject in 1926. Gödel was also interested in philosophy,

and took part in seminars about mathematical logic. The exploration of philosophy and mathematics set Gödel's life course.

In 1929, at the age of 23, he completed his doctoral dissertation. In it, he established his completeness theorem regarding the first-order predicate calculus. He was going to further study Hilbert's program to prove the completeness and consistency of mathematics in finite steps. However, he soon developed an unexpected result. In 1930 Gödel attended the Second Conference on the Epistemology of the Exact Sciences, held in Königsberg. Here he delivered his first incompleteness theorem, and soon, proved the second incompleteness theorem.

Gödel worked at University of Vienna from 1932. In 1933, Adolf Hitler came to power in Germany, the Nazis rose in influence in Austria academy over the following years. In 1936, Moritz Schlick, whose seminar had aroused Gödel's interest in logic, was assassinated by one of his former students. This triggered "a severe nervous crisis" in Gödel. He developed paranoid symptoms, including a fear of being poisoned. After the world war II broken out, Gödel accepted the invitation from the Institute for Advanced Study in Princeton, New Jersey. and moved to US. To avoid the difficulty of an Atlantic crossing, the Gödels took the Trans-Siberian Railway to the Pacific, sailed from Japan to San Francisco, then crossed the US by train to Princeton. He met Albert Einstein in Princeton, who became a good friend. They were known to take long walks together to and from the Institute for Advanced Study. Einstein's death in 1955 impacted him a lot. In his later life, logician and mathematician Wang Hao was his close friend and commentator.

Gödel's married Adele Nimburksy, whom he had known for over 10 years on September 20th, 1938. Their relationship had been opposed by his parents on the grounds that she was a divorced dancer, six years older than Gödel. Later in his life, Gödel suffered periods

of mental instability and illness. He had an obsessive fear of being poisoned; he would eat only food that Adele prepared for him. Late in 1977, she was hospitalized for six months and could subsequently no longer prepare her husband's food. In her absence, he refused to eat, eventually starving to death. He weighed 29 kilograms (65 lb) when he died. His death certificate reported that he died of "malnutrition and inanition caused by personality disturbance" in 1978. Because of the outstanding contributions in logic, he was regarded as the greatest logician since Aristotle.

In 1931, Gödel published his paper titled *On Formally Undecidable Propositions of Principia Mathematica and Related Systems*. Where 'Principia Mathematica' is the work of Russell and Whitehead. In that article, he proved for any computable axiomatic system that is powerful enough to describe the arithmetic of the natural numbers:

1. If a system is consistent, it cannot be complete.
2. The consistency of axioms cannot be proved within their own system.

For any consistency formal system, Gödel gave an undecidable statement G , that can neither be proved nor disproved. This theorem is called Gödel's first incompleteness theorem. It tells that consistent formalized system is incomplete. As far as the system is powerful enough to contain arithmetic of natural numbers, there will be problems exceed it. One may ask, since G is undecidable, what if accept G or G 's negation as an additional axiom, to obtain a more powerful system? However, Gödel soon proved the second incompleteness. It tells that if a formal system containing elementary arithmetic, then the consistency cannot be proved within its own system. Whether accept or reject G , the new system is still incomplete. There always exists undecidable statement in the higher level.

In Euclidean geometry for example, we can exclude the fifth postulate, to obtain the axiomatic system with the first four postulates. However, we cannot prove the fifth postulate true or false. We know that whether accept or reject the fifth postulate gives consistent geometry – Euclidean geometry and varies of non-Euclidean geometries respectively. In axiomatic set theory ZF system, we cannot prove axiom of choice true or false. Accepting it gives the consistent ZFC system; while rejecting it gives another consistent system. After add the axiom of choice to establish ZFC system, we cannot prove the continuum hypothesis true or false in ZFC. Accepting continuum hypothesis gives a consistent system; while rejecting continuum hypothesis gives another consistent system.

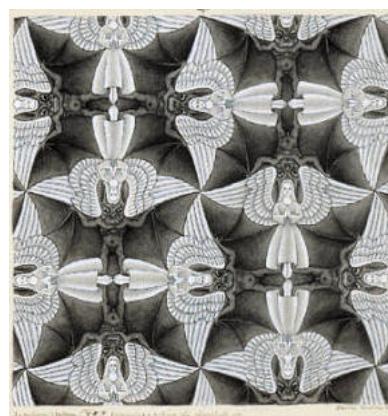


Figure 7.14: Escher, Angles and Devils, 1941

Gödel's first and second incompleteness theorems ended a half-century of attempts, beginning with the work of Frege and culminating in Principia Mathematica and Hilbert's formalism, to find a set of axioms sufficient for all mathematics. Even the elementary arithmetic system is consistent, such consistency cannot be proved within itself. As mathematician André Weil said: “God exists since mathematics is consistent, and the Devil exists since we cannot prove it.”[8]

7.5 Proof sketch for the first incompleteness theorem

According to Hilbert's project, the first step is to formalize all the mathematics into a system. Then study it with metamathematics. In order to do that, we need represent every mathematical branch as a formal system, which only contains finite many axioms, then prove it is complete and consistent. Among them, the most fundamental system is the arithmetic of natural numbers. Many mathematical branches are isomorphic to it. In previous chapter, we see how to extend natural numbers to define integers, rationals, and real numbers. By establishing the correspondence between points and real numbers, we can further treat the Euclidean geometry as arithmetic through coordinate geometry.

7.5.1 Formalize the system

Here we use the method and terms, that Douglas Hofstadter wrote in his popular book *Gödel, Escher, Bach: An Eternal Golden Braid* to introduce the proof sketch. Gödel's proof also starts from modeling a formal system. We call this system “Typographical Number Theory”, abbreviated as TNT. It happens to be the same abbreviation of Trinitrotoluene, a powerful explosive. Hofstadter intended to give this name to indicate it's powerful enough to destroy the building itself. TNT formalizes the number theory in natural language into a series of typographical strings. Although sounds complex, we can realize it step by step on top of the Peano Axioms we introduced in chapter 1. As the first thing, we need define numbers. According to Peano's axioms, zero is natural number, every natural number has its successor, we can define the typographical string for numbers as the following:

zero	0
one	S0
two	SS0
three	SSS0
...	...

Where 'S' means successor, two letters, 'SS' mean the successor of a successor. A hundred 'S's and a '0' are the 100 times successor of zero, which is the natural number 100. Although it is very long, the rule itself is simple enough. With natural numbers being defined, we need next define variables. To make the system as simple as possible, we only use 5 typographical letters a, b, c, d, e . When need more variables, we can simply add primes like a', a'', a''' . Next we need '+' for addition, '·' for multiplication, and the parenthesis to control the arithmetic orders. To formalize the proposition we need '=', ' \neg ' for negation, and \rightarrow for implication. Here are some examples of formal propositions (no matter their truth or falsehood):

- one plus two equals four: $(S0 + SS0) = SSS0$
- two plus two is not equal to five: $\neg(SS0 + SS0) = SSSS0$

- if one equals to zero, then zero equals one: $(S0 = 0) \rightarrow (0 = S0)$

A proposition can have free variables, for example:

$$(a + SS0) = SSS0$$

It means a plus 2 equals 3. Obviously the value of a determines if this proposition is true or false. Therefore, we need universal quantifier \forall , existential quantifier \exists , and colon ‘:’ to indicate quantifier scope. The following proposition:

$$\exists a : (a + SS0) = SSS0$$

means, there exists a , such that a plus 2 equals 3. Here is another example:

$$\forall a : \forall b : (a + b) = (b + a)$$

It is exactly the commutative law of addition for natural numbers. When remove the quantifier for a , it changes to:

$$\forall b : (a + b) = (b + a)$$

This is a open formula, since a is free. It expresses a unspecified number a commutes with all numbers b . It may or may not be true. In order to compose propositions, we need logical conjunction (and) \wedge , logical disjunction (or) \vee . Although there are few symbols, TNT is very expressive. Here are some examples:

2 is not the square of any natural numbers: $\neg \exists a : (a \cdot a) = SS0$

Fermat's last theorem holds when n equals 3: $\neg \exists a : \exists b : \exists c : ((a \cdot a) \cdot a) + ((b \cdot b) \cdot b) = ((c \cdot c) \cdot c)$

We defined typographical symbols to express propositions so far. To construct TNT system, we also need axioms and reasoning rules.

Axioms and reasoning rules

Following Peano's axioms, we define 5 axioms for the TNT system:

1. $\forall a : \neg Sa = 0$, this axiom states that, zero is not the successor of any number;
2. $\forall a : (a + 0) = 0$, this axiom states that, any number plus 0 equals itself;
3. $\forall a : \forall b : (a + Sb) = S(a + b)$, this axiom defines the addition for natural numbers;
4. $\forall a : (a \cdot 0) = 0$, this axiom states that, any number multiplies zero equals zero;
5. $\forall a : \forall b : (a \cdot Sb) = ((a \cdot b) + a)$, this axiom defines multiplication for natural numbers.

Next we establish reasoning rules. For example, from axiom 1, that 0 is not the successor of any number, we want to deduce a special case, that 1 is not the successor of 0. In order to do this, we introduce the rule of specification:

Rule of specification: Suppose u is a variable occurs in string x . If $\forall u : x$ is a theorem, then x is also a theorem, and any replacement of u in x wherever it occurs, also gives a theorem.

There is a restriction, the term that replaces u , must not contain any variables that is quantified in x . And the replacement should be consistent. The opposite rule to specification is the rule of generalization. It allows us to add the universal quantifier before a theorem.

Rule of generalization: Suppose x is a theorem, u is a free variable in it. Then $\forall u : x$ is a theorem.

For example, $\neg S(c + S0) = 0$ means there is no such a number, that plus 1, then take the successor gives 0. We can generalize it as: $\forall c : \neg S(c + S0) = 0$.

The next rule tells us how to convert universal and existential quantifiers.

Rule of interchange: Suppose u is a variable, then the string $\forall u : \neg$ and $\neg \exists u :$ are interchangeable.

When applying this rule to axiom 1 for example, it transforms to $\neg \exists a : Sa = 0$. The next rule allows us to put a existential quantifier before a string.

Rule of existence: Suppose a term appears once or multiply in a theorem, then it can be replaced with a variable, and add a corresponding existential quantifier in front.

Use axiom 1 for example again: $\forall a : \neg Sa = 0$, we can replace 0 with a variable b , and add the corresponding existential quantifier to give: $\exists b : \forall a : \neg Sa = b$. It states that, there exists a number, such that any natural number is not its successor.

We next consider the symmetry and transitivity for equality, and define rules. Let r, s, t all stand for arbitrary terms.

Rules of quality:

- Symmetry: if $r = s$ is a theorem, then $s = r$ is also theorem;
- Transitivity: if $r = s$ and $s = t$ are theorems, then $r = t$ is also theorem.

To add or remove the successorship S , we define below rules.

Rules of successorship:

- Add: If $r = t$ is theorem, then $Sr = St$ is a theorem;
- Drop: If $Sr = St$ is theorem, then $r = t$ is a theorem.

So far, the TNT system is very powerful, we can construct complex theorems with it.

Exercise 7.2

1. Translate Fermat's last theorem into a TNT string.
2. Prove the associative law of addition with TNT reasoning rules.

Incompleteness of TNT

With the axioms and reasoning rules in TNT system, we can prove a series of theorems easily:

$$\begin{array}{rcl}
 (0 + 0) & = & 0 \\
 (0 + S0) & = & S0 \\
 (0 + SS0) & = & SS0 \\
 (0 + SSS0) & = & SSS0 \\
 \dots & & \dots
 \end{array}$$

From axiom 2, we can deduce the first theorem by replacing a with 0; on top of this theorem, and use axiom 3, we can obtain the second theorem; every theorem can be deduced from the previous one. Observe this pattern, we immediately ask, why can't we summarize them to a theorem?

$$\forall a : (0 + a) = a$$

Note it is different from axiom 2. Unfortunately, we can't reason this theorem with all the rules in TNT so far. We may want to add an additional rule: if all this series

of strings are theorems, then the universally quantified string which summarizes them is also a theorem. However, only human that outside TNT has this insight. It's not a valid rule for the formal system.

Lack of such summarize capability indicates TNT is incomplete. Accurately speaking, a system with this kind of 'defect' is called ω incomplete. Where ω is the cardinal of countable infinite set introduced in previous chapter. We say a system is ω incomplete if all the strings in a series are theorems, but the universal quantified summarizing string is not a theorem. Incidentally, the negation of the summarizing string:

$$\neg \forall a : (0 + a) = a$$

is not a theorem of TNT too. It means the string is undecidable within TNT system. The capability of TNT is not enough to determine this string is theorem or not. It just likes the same situation, that with only the first four postulates in Euclidean geometry, the fifth postulation is undecidable. We can either accept to add the fifth postulation to obtain Euclidean geometry, or reject to add its negation to obtain non-Euclidean geometry. Similarly, we can either add this string or its negation to TNT to construct different formal systems.

It looks a bit counter intuitive if we chose the negation as theorem: zero plus any number does not equal to this number any more. It's quite different from the arithmetic of natural numbers that familiar to us. It exactly reminds us, the concept in a formal system is undefined. We give it the meaning of addition for natural numbers only for the purpose of easy understanding.

The ω incompleteness of TNT tells us, we are missing an important rule – you may have already thought of – Peano's fifth axiom that corresponding to mathematical induction. Let's add this last piece of tile to the puzzle.

Rule of induction: Suppose u is a variable in string X , denoted as Xu . If it is a theorem when replace u with 0, and $\forall u : Xu \rightarrow XSu$. It means if X is a theorem for u , so as it is when replace u to Su . Then $\forall u : Xu$ is also a theorem.

With mathematical induction supported, TNT system now has the same capability as Peano's arithmetic.

Exercise 7.3

1. Prove that $\forall a : (0 + a) = a$ with the newly added rule of induction.

7.5.2 Gödel numbering

One critical step Gödel took was to introduce Gödel numbering. TNT system is powerful enough to mirror other formal system, is it possible to mirror TNT by itself? What Gödel thought is to 'arithmetize' the reasoning rules. To do this, he assigned all symbols with a number.

Axiom 1 is translated to such numerals:

$$\begin{array}{ccccccccc} \forall & a & : & \neg & S & a & = & 0 \\ 626 & 262 & 636 & 223 & 123 & 262 & 111 & 666 \end{array}$$

The numbering scheme is not unique. It does not matter if one assigns different numbers. With Gödel numbering, every TNT string can be represented as a number (although it can be a very big number). The problem is in the other direction: given any number, can we determine if it represents a TNT theorem? We know the first five TNT numbers, which represent the five axioms. With the TNT reasoning rules, we can construct infinite many TNT numbers from these five numbers. Atop of this, we introduce a number theory predication:

a is a TNT number.

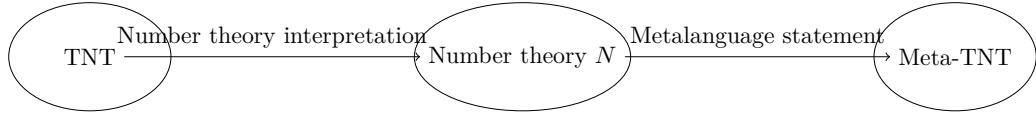
symbol	number	symbol	number
0	666	S	123
=	111	+	112
.	236	(362
)	323	<i>a</i>	262
'	163	\wedge	161
\vee	616	\rightarrow	633
\neg	223	\exists	333
\forall	626	:	636

Table 7.1: A Gödel numbering to TNT

For example, 626,262,636,223,123,262,111,666 is a TNT number (We add commas to make it easy to read), it represents axiom 1. Its negation form is:

$\neg a$ is a TNT number.

For example, 123,666,111,666 is not a TNT number. It means we can replace a by a string of 123666111666 Ss and a 0. This huge string actually means: $S0 = 0$ is not a TNT theorem. TNT system can really speak about itself. It is not an accidental feature, but because of the fact that all formal systems are isomorphic to number theory N . Hence we formed a circle: A TNT string has its interpretation in number theory N , while the statement in N can have a second meaning, which is the metalanguage interpretation about TNT.

Figure 7.15: $\text{TNT} \rightarrow \text{number theory } N \rightarrow \text{Meta-TNT}$

7.5.3 Construct the self-reference

The last step in Gödel's proof is to construct a self-reference. It's such a TNT string, called G , which is about itself:

G is not a theorem of TNT.

Now we can detonate TNT. Whether G is a theorem of TNT or not? If G is a theorem, then it states the truth that " G is not a theorem". Here we see the power of self-reference. As a theorem, G can't be falsehood. Because we assume that TNT will not treat falsehood as theorem, we have to draw the conclusion that G is not a theorem. While known that G is not a theorem, we should admit that G is truth. It reflects that TNT does not meet our expectation – we found a string, it states the truth, but is not a theorem. Further, considering the fact that G has its number theory interpretation, which is a statement of arithmetic property about natural numbers. From the reasoning outside TNT, we can confirm this statement is true, and the string is not a theorem of TNT. However, when we ask TNT whether this string is true, TNT could never say 'Yes' or 'No'.

G is that undecidable proposition. This is the sketch of the proof of Gödel's first incompleteness theorem.

7.6 Universal program and diagonal argument

What does Gödel's incompleteness theorems mean to programming? We have an exactly isomorphic problem of programming. In order to see it, let us start from a formal computer programming language. This language supports primitive recursive function. The so called primitive recursive function is a kind of number theory functions, they can map from natural numbers to natural numbers, and follow the 5 axioms:

1. Constant function: The 0-ary constant function 0 is primitive recursive;
2. Successor function: The 1-ary successor function $S(k) = k + 1$ is primitive recursive;
3. Projection function: The n -ary function, which accepts n arguments, and returns its i -th argument P_i^n is primitive recursive;
4. Composition: The result of finite many times composition of primitive functions is still primitive recursive;
5. Primitive recursion:

$$\begin{cases} h(0) = k \\ h(n + 1) = g(n, h(n)) \end{cases}$$

We say h is computed from g through primitive recursion. It can be extended to the case of multiple arguments.

The programming language that supports basic arithmetic like addition, subtraction, if-then branches, equal, less than prediction, and bounded loop is called primitive recursive programming language. The bounded loop means the number of loops are determined beforehand. It can be loop without go-to statement, or for-loop that does not allow to alter the loop variable inside it. However, it cannot be while loop, or repeat-until loop. Because of these limitations, all primitive recursive programs must halt.

An important property of primitive recursive function is that, all primitive recursive programs are recursively enumerable. Suppose we can list all primitive recursive functions that with one input and one output, store them in a infinite big library. We can number each of them⁶, from 0, 1, 2, 3, ... And denote these programs as $B[0], B[1], B[2], \dots$. For the i -th program, when input n , it gives result $B[i](n)$.

Now, we construct a special function $f(n)$, when input n , its output is what the n -th program's output for n plus 1. Like this:

$$f(n) = Bn + 1$$

Such f is definitely computable. Now we ask whether f is a primitive recursive program stored in our library? If yes, suppose its number in the library is m . According to our numbering method, when input m to the m -th program, the result should be Bm. However, from the definition of f , its output should be $f(m) = Bm + 1$. These two results are not equal obviously. The contradiction proves there exists computable function that is not primitive recursive.

Our proof uses the same method as Cantor's diagonal argument in previous chapter. We have to relax the bounded loop limitation to make the programming language more powerful. To do that, we allow go-to statement in the loop to jump out; the loop variable can be altered inside; we introduced while loop, repeat-until loop; and general

⁶One numbering method is to concatenate all ASCII codes of the program to form a number, then sort them from less to big. Since each program is unique, their ASCII codes are different

recursive functions. As such, we extend from primitive recursive function to *total recursive function*. This kind of programming language is called Turing-complete language. Most computer programming languages are Turing-complete, which are isomorphic to the formal systems like arithmetic of natural numbers. However, there is defect in Turning-complete language, as we can construct the primitive recursive function of Turing halting problem. It proves there exists incomputable problem. Is it possible to relax the limitation further, empower Turning-complete language, to design a universal program? The answer is no. Turning-complete is at the highest level, that reaches to the limitation of formal system. There is no other limitation can be relaxed any more. Gödel's incompleteness theorems tell us, once the formal system is powerful enough to include arithmetic of natural numbers, there must be undecidable problem in it.

7.7 Epilogue

As human being, our rational thinking is great. It can lead us back to thousands of years, and talk to ancient sages; it can send us across the universe, and step onto the unreachable planet; it can foresee elementary parcels that are invisible to eyes; it can break through intuition and reach to high dimension magic world. Looking up the sky, through the clouds, dust, and stars, we feel the insignificant of ourselves. We are just passing passengers in the long river of time, just like a drop in the vast sea.

The problem we introduce in this chapter, is essentially about ourselves as human. Does there exist the boundary of our rational thinking? Are we swallowing ourselves along a strange circle? These are questions everybody will ask at the era of artificially intelligence. People are making machine isomorphic to people, making the huge machinery computing isomorphic to brain and rational thinking. Just as Escher illustrated in his *Dragon*, he tried best to break free from the two-dimension picture. He has found the two slits in the paper. His head and neck pokes through one slit, and the tail through the other, with the head biting the tail, he wants to pull himself out to the three-dimension world. As the observer, we clearly know all still happen on the two-dimension paper, the dragon's hard work is in vain. All these are “like a dream, an illusion, a bubble and a shadow, like dew and lightning.”

Even it was about a hundred years ago, the hot debate about mathematical foundation, the genius proof given by Gödel still have their practical significance today. As human beings, we are humbly in awe of the nature, the universe, our ancestors, and ourselves.



Figure 7.16: Escher, *Dragon*, 1952

Appendix

Answers

.1 Preface

1. Implementing a tick-tack-toe game is a classic programming exercise. It's trivial to test if the sum of three numbers is 15. Please use this point to implement a simplified tick-tack-toe program that never loses a game.

We use Lo Shu square to model tick-tack-toe game as they are isomorphic. We setup two sets X, O to represent the cells each player has occupied. For the game in the preface, it starts with $X = \emptyset, O = \emptyset$, and ends with $X = \{2, 5, 7, 1\}, O = \{4, 3, 8, 6\}$. To determine if either player wins, we write a program to check if there are any 3 elements add up to 15.

There are two methods to do this checking. One is to list all the rows, columns, and diagonals. There 8 tuples in total as: $\{\{4, 9, 2\}, \{3, 5, 7\}, \dots, \{2, 5, 8\}\}$. Then check if anyone is the subset of the cells occupied by the player. The other method is interesting. Suppose a player covers cells $X = \{x_1, x_2, \dots, x_n\}$, sorted in ascending order as they appear in the Lo Shu square. We can first pick x_1 , then use two pointers l, r point to the next element and the last element. If the sum of three numbers $s = x_1 + x_l + x_r$ equals to 15, it means the player lines up and win. If it is less than 15, because the elements are in ascending order, we can increase the left pointer l by 1 and examine again. otherwise, if it is greater than 15, then we decrease the right pointer r by 1. When the left and right pointers meet, then no tuple add up to 15 when fixing x_1 . We next pick x_2 and do the similar things. In worst case, after $(n - 2) + (n - 3) + \dots + 1$ checks, we know whether a player wins or not.

```
def win(s):
    n = len(s)
    if n < 3:
        return False
    s = sorted(s)
    for i in range(n - 2):
        l = i + 1
        r = n - 1
        while l < r:
            total = s[i] + s[l] + s[r]
            if total == 15:
                return True
            elif total < 15:
                l = l + 1
            else:
                r = r - 1
    return False
```

Given X and O , we can test if the game is in end state – either a player wins or draw with all 9 cells being occupied. Next we use the classic **min-max** method in

AI to realize the game. For each state, we evaluate its score. One player tries to maximize the score, while the opponent tries to minimize it. For a draw state, we evaluate the score as zero; if player X wins, we give it score 10; and player O wins, we give it a negative score -10. These numbers are arbitrary. They can be any other values without impact the correctness.

```
WIN = 10
INF = 1000

# Lo Shu magic square
MAGIC_SQUARE = [4, 9, 2,
                 3, 5, 7,
                 8, 1, 6]

def eval(x, o):
    if win(x):
        return WIN
    if win(o):
        return -WIN
    return 0

def finished(x, o):
    return len(x) + len(o) == 9
```

For any game state, we let the program explore ahead till an end state (one side wins, or draw). The explore method, is to exhaustive cover all unoccupied cells, then turn to the opponent player, consider what is the best move to beat the other. For all candidate moves, select the highest score for player X , or select the lowest score for player O .

```
def findbest(x, o, maximize):
    best = -INF if maximize else INF
    move = 0
    for i in MAGIC_SQUARE:
        if (i not in x) and (i not in o):
            if maximize:
                val = minmax([i] + x, o, 0, not maximize)
                if val > best:
                    best = val
                    move = i
            else:
                val = minmax(x, [i] + o, 0, not maximize)
                if val < best:
                    best = val
                    move = i
    return move
```

The **min-max** search is recursive. In order to win fast, we take the number of steps into account on top of the state score. For player X , we deduce the score from the recursion depth; while for player O , we add the depth to the score.

```
def minmax(x, o, depth, maximize):
    score = eval(x, o)
    if score == WIN:
        return score - depth
    if score == -WIN:
        return score + depth
    if finished(x, o):
        return 0 # draw
    best = -INF if maximize else INF
    for i in MAGIC_SQUARE:
        if (i not in x) and (i not in o):
```

```

if maximize:
    best = max(best, minmax([i] + x, o, depth + 1, not maximize))
else:
    best = min(best, minmax(x, [i] + o, depth + 1, not maximize))
return best

```

We obtain a program that will never lose to human player. It uses the Lo Shu square on the back end essentially.

```

def board(x, o):
    for r in range(3):
        print "-----"
        for c in range(3):
            p = MAGIC_SQUARE[r*3 + c]
            if p in x:
                print "|X",
            elif p in o:
                print "|O",
            else:
                print "| ",
        print "|"
    print "-----"

def play():
    x = []
    o = []
    while not (win(x) or win(o) or finished(x, o)):
        board(x, o)
        while True:
            i = int(input("[1..9]==>"))
            if i not in MAGIC_SQUARE or MAGIC_SQUARE[i-1] in x or
               MAGIC_SQUARE[i-1] in o:
                print "invalid move"
            else:
                x = [MAGIC_SQUARE[i-1]] + x
                break
        o = [findbest(x, o, False)] + o
    board(x, o)

```

.2 Natural Numbers

1. Define 1 as the successor of 0, prove $a \cdot 1 = a$ holds for all natural numbers;

We first use mathematical induction to prove $0 + a = a$ (refer to *Appendix - Proof of commutative law of addition*). Then:

$$\begin{aligned}
 a' \cdot 1 &= a' \cdot 0' && \text{1 as the successor of 0} \\
 &= a' \cdot 0 + a' && \text{2nd rule of multiplication} \\
 &= 0 + a' && \text{1st rule of multiplication} \\
 &= a' && \text{proved previously}
 \end{aligned}$$

2. Prove the distributive law for multiplication;

Proof. We can prove the left side distribution law $c(a+b) = ca+cb$ by mathematical induction. First when $b = 0$:

$$\begin{aligned}
 c(a+0) &= ca && \text{1st rule of addition} \\
 &= ca + 0 && \text{reverse of 1st rule of addition} \\
 &= ca + c0 && \text{reverse of 1st rule of multiplication}
 \end{aligned}$$

Next suppose $c(a + b) = ca + cb$ holds, we need prove $c(a + b') = ca + cb'$

$$\begin{aligned}
 c(a + b') &= c(a + b)' && \text{2nd rule of addition} \\
 &= c(a + b) + c && \text{2nd rule of multiplication} \\
 &= ca + cb + c && \text{induction hypothesis} \\
 &= ca + (cb + c) && \text{associative law of addition} \\
 &= ca + cb' && \text{reverse of 2nd rule of multiplication}
 \end{aligned}$$

□

3. Prove the associative and commutative laws for multiplication.

We only prove the associative law for multiplication $(ab)c = a(bc)$. For the commutative law, we provide a proof outline.

Using mathematical induction, when $c = 0$:

$$\begin{aligned}
 (ab)0 &= 0 && \text{1st rule of multiplication} \\
 &= a0 && \text{reverse of 1st rule of multiplication} \\
 &= a(b0) && \text{reverse of 1st rule of multiplication}
 \end{aligned}$$

Next suppose $(ab)c = a(bc)$, we need prove $(ab)c' = a(bc')$

$$\begin{aligned}
 (ab)c' &= (ab)c + ab && \text{2nd rule of multiplication} \\
 &= a(bc) + ab && \text{induction hypothesis} \\
 &= a(bc + b) && \text{distribution law proven above} \\
 &= a(bc') && \text{reverse of 2nd rule of multiplication}
 \end{aligned}$$

To prove the commutative law of multiplication, we take three steps, all with mathematical induction. First we prove $1a = a$, then prove the right side distribution law $(a + b)c = ac + bc$, finally, prove the commutative law.

4. How to verify $3 + 147 = 150$ with Peano axioms?

Let us first see the classic proof of $2 + 2 = 4$:

$$\begin{aligned}
 2 + 2 &= 0'' + 0'' && 2 \text{ is the successor of successor of } 0 \\
 &= (0'' + 0')' && \text{2nd rule of addition} \\
 &= ((0'' + 0)')' && \text{2nd rule of addition} \\
 &= ((0'')')' && \text{1st rule of addition} \\
 &= 0''' = 4 && 4 \text{ times successor of } 0
 \end{aligned}$$

It will be too long to prove $3 + 147 = 150$ in this way. One method is to apply the commutative law of addition, then prove $147 + 3 = 150$; another method is to use the mathematical induction to prove $3 + a = a''''$

5. Give the geometric explanation for distributive, associative, and commutative laws of multiplication.

See figure 17

6. Define square for natural number $()^2$ with *foldn*;

We can define square from the iterative relation $(n + 1)^2 = n^2 + 2n + 1$

$$()^2 = 2nd \cdot \text{foldn} (0, 0) h$$

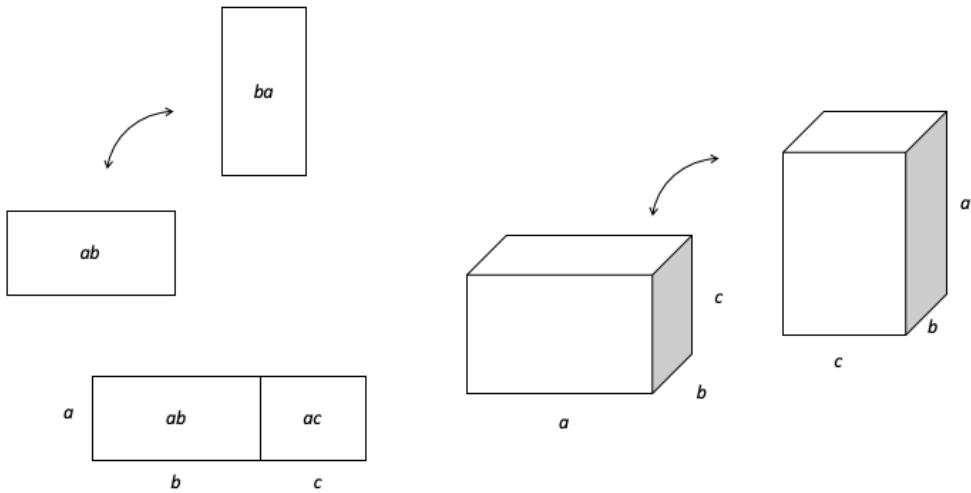


Figure 17: Geometric explanation for arithmetic laws

Where h accepts a pair (i, s) , containing number i and its square s . It increases i by 1, then uses the iterative relation to calculate the next square.

$$h(i, s) = (i + 1, s + 2i + 1)$$

7. Define $()^m$ with *foldn*, which gives the m -power of a natural number;

One simple method is to re-use the definition of $m^() = \text{foldn}(1, (\cdot \cdot m))$ in chapter 1:

$$()^m = 2nd \cdot \text{foldn}(0, 0) h$$

where

$$h(i, b) = (i + 1, (i + 1)^m)$$

It looks a bit strange, as all the intermediate results are dropped. Another method is to leverage Newton's binomial theorem:

$$(n + 1)^m = n^m + \binom{m}{1} n^{m-1} + \dots + \binom{m}{m-1} n + 1$$

We can establish the iterative relation from it:

$$(n)^m = 2nd(\text{foldn}(1, 1) h(n - 1))$$

where

$$h(i, x) = (i + 1, C \cdot X)$$

The $C \cdot X$ is the dot product between binomial coefficients and the powers: $C \cdot X = \sum c_j x_j$. The powers can be calculated by repeatedly dividing x by i , and the binomial coefficients can be iterated from Pascal triangle. Below is an example program that put them together:

```

exp m n = snd $ foldn (1, 1) h (n - 1) where
  cs = foldn [1] pascal m
  h (i, x) = (i + 1, sum $ zipWith (*) cs xs) where
    xs = take (m + 1) $ iterate (`div' i) x

pascal = gen [1] where
  gen cs (x:y:xs) = gen ((x + y) : cs) (y:xs)
  gen cs _ = 1 : cs

```

8. Define sum of odd numbers with *foldn*, what sequence does it produce?

$1 + 3 + 5 + \dots$ can be defined with *foldn* as $2nd \cdot foldn (1, 0) h$, where:

$$h (i, s) = (i + 2, s + i)$$

As shown in the figure below this exercise in chapter 1, the sum of odd numbers is always a square number.

9. There is a line of holes in the forest. A fox hides in a hole. It moves to the next hole every day. If we can only check one hole a day, is there a way to catch the fox? Prove this method works. What if the fox moves more than one hole a day?

No matter which hole the fox hides in, we only examine the odd numbered ones 1, 3, 5, ... It can ensure us always catch the fox. Observe the below table:

1	3	5	...	2m - 1
m	m + 1	m + 2	...	2m - 1

Suppose the fox hides in the hole number m , solving equation $m + k = 2k + 1$ gives that, after $k = m - 1$ days, we will examine the hole number $2m - 1$, while the fox moves exactly to it. Below *foldn* program demonstrates this process:

$$\begin{aligned}
 fox m &= foldn (1, m) h (m - 1) \\
 \text{where : } h (c, f) &= (c + 2, f + 1)
 \end{aligned}$$

If the fox hides in hole number p , and moves q holes everyday, we can denote such pair as (p, q) , then map them to the natural numbers with the method we introduced in chapter 6 about infinity. With this method, we can enumerate all (p, q) combinations and catch the fox.

10. What does the expression *foldr*(*nil*, *cons*) define?

It defines the list itself.

11. Read in a sequence of digits (string of digit numbers), convert it to decimal with *foldr*. How to handle hexadecimal digit and number? How to handle the decimal point?

If the lowest digit is on the left, and the highest digit on the right in the input list, we can convert it as below:

$$foldr (c d \mapsto 10d + c) 0$$

However, if the lowest digit is on the right, and the elements in the list are characters but not digit, then we need adjust it to:

$$1st \cdot foldr (c, (d, e) \mapsto ((toInt c)e + d, 10e)) (0, 1)$$

We can make it process the hexadecimal numbers by replacing 10 to 16 in this definition. When meet the decimal point, we can divide d , the result so far, by e to calculate the fractional part value.

$$1st \cdot foldr h (0, 1)$$

where

$$h (c, (d, e)) = \begin{cases} c = ' . ' & (d/e, 1) \\ \text{otherwise} & ((toFloat c)e + d, 10e) \end{cases}$$

12. Jon Bentley gives the maximum sum of sub-vector puzzle in *Programming Pearls*. For integer list $\{x_1, x_2, \dots, x_n\}$, find the range i, j , that maximizes the sum of $x_i + x_{i+1} + \dots + x_j$. Solve it with *foldr*.

If all numbers are positive, then the maximum sum is the sum of the whole list. This is because addition is monotone increasing upon positive numbers. If all numbers are negative, then the maximum sum should be zero, which is the sum of empty list. For any sub-list, the sum increases when add a positive number, while it decreases when add a negative number. We can maintain two things during folding: one is the maximum sum found so far S_m , the other is the sum of the sub-list till the current number being examined S . By adding the next element, if S exceeds S_m , it means we found a larger sub-list sum. Hence we replace S_m with S ; If S becomes negative, it means we complete the previous sub-list, and should start a new one.

$$\begin{aligned} max_s &= 1st \cdot foldr f (0, 0) \\ \text{where : } & f x (S_m, S) = (S'_m, S') \\ & \text{where in } f : S' = \max(0, x + S), S'_m = \max(S_m, S') \end{aligned}$$

Here is the example program implements this solution.

```
maxSum :: (Ord a, Num a) => [a] -> a
maxSum = fst . foldr f (0, 0) where
  f x (m, mSofar) = (m', mSofar') where
    mSofar' = max 0 (mSofar + x)
    m' = max mSofar' m
```

If want to return the sub-list together with the maximum sum, we can maintain two pairs P_m and P during folding, each pair contains the sum and the sub-list (S, L) .

$$\begin{aligned} max_s &= 1st \cdot foldr f ((0, []), (0, [])) \\ \text{where : } & f x (P_m, (S, L)) = (P'_m, P') \\ & \text{where in } f : P' = \max((0, []), (x + S, x : L)), P'_m = \max(P_m, P') \end{aligned}$$

13. The longest sub-string without repeated characters. Given a string, find the longest sub-string without any repeated characters in it. For example, the answer for string “abcabcb” is “abc”. Solve it with *foldr*.

We give two methods. One solution is to maintain the longest sub-string without repeated characters during folding, record and check if the character c has met before and its last appeared position. If c never occurred, or it appears before

the current sub-string we are examining, then we append it to the current sub-string, and compare with the longest one we've found so far. Otherwise, it means the current sub-string contains a repeated character, we need go back to its last occurred position, move ahead one, then restart searching.

$$\text{longest}(S) = \text{fst2} \cdot \text{foldr } f (0, |S|, |S|, \emptyset) \text{ zip}(\{1, 2, \dots\}, S)$$

Where folding starts from a tuple of 4 elements. It contains the length of the longest sub-string we found so far, the right boundary of the longest sub-string, the right boundary of the current sub-string, and a map recorded the last occurred position for different characters. Function fst2 extract the first two elements from the tuple as result. To obtain the position of each character easily during folding, we zip the string S and natural number sequence together. The most critical function f is defined as below:

$$f (i, c) (n_{\max}, e_{\max}, e, \text{Idx}) = (n'_{\max}, e'_{\max}, e', \text{Idx}[c] = i)$$

where:

$$\begin{aligned} n'_{\max} &= \max(n_{\max}, e' - i + 1) \\ e' &= \begin{cases} c \notin \text{Idx} : & e \\ \text{Idx}[c] = j : & \min(e, j - 1) \end{cases} \\ e'_{\max} &= \begin{cases} e' - i + 1 > n_{\max} : & e' \\ \text{otherwise} : & e_{\max} \end{cases} \end{aligned}$$

Here is an example program implement this solution. It returns the maximum length and the end position of the sub-string.

```
longest :: String -> (Int, Int)
longest xs = fst2 $ foldr f (0, n, n, Map.empty :: (Map Char Int))
  (zip [1..] xs) where
  fst2 (len, end, _, _) = (len, end)
  n = length xs
  f (i, x) ( maxlen, maxend, end, pos) =
    (maxlen', maxend', end', Map.insert x i pos) where
    maxlen' = max maxlen (end' - i + 1)
    end' = case Map.lookup x pos of
      Nothing -> end
      Just j -> min end (j - 1)
    maxend' = if end' - i + 1 > maxlen then end' else maxend
```

We record ending position because foldr starts from right. While the traditional way starts from the left, for example:

```
function LONGEST(S)
  Idx ← ∅
  nmax ← 0, smax ← 0, s ← 0
  for i ∈ {0, 1, ..., |S|} do
    if S[i] ∈ Idx then
      j ← Idx[S[i]]
      s = max(s, j + 1)
    if i - s + 1 > nmax then
      smax ← s
    nmax ← max(nmax, i - s + 1)
```

```

Idx[S[i]] = i
return S[smax...smax + nmax]

```

The second method is a number theory solution by leveraging prime numbers. We map each unique character c to a prime number p_c . For any string S , we can calculate the product of prime numbers mapped from its characters:

$$P = \prod_{c \in S} p_c$$

Therefore, for any new character c' , we can check whether the corresponding prime number p' divides P or not to know if c' appears in S . Based on this fact, we can design an algorithm. It maintains the product of primes during folding. When there is a character, that its corresponding prime number divides the product, we then find a repeated character. We need drop off the part containing the repeated character and go on folding. During this process, we also need update the longest sub-string.

$$\text{longest} = \text{fst} \cdot \text{foldr } f ((0, []), (0, []), 1)$$

Where the folding starts from a tuple of three elements. The first two are pairs, represent the longest sub-string (length and content), and the current sub-string. The last one in the tuple is the product of primes, starts from 1. Function f is defined as the following:

$$f \ c \ (m, (n, C), P) = \begin{cases} p_c | P : & \text{update}(m, (n + 1, c : C), p_c \times P) \\ \text{otherwise} : & \text{update}(m, (|C'|, C'), \prod_{x \in C'} p_x) \end{cases}$$

where:

$$\begin{aligned} \text{update}(a, b, P) &= (\max(a, b), b, P) \\ C' &= c : \text{takeWhile } (\neq c) C \end{aligned}$$

14. In the fold definition of Fibonacci numbers, the successor is computed as $(m', n') = (n, m + n)$. It is essentially matrix multiplication:

$$\begin{pmatrix} m' \\ n' \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} m \\ n \end{pmatrix}$$

Where it starts from $(0, 1)^T$. Then the Fibonacci numbers is isomorphic to natural numbers under the matrix multiplication:

$$\begin{pmatrix} F_n \\ F_{n+1} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Write a program to compute the power of 2-order square matrix, and use it to give the n -th Fibonacci number.

First we need define multiplication for square matrix of order 2, and the multiplication between square matrix and vector:

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \times \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{pmatrix}$$

and

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \times \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} = \begin{pmatrix} a_{11}b_1 + a_{12}b_2 \\ a_{21}b_1 + a_{22}b_2 \end{pmatrix}$$

When calculate the n -th power of M^n , we need not repeat multiplication n times. If $n = 4$, we can first calculate M^2 , then multiply the result to itself to obtain $(M^2)^2$. There are total two times of multiplication; If $n = 5$, we only need compute $M^4 \times M$, hence there are three times of multiplication. We can recursively compute the exponential fast based on n 's parity.

$$M^n = \text{pow}(M, n, I)$$

Where I is the identical square matrix of order 2: $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$, and the function pow is defined as below:

$$\text{pow}(M, n, A) = \begin{cases} n = 0 : & A \\ n \text{ is even} : & \text{pow}(M \times M, \frac{n}{2}, A) \\ \text{otherwise} : & \text{power}(M \times M, \lfloor \frac{n}{2} \rfloor, M \times A) \end{cases}$$

In fact, we can represent n in binary format, then perform folding on these 0, 1 bits to compute M^n fast.

.3 Recursion

1. The Euclidean algorithm described in this section is in recursive manner. Try to eliminate recursion, implement it and the extended Euclidean algorithm only with loop.

The classic Euclidean algorithm is tail recursive, it can be convert to loop easily:

```
function GCM( $a, b$ )
  while  $b \neq 0$  do
     $a, b \leftarrow b, a \bmod b$ 
  return  $a$ 
```

However, it's a bit hard to convert the extended Euclidean algorithm. Let us see three sequences r, s, t :

$$\begin{aligned} r_0 &= a, r_1 = b \\ s_0 &= 1, s_1 = 0 \\ t_0 &= 0, t_1 = 1 \\ \dots & \\ r_{i+1} &= r_{i-1} - q_i r_i, \text{ where } q_i = \lfloor r_i / r_{i-1} \rfloor \\ s_{i+1} &= s_{i-1} - q_i s_i \\ t_{i+1} &= t_{i-1} - q_i t_i \\ \dots & \end{aligned}$$

Obviously, when $r_{k+1} = 0$, the sequences terminate. We also know from Euclidean algorithm that at this time:

$$\text{gcm}(a, b) = \text{gcm}(r_{k-1}, r_k) = \text{gcm}(r_k, 0) = r_k$$

The more important fact is that, at this time, the following Bézout's identity holds:

$$\text{gcm}(a, b) = r_k = as_k + bt_k$$

Proof. We prove it with mathematical induction. First for 0 and 1:

$$\begin{aligned} 0 : \quad r_0 &= a & as_0 + bt_0 &= a \cdot 1 + b \cdot 0 = a \\ 1 : \quad r_1 &= b & as_1 + bt_1 &= a \cdot 0 + b \cdot 1 = b \end{aligned}$$

Next suppose $r_{i-1} = as_{i-1} + bt_{i-1}$ and $r_i = as_i + bt_i$ hold, for $i + 1$ case:

$$\begin{aligned} r_{i+1} &= r_{i-1} - q_i r_i && \text{sequence definition} \\ &= (as_{i-1} + bt_{i-1}) - q_i(as_i + bt_i) && \text{induction hypothesis} \\ &= a(s_{i-1} - q_i s_i) + b(t_{i-1} - q_i t_i) && \text{rearrange} \\ &= as_{i+1} + bt_{i+1} && \text{sequence definition} \end{aligned}$$

Hence the sequences satisfy Bézout's identity at any time. \square

With this fact, we can obtain the non-recursive extended Euclidean algorithm:

```
function EXT-GCM( $a, b$ )
   $s', s \leftarrow 0, 1$ 
   $t', t \leftarrow 1, 0$ 
  while  $b \neq 0$  do
     $q, r \leftarrow \lfloor a/b \rfloor, a \bmod b$ 
     $s', s \leftarrow s - qs', s'$ 
     $t', t \leftarrow t - qt', t'$ 
     $a, b \leftarrow b, r$ 
  return  $(a, s, t)$ 
```

2. Most programming environments require integers for modular operation. However, the length of segment isn't necessarily integer. Implement a modular operation that manipulates segments. What's about its efficiency?

Consider the compass and straightedge construction, we can use compass to intercept segment to obtain the modular result.

```
function MOD( $a, b$ )
  while  $b < a$  do
     $a \leftarrow a - b$ 
  return  $a$ 
```

It's efficiency is linear obviously. To optimize it, we introduce a lemma:

Lemma .3.1 (Recursive remainder lemma). *If $r = a \bmod 2b$, then:*

$$a \bmod b = \begin{cases} r \leq b : & r \\ r > b : & r - b \end{cases}$$

With this lemma, we can speed up the modular operation to logarithmic:

$$a \bmod b = \begin{cases} a & \text{if } a \leq b \\ a - b & \text{if } a - b \leq b \\ \text{otherwise} : & \begin{cases} a' \leq b : & a', \text{ where } a' = a \bmod (b + b) \\ a' > b : & a' - b \end{cases} \end{cases}$$

Inspired by Fibonacci numbers, Robot Floyd, and Donald Knuth managed to eliminate the recursion in this algorithm, hence obtained a purely iterative modular operation:

```

function MOD( $a, b$ )
  if  $a < b$  then
    return  $a$ 
   $c \leftarrow b$ 
  while  $c \leq a$  do
     $c, b \leftarrow (b + c, c)$  ▷ Increase  $c$  like Fibonacci numbers
  while  $b \neq c$  do
     $c, b \leftarrow (b, c - b)$  ▷ Decrease  $c$  back
    if  $c \leq a$  then
       $a \leftarrow a - c$ 
  return  $a$ 

```

3. In the proof of Euclidean algorithm, we mentioned “Remainders are always less than the divisor. We have $b > r_0 > r_1 > r_2 > \dots > 0$. As the remainder can not be less than zero, and the initial magnitude is finite, the algorithm must terminate.” Can r_n infinitely approximate zero, but not be zero? Does the algorithm always terminate? What does the precondition that a and b are commensurable ensure?

For the commensurable magnitudes, we can use the *Well-ordering principle* to show that the Euclidean algorithm must terminate. According to the well-ordering principle, every non-empty set of natural numbers has the minimum number. This property can be extended to set of integers, rationals, or even to the finite, non-empty sub-set of real numbers. From the definition of commensurable, we know the remainders form a finite set.

4. For the binary linear Diophantine equation $ax + by = c$, let x_1, y_1 and x_2, y_2 be two pairs of solution. Proof that the minimum of $|x_1 - x_2|$ is $b/gcm(a, b)$, and the minimum of $|y_1 - y_2|$ is $a/gcm(a, b)$

Let the greatest common divisor of a and b be $g = gcm(a, b)$. If x_0, y_0 is a pair of solution to the Diophantine equation $ax + by = c$, then the following x, y also form a pair of solution:

$$\begin{cases} x = x_0 - k \frac{b}{g} \\ y = y_0 + k \frac{a}{g} \end{cases}$$

It's easy to verify this fact:

$$\begin{aligned}
 ax + by &= a(x_0 - k \frac{b}{g}) + b(y_0 + k \frac{a}{g}) \\
 &= ax_0 + by_0 - ak \frac{b}{g} + bk \frac{a}{g} \\
 &= c - 0 = c
 \end{aligned}$$

We next prove that, every solution can be expressed in this form. Let x, y be an arbitrary pair of solution, we have $ax+by = c$ and $ax_0+by_0 = c$ both hold, therefore:

$$a(x - x_0) + b(y - y_0) = c - c = 0$$

Divide both sides with the greatest common divisor of a and b . It gives:

$$\frac{a}{g}(x - x_0) + \frac{b}{g}(y - y_0) = 0$$

$$\frac{b}{g}(y - y_0) = -\frac{a}{g}(x - x_0)$$

Note that the left side can be divided by $\frac{b}{g}$, hence it must divide the right side too.

But since $(\frac{a}{g}, \frac{b}{g}) = 1$, they are co-prime, hence $\frac{b}{g}$ must divide $(x - x_0)$. Let

$$x - x_0 = k \frac{b}{g}, \text{ for some } k \in \mathbf{Z}$$

Therefore

$$x = x_0 + k \frac{b}{g}$$

Substitute it back to above equation, we obtain:

$$y = y_0 - k \frac{a}{g}$$

Hence proved every pair of solution is in this form. Obviously, for any two such pairs, the minimized difference is obtained when $k = 1$. It means the minimum of $|x_1 - x_2|$ is $b/gcm(a, b)$, and the minimum of $|y_1 - y_2|$ is $a/gcm(a, b)$.

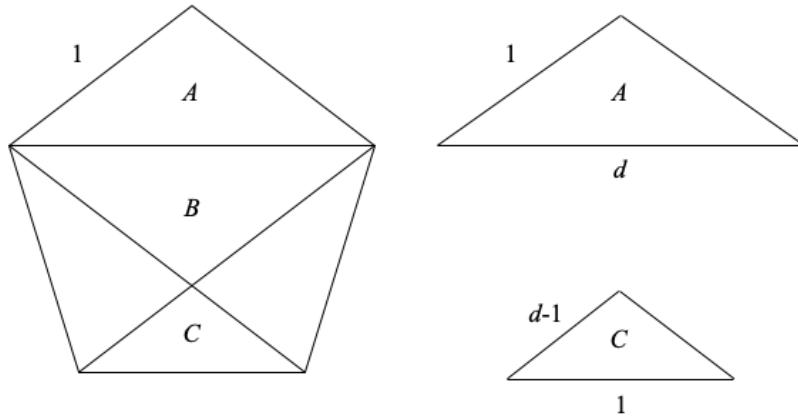
5. For the regular pentagon with side of 1, how long is the diagonal? Proof that in the pentagram shown in this chapter, the segment AC and AG are incommensurable. What's their ratio in real number?

Paul Lockhart gives a beautiful method in his *Measurement*[69]. As shown in figure 18, we can divide the regular pentagon into three triangles. It's easy to show that triangle A and B are congruent, and they are similar to triangle C (can you prove it?). If the length of the pentagon side is 1, let the diagonal length be d , then the base of triangle C is 1, and its two hypotenuses both are $d - 1$. From the similar triangles, we have:

$$1/d = (d - 1)/a$$

Solving this quadratic equation gives $d = \frac{\sqrt{5} + 1}{2}$. We drop the other solution $d = \frac{\sqrt{5} - 1}{2}$ as it is shorter than the side (in fact, it is the length of the hypotenuse of the smaller triangle C).

For the segment AC and AG in the figure, according to the 3 triangles we divided, they are actually the side and the diagonal of the pentagon. Suppose they are commensurable, then the base and hypotenuse of the smaller triangle are commensurable. Hence in the recursive inner pentagram star, the side and diagonal are also commensurable. We can repeat this process infinitely without end. Therefore, our assumption cannot hold. The side and diagonal are incommensurable. Written in decimals, it is about 0.6180339887498949...



The unit regular pentagon

6. Use λ conversion rules to verify $\text{tail} (\text{cons } p \ q) = q$.

The λ expressions for cons and tail are:

$$\begin{aligned} \text{cons} &= a \mapsto b \mapsto f \mapsto f \ a \ b \\ \text{tail} &= c \mapsto c \ (a \mapsto b \mapsto b) \end{aligned}$$

From the two definitions, we can verify $\text{tail} (\text{cons } p \ q) = q$ holds.

$$\begin{aligned} \text{tail} (\text{cons } p \ q) &= (c \mapsto c \ (a \mapsto b \mapsto b)) \ (\text{cons } p \ q) \\ &\xrightarrow{\beta} (\text{cons } p \ q) \ (a \mapsto b \mapsto b) \\ &= ((a \mapsto b \mapsto f \mapsto f \ a \ b) \ p \ q) \ (a \mapsto b \mapsto b) \\ &\xrightarrow{\beta} ((b \mapsto f \mapsto f \ p \ b) \ q) \ (a \mapsto b \mapsto b) \\ &\xrightarrow{\beta} (f \mapsto f \mapsto f \ p \ q) \ (a \mapsto b \mapsto b) \\ &\xrightarrow{\beta} (a \mapsto b \mapsto b) \ p \ q \\ &\xrightarrow{\beta} (b \mapsto b) \ q \\ &\xrightarrow{\beta} q \end{aligned}$$

7. We can define numbers with λ calculus. The following definition is called Church numbers:

0	$\lambda f. \lambda x. x$
1	$\lambda f. \lambda x. f x$
2	$\lambda f. \lambda x. f (f x)$
3	$\lambda f. \lambda x. f (f (f x))$
:	...

Define the addition and multiplication operators for the Church numbers with what we introduced in chapter 1.

For natural number n , the meaning as a Church number is to apply function f to x by n times. Let's define successor function first:

$$succ = \lambda n. \lambda f. \lambda x. f (n f x)$$

It means $f^{n+1}(x) = f(f^n(x))$. Then we can define addition as:

$$plus = \lambda m. \lambda n. \lambda f. \lambda x. m f (n f x)$$

Which means $f^{m+n}(x) = f^m(f^n(x))$. While the multiplication is defined as:

$$mul = \lambda m. \lambda n. \lambda f. \lambda x. m (n f) x$$

It means $f^{mn} = (f^n)^m(x)$.

8. The following defines the Church Boolean values, and the relative logic operators:

true	$\lambda x. \lambda y. x$
false	$\lambda x. \lambda y. y$
and	$\lambda p. \lambda q. p q p$
or	$\lambda p. \lambda q. p p q$
not	$\lambda p. p \text{ false true}$

where **false** is defined as same as the Church number 0. Use the λ conversion rules to prove that: **and true false = false**. Please give the definition of if ... then ... else ... expression with the λ calculus.

$$\begin{aligned} \text{and true false} &= (\lambda p. \lambda q. p q p) \text{ true false} \\ &\xrightarrow{\beta} \text{ true false true} \\ &= (\lambda x. \lambda y. x) \text{ false true} \\ &\xrightarrow{\beta} \text{ false} \end{aligned}$$

if ... then ... else ... expression can be defined as: $\lambda p. \lambda a. \lambda b. p a b$

9. Define the abstract *mapt* for binary trees without of using *foldt*.

$$\begin{cases} \text{mapt}(f, \text{nil}) &= \text{nil} \\ \text{mapt}(f, \text{node}(l, x, r)) &= \text{node}(\text{mapt}(f, l), f(x), \text{mapt}(f, r)) \end{cases}$$

10. Define a function *depth*, which counts for the maximum depth of a binary tree.

$$depth = \text{foldt}(\text{one}, x, y \mapsto 1 + \max(x, y), 0)$$

Where *one* is a constant function. It always returns 1. i.e. $\text{one} = x \mapsto 1$.

11. Someone thought the abstract fold operation for binary tree $foldt$, should be defined as the following:

$$\begin{aligned} foldt(f, g, c, nil) &= c \\ foldt(f, g, c, node(l, x, r)) &= foldt(f, g, g(foldt(f, g, c, l), f(x)), r) \end{aligned}$$

That is to say $g : (B \times B) \rightarrow B$ is a binary operation like add. Can we use this $foldt$ to define $mapt$?

This $foldt$ cannot define tree mapping. A tree should be mapped to another tree with the same structure. Each value in the tree is sent to another value. Note the type of f is $f : A \rightarrow B$, it send the element of type A in the tree to type B . While the type of g is $g : (B \times B) \rightarrow B$, it only maps values of B , but cannot preserve the tree structure.

12. The binary search tree (BST) is a special tree that the type A is comparable. For any none empty $node(l, k, r)$, all elements in the left sub-tree l are less than K , and all elements in the right sub-tree r are greater than k . Define function $insert(x, t) : (A \times Tree\ A) \rightarrow Tree\ A$ that inserts an element into the tree.

$$\begin{cases} insert(x, nil) = node(nil, x, nil) \\ insert(x, node(l, k, r)) = \begin{cases} x < k : & node(insert(x, l), k, r) \\ \text{otherwise} : & node(l, k, insert(x, r)) \end{cases} \end{cases}$$

13. Can we define the mapping operation for multi-trees with folding? If not, how should we modify the folding operation?

Similar to above exercise, we need modify the folding definition for multi-tree to preserve the tree structure:

$$\begin{cases} foldm(f, g, c, nil) &= c \\ foldm(f, g, c, node(x, ts)) &= g(f(x), map(foldm(f, g, c), ts)) \end{cases}$$

Where map applies to list. With this tree folding tool, we can define multi-tree map as below:

$$mapm(f) = foldm(f, node, nil)$$

.4 Symmetry

1. Do all the even numbers form a group under addition?

Yes, even numbers form a group under addition. Even number add even number, the result is still even. Add is associative. The unit is zero. The inverse of a number is its negate.

2. Can we find a subset of integers, that can form a group under multiplication?

The subset $\{-1, 1\}$ form a group under multiplication. The unit is 1. Every element is the reverse of itself.

3. Do all the positive real numbers form a group under multiplication?

Yes. Positive real numbers are close under multiplication. The unit is 1. For each number r , the reverse is $1/r$.

4. Do integers form a group under subtraction?

No, integers cannot form a group under subtraction. This is because subtraction is not associative. for e.g. $(3 - 2) - 1 = 0$, while $3 - (2 - 1) = 2$.

5. Find an example of group with only two elements.

As shown in previous exercise, set $\{-1, 1\}$ forms a group under addition. Another example is Boolean values $\{T, F\}$, they form a group under logic exclusive or (xor). The logic exclusive or is closed and associative. The unit is F , because every element exclusive or F gives itself. And every element is the reverse of itself.

6. What is the identity element for Rubik cube group? What is the inverse element for F ?

The unit for the Rubik cube group is the identity transform, which keeps any state unchanged. The reverse of F is F' , which rotate the face side 90 degree counter-clockwise.

7. The set of Boolean values $\{\text{True}, \text{False}\}$ forms a monoid under the logic or operator \vee . It is called ‘Any’ logic monoid. What is the identity element for this monoid?

False

8. The set of Boolean values $\{\text{True}, \text{False}\}$ forms a monoid under the logic and operator \wedge . It is called ‘All’ logic monoid. What is the identity element for this monoid?

True

9. For the comparable type, when compare two elements, there can be three different results. We abstract them as $\{<, =, >\}$ ⁷. For this set, we can define a binary operation to make it a monoid. What is the identity element for this monoid?

Define the binary operation as:

$$\begin{array}{rcl} < \circ x & = & < \\ = \circ x & = & x \\ > \circ x & = & > \end{array}$$

Where x is any element among the three. These three relations form a monoid. The unit is $=$.

10. Prove that the power operation for group, monoid, and semigroup is commutative: $x^m x^n = x^n x^m$

In order to prove the commutative law, we first use mathematical induction to prove a lemma: $x^n x = x x^n$. For group and monoid, when $n = 0$:

$$x^0 x = ex = x = xe = xx^0$$

As there is no unit in semigroup, we start from $n = 1$:

$$x^1 x = xx = xx^1$$

Suppose for n , $x^n x = xx^n$ holds, then for $n + 1$:

⁷Some programming languages, such as C, C++, Java use negative number, zero, and positive number to represent these three results. In Haskell, they are GT, EQ, and LE respectively.

$$\begin{aligned}
 x^{n+1}x &= (xx^n)x && \text{recursive definition of power} \\
 &= x(x^n x) && \text{associative} \\
 &= x(xx^n) && \text{induction hypothesis} \\
 &= xx^{n+1} && \text{recursive definition of power}
 \end{aligned}$$

With this lemma, we use mathematical induction again to prove the commutativity for power operation. For group and monoid, when $n = 0$, we have:

$$\begin{aligned}
 x^m x^0 &= x^m e && \text{definition of 0-th power} \\
 &= x^m && \text{definition of unit} \\
 &= e x^m && \text{definition of unit} \\
 &= x^0 x^m && \text{definition of 0-th power}
 \end{aligned}$$

Because semigroup does not have unit, we start from $n = 1$:

$$\begin{aligned}
 x^m x^1 &= x^m x && \text{definition of 1-st power} \\
 &= x x^m && \text{the lemma} \\
 &= x^1 x^m && \text{definition of 1-st power}
 \end{aligned}$$

Suppose the commutativity law $x^m x^n = x^n x^m$ holds for n , then for $n + 1$:

$$\begin{aligned}
 x^m x^{n+1} &= x^m (x x^n) && \text{recursive definition of power} \\
 &= (x^m x) x^n && \text{associative law for multiplication} \\
 &= x x^m x^n && \text{the lemma} \\
 &= x (x^m x^n) && \text{associative law for multiplication} \\
 &= x (x^n x^m) && \text{induction hypothesis} \\
 &= (x x^n) x^m && \text{associative law for multiplication} \\
 &= x^{n+1} x^m && \text{recursive definition of power}
 \end{aligned}$$

11. Is the odd-even test function homomorphic between the integer addition group $(\mathbb{Z} +)$ and the Boolean logic-and group $(Bool, \wedge)$? What about the group of integers without zero under multiplication?

odd-even test function maps every integer to Boolean value true or false. To test homomorphic, we need verify whether $f(a)f(b) = f(a \cdot b)$ holds. However, there is a negative case:

a, b are all odd, i.e. $odd(a) = odd(b) = True$. Their sum is even, $odd(a + b) = False$. However, the logic and result is $odd(a) \wedge odd(b) = True \neq odd(a + b) = False$.

Hence they are not homomorphic.

While the multiplicative group of integers without zero is homomorphic with the logic and group. We can verify all the three cases:

- a, b are all odd, i.e. $odd(a) = odd(b) = True$. Their product ab is still odd: $odd(ab) = True$. Hence, $odd(a) \wedge odd(b) = odd(ab)$;
- a, b are all even, i.e. $odd(a) = odd(b) = False$. Their product ab is still even: $odd(ab) = False$. Hence, $odd(a) \wedge odd(b) = odd(ab)$;
- a, b are odd and even respectively. Let $odd(a) = True, odd(b) = False$. Their product ab is even: $odd(ab) = False$. Hence, $odd(a) \wedge odd(b) = odd(ab)$.

Therefore, they are homomorphic.

12. Suppose two groups G and G' are homomorphic. In G , the element $a \rightarrow a'$. Is the order of a same as the order of a' ?

Their orders are same. To prove it, let the image of unit e be e' , the homomorphism is f . Denote the order of a is n , i.e. $a^n = e$.

On one hand:

$$f(a^n) = f(e) = e'$$

On the other hand, according to the definition of homomorphism:

$$\begin{aligned} f(a^n) &= f(a)f(a)\dots f(a) && \text{total } n \\ &= a'a'\dots a' && \text{total } n \\ &= (a')^n \end{aligned}$$

Summarize the above two results, we have $(a')^n = e'$. Hence the order of a and a' are same.

13. Prove that the identity element for transformation group must be identity transformation.

Suppose the unit transformation is $\epsilon' : a \rightarrow a^{\epsilon'} = \epsilon'(a)$. According to the definition of unit, for any transformation τ , $\epsilon'\tau = \tau$ holds, i.e.

$$\begin{aligned} \tau : a &\rightarrow a^\tau \\ \epsilon'\tau : a &\rightarrow (a^{\epsilon'})^\tau \end{aligned}$$

Therefore, $a^{\epsilon'} = a = \epsilon'(a)$, ϵ' is identity transformation.

14. List all the elements in S_4 .

(1);
 (12), (34), (13), (24), (14), (23);
 (123), (132), (134), (143), (124), (142), (234), (243);
 (1234), (1243), (1324), (1342), (1423), (1432);
 (12)(34), (13)(24), (14)(23).

There are total $4! = 24$ elements.

15. Express all the elements in S_3 as the product of cyclic forms.

permutation	123	213	132	321	231	312
cyclic notation	(1)	(12)	(23)	(13)	(123)	(132)

16. Write a program to convert the product of k -cycles back to permutation.

```

function PERMUTE( $C, n$ )
   $\pi \leftarrow [1, 2, \dots, n]$ 
  for  $c \in C$  do
     $j \leftarrow c[1]$ 
     $m \leftarrow |c|$ 
    for  $i \leftarrow 2$  to  $m$  do
       $\pi[j] \leftarrow c[i]$ 
       $j \leftarrow c[i]$ 
  
```

```

 $\pi[c[m]] \leftarrow c[1]$ 
return  $\pi$ 

```

17. What symmetries for what shape are defined by the symmetric group S_4 ?

It defines the symmetry of a tetrahedron. A tetrahedron has four vertex corners, each has an axis. Rotation around such axis by 120° and 240° is symmetric. Connect the middle point of every two opposite edges also gives an axis. There are total 3 such axes, each one is reflective symmetric. Include the identity transformation, there are total $2 \times 4 + 3 + 1 = 12$ symmetries. They form a group called A_4 , which is an alternating group.

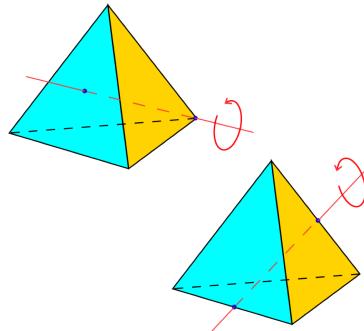


Figure 19: The rotation and reflection axes

For the tetrahedron 1234 in figure 20, elements in group A_4 transform it to:

3124, 2314, 1423, 1342, 4213, 3241, 4132, 2431, 2143, 3412, 4321, 1234

These 12 transformations are all proper congruences. Besides, there are also improper congruences. As shown in figure 20, when rotated by 120° against the axis, the tetrahedron transforms to 3124 on the right, this is a proper congruence. Then reflect it against surface $O13$, we get the bottom-right tetrahedron 3142. This is an improper congruence.

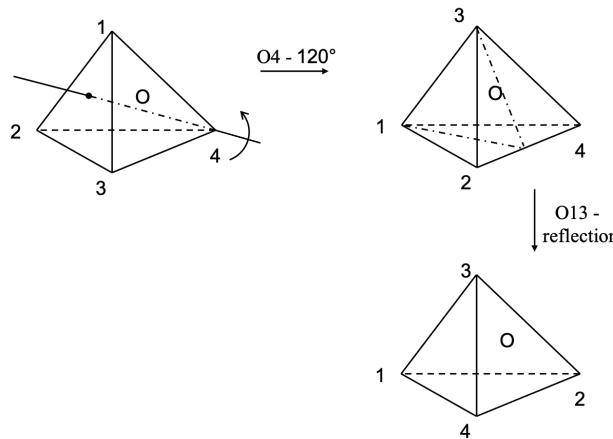


Figure 20: Improper congruence

For every element in A_4 , there is such an improper congruence by reflection. In total 12:

3142, 2341, 1432, 1324, 4231, 3214, 4123, 2413, 2134, 3421, 4312, 1243

Although there are other 2 reflection surfaces, they don't generate new transformations, as there are $4! = 24$ permutations for 4 points. All the 24 transformations of proper and improper congruences are corresponding to the elements in S_4 .

18. Proof that cyclic groups are abelian.

Let the generator be a . For any two elements, express them as the power of a , i.e. a^p, a^q respectively. Because:

$$a^p a^q = a^{p+q} = a^{q+p} = a^q a^p$$

Hence it is abelian.

19. Proof theorem that determines if a none empty subset H of group G forms a subgroup, if and only if:

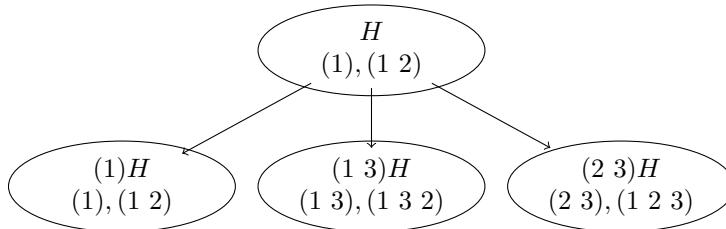
- i For all $a, b \in H$, the product $ab \in H$;
- ii For all element $a \in H$, its reverse $a^{-1} \in H$.

First prove the sufficiency. Condition i ensures H is closed on multiplication. The associativity holds for multiplication in G , hence also holds in H . Because H is not empty, there exists an element a , based on condition ii, the corresponding a^{-1} is also in H . And from condition i, $aa^{-1} = e \in H$ holds. Therefore H is a subgroup.

Next prove the necessity. If H forms a subgroup, then condition i is true obviously. For condition ii, since H is a group, there exists the unit element e' , such that for every element a in H , equation $e'a = a$ holds. As both e' and a are in G , we say e' is a solution to equation $ya = a$ in G . However, there is only one solution in G for this equation, which is the unit e of G , hence $e' = e \in H$.

As H is a group, equation $ya = e$ has solution a' in H . While a' is also the solution of this equation in G . However, the unique solution to this equation in G is a^{-1} . Therefore, $a' = a^{-1} \in H$.

20. List the left cosets for H in below figure.



Multiply the subgroup $H = \{(1), (1 2)\}$ from left with (1) , $(1 3)$, and $(2 3)$ gives:

$$\begin{aligned} (1)H &= \{(1), (1 2)\} \\ (1 3)H &= \{(1 3), (1 3 2)\} \\ (2 3)H &= \{(2 3), (1 2 3)\} \end{aligned}$$

21. Today is Sunday, what day it will be after 2^{100} days?

There are 7 days in a week. According to the Fermat's little theorem, $2^{7-1} \equiv 1 \pmod{7}$. We have:

$$\begin{aligned}
 2^{100} = 2^{16 \times 6 + 4} &\equiv 1 \times 2^4 \pmod{7} \\
 &\equiv 16 \pmod{7} \\
 &\equiv 2 \pmod{7}
 \end{aligned}$$

Therefore, it will be Tuesday.

22. Given two strings (character string or list), write a program to test if they form the same necklace.

Given two strings S_1, S_2 with the same length, we can duplicate S_1 and append it after itself, then examine whether S_2 is the sub-string of S_1S_1 . If yes, then they can form the same necklace.

$$equiv(S_1, S_2) = S_2 \subset (S_1 \# S_1)$$

23. Write a program to realize Eratosthenes sieve algorithm.

For all natural numbers from 2, pick the next as a prime, then remove all its multiplicands. Then do this repeatedly. Here is an example Haskell program:

```
primes = sieve [2..] where
  sieve (x:xs) = x : sieve [y | y <- xs, y `mod` x > 0]
```

The following are the example Python and Java programs.

```
def odds():
    i = 3
    while True:
        yield i
        i = i + 2

class prime_filter(object):
    def __init__(self, p):
        self.p = p
        self.curr = p

    def __call__(self, x):
        while x > self.curr:
            self.curr += self.p
        return self.curr != x

    def sieve():
        yield 2
        iter = odds()
        while True:
            p = next(iter)
            yield p
            iter = filter(prime_filter(p), iter)

list(islice(sieve(), 100))
```

```
public class Prime {
    private static LongPredicate sieves = x -> true; // initialize sieve as id
    public final static long[] PRIMES = LongStream
        .iterate(2, i -> i + 1)
        .filter(i -> sieves.test(i))
        .peek(i -> sieves = sieves.and(v -> v % i != 0)) // update, chain the
    sieve
        .limit(100) // take first 100
        .toArray();
}
```

24. Extend the idea of Eratosthenes sieve algorithm, write a program to generate Euler ϕ function values for all numbers from 2 to 100.

When generate prime numbers within n with the sieve of Eratosthenes, we can update the Euler ϕ function list by every prime numbers found so far. The list is initialized with all elements start from 1. For every prime number p , the corresponding $\phi(p) = p(1 - \frac{1}{p}) = p - 1$. All the multiplicands of p need multiply with

this value. However, it is not enough as $\phi(p^2) = p^2(1 - \frac{1}{p}) = p\phi(p)$. Next we need multiply all the Euler function value for multiplicands of p^2 by p , and repeat for multiplicands of p^3 and so on until p^m exceeds n . Below is the algorithm of this idea.

```

function EULER-TOTIENT( $n$ )
   $\phi \leftarrow \{1, 1, \dots, 1\}$                                  $\triangleright 1 \text{ to } n$ 
   $P \leftarrow \{2, 3, \dots, n\}$                              $\triangleright \text{sieve input}$ 
  while  $P \neq \emptyset$  do
     $p \leftarrow P[0]$ 
     $P \leftarrow \{x | x \in P[1\dots], x \bmod p \neq 0\}$ 
     $p' \leftarrow p$ 
    repeat
      for  $i \leftarrow$  from  $p'$  to  $n$  step  $p'$  do
        if  $p' = p$  then
           $\phi[i] \leftarrow \phi[i] \times (p - 1)$ 
        else
           $\phi[i] \leftarrow \phi[i] \times p$ 
         $p' \leftarrow p' \times p$ 
      until  $p' > n$ 
    return  $\phi$ 
  
```

25. Write a program to realize fast modular multiplication, and Fermat's primality test.

Our idea is to realize the fast modular multiplication with the similar approach when calculate power.

$$x^y = \begin{cases} y \text{ is even : } x^{\lfloor y/2 \rfloor} \\ y \text{ is odd : } x \cdot x^{\lfloor y/2 \rfloor} \end{cases}$$

Based on this, we can change it to modular multiplication as below:

```

function MOD-EXP( $x, y, n$ )
  if  $y = 0$  then
    return 1
   $z \leftarrow \text{MOD-EXP}(x, \lfloor y/2 \rfloor, n)$ 
  if  $y$  is even then
    return  $z^2 \bmod n$ 
  else
    return  $x \cdot z^2 \bmod n$ 
  
```

With Fermat's little theorem, we can realize the primality test with some selected 'witnesses':

```

function PRIMALITY( $n$ )
  random select  $k$  positive numbers  $a_1, a_2, \dots, a_k < n$ 
  if  $a_i^{n-1} \equiv 1 \pmod n$ , for all  $i = 1, 2, \dots, k$  then
  
```

```

return prime
else
    return composite

```

26. Prove the theorem that the two cancellation rules hold in a nonzero ring (ring without zero divisor).

Proof. Suppose there is no zero divisor in R . Because

$$ab = ac \Rightarrow a(b - c) = 0$$

as no zero divisor, hence:

$$a \neq 0, ab = ac \Rightarrow b - c = 0 \Rightarrow b = c$$

Similarly,

$$a \neq 0, ba = ca \Rightarrow b = c$$

Therefore, both cancellation rules hold in R . Reversely, suppose the first cancellation rule hold in R . Because:

$$ab = 0 \Rightarrow ab = a0$$

Based on the assumption,

$$a \neq 0, ab = 0 \Rightarrow b = 0$$

Hence R does not have zero divisor. We can make the similar prove when the second cancellation rule holds. \square

27. Prove that, all real numbers in the form of $a + b\sqrt{2}$, where a, b are integers form a integral domain under the normal addition and multiplication.

Proof. We need verify three things:

i The commutative law for multiplication holds.

$$\begin{aligned}
 (a + b\sqrt{2})(c + d\sqrt{2}) &= ac + 2bd + (ad + bc)\sqrt{2} \\
 &= (c + d\sqrt{2})(a + b\sqrt{2})
 \end{aligned}$$

ii There is unit 1 for multiplication.

$$1(a + b\sqrt{2}) = (a + b\sqrt{2})1 = a + b\sqrt{2}$$

iii No zero divisor

$$(a + b\sqrt{2})(c + d\sqrt{2}) = 0 \Rightarrow a = b = 0 \text{ or } c = d = 0$$

\square

28. Prove that $Q[a, b] = Q[a][b]$, where $Q[a, b]$ contains all the expressions combined with a and b , such as $2ab, a + a^2b$ etc.

Let us see an example first:

$$Q[\sqrt{2}, \sqrt{3}] = \{a + b\sqrt{2} + c\sqrt{3} + d\sqrt{6}, \text{ where } a, b, c, d \in Q\}$$

$$\begin{aligned} Q[\sqrt{2}][\sqrt{3}] &= \{a + b\sqrt{3}, \text{ where } a, b \in Q[\sqrt{2}]\} \\ &= \{a' + b'\sqrt{2} + (c + d\sqrt{2})\sqrt{3}, \text{ where } a', b', c, d \in Q\} \\ &= \{a' + (b' + d)\sqrt{2} + c\sqrt{3} + d\sqrt{6}, \text{ where } a', b', c, d \in Q\} \end{aligned}$$

Proof.

$$Q[a][b] = \{x_0 + x_1b + x_2b^2 + \dots + x_nb^n, \text{ where } x_i \in Q[a]\}$$

n is the minimum integer that polynomial $p(b) = 0$ exists. Substitute x_i with the expressions in field $Q[a]$.

$$\begin{aligned} Q[a][b] &= \{y_{0,0} + y_{0,1}a + y_{0,2}a^2 + \dots + y_{0,m}a^m + \\ &\quad (y_{1,0} + y_{1,1}a + y_{1,2}a^2 + \dots + y_{1,m}a^m)b + \\ &\quad \dots \\ &\quad +(y_{n,0} + y_{n,1}a + y_{n,2}a^2 + \dots + y_{n,m}a^m)b^n\} \end{aligned}$$

Where $y_{i,j} \in Q$, m is the minimum integer that polynomial $p(a) = 0$ exists.

Without loss of generality, we assume $m < n$ (otherwise, we let $m' = \min(m, n)$, $n' = \max(m, n)$). We can further convert it as:

$$\begin{aligned} Q[a][b] &= \{y_{0,0} + y_{0,1}a + y_{1,0}b + y_{0,2}a^2 + y_{1,1}ab + y_{2,0}b^2 + \dots \\ &\quad + y_{0,m}a^m + y_{1,m-1}a^{m-1}b + \dots + y_{m,0}b^m + \\ &\quad y_{1,m}a^m b + y_{2,m-1}a^{m-1}b^2 + \dots + y_{m,1}b^{m+1} + \dots \\ &\quad + y_{n,m}a^m b^n\} \end{aligned}$$

This field is formed with all the expressions of a, b . □

29. Prove that, for any polynomial $p(x)$ with rational coefficients, E/Q is the field extension, f is the Q -automorphism of E , then equation $f(p(x)) = p(f(x))$ holds.

Proof. Because f is automorphism, we have:

$$f(x + y) = f(x) + f(y), f(ax) = f(a)f(x), f(1/x) = 1/f(x)$$

Further, since f is Q -automorphism, we have:

$$f(x) = x, \forall x \in Q$$

Let $p(x) = a_0 + a_1x + \dots + a_nx^n$, where $a_i \in Q$, then:

$$\begin{aligned} f(p(x)) &= f(a_0 + a_1x + \dots + a_nx^n) \\ &= f(a_0) + f(a_1x) + \dots + f(a_nx^n) & f(x + y) = f(x) + f(y) \\ &= f(a_0) + f(a_1)f(x) + f(a_2)f(x)^2 + \dots + f(a_n)f(x)^n & f(ax) = f(a)f(x) \\ &= a_0 + a_1f(x) + a_2f(x)^2 + \dots + a_nf(x)^n & f(x) = x, \forall x \in Q \\ &= p(f(x)) \end{aligned}$$

□

30. Taking the complex number into account, what is the splitting field for polynomial $p(x) = x^4 - 1$? What are the functions in its Q -automorphism?

There are four roots for polynomial $x^4 - 1$. They are $\pm 1, \pm i$. We can factor the polynomial to $p(x) = (x + 1)(x - 1)(x + i)(x - i)$. Actually, the splitting field of $p(x)$ is not complex number field C . It is too big. The splitting field is $Q[i]$.

There are two transformations in this Q -automorphism. One is $f(a + bi) = a - bi$, the other is the identity transformation $g(x) = x$.

31. What's the Galois group for quadratic equation $x^2 - bx + c = 0$?

We know the two roots for quadratic equation are:

$$x_1, x_2 = \frac{b \pm \sqrt{b^2 - 4c}}{2}$$

There are three cases: (1) there exists a rational number, such that $b^2 - 4c = r^2$. There are two rational roots (including duplicated ones) in this case; (2) No such rational number. Equation is not solvable in rational field, but there are real roots; (3) the discriminant is negative, hence the equation is not solvable in real field. But there are complex roots. Let's see the corresponding Galois groups for these three cases:

Case 1: there are two rational roots $\frac{b \pm r}{2}$. There is only one element in its Galois group, which is the identity automorphism $f(x) = x$.

Case 2: there are two irrational roots $\frac{b \pm \sqrt{d}}{2}$. There are two elements in its Galois group. One is the automorphism $f(p + q\sqrt{d}) = p - q\sqrt{d}$, where p, q are rationals; the other is the identity transformation.

Case 3: There are two complex roots $\frac{b \pm i\sqrt{d}}{2}$. There are two elements in its Galois group. One is the automorphism $f(p + qi) = p - qi$, where p, q are real numbers; the other is the identity transformation.

Actually, in case 2 and 3, their Galois groups are isomorphic in the splitting field. Note that $f(f(x)) = x$. It is isomorphic to the group of two elements 0, 1 under the addition modulo 2. It is also isomorphic to the cyclic group C_2 or $\mathbf{Z}/2\mathbf{Z}$. Where the notation $\mathbf{Z}/2\mathbf{Z}$ means the quotient group of integers under addition \mathbf{Z} and its sub-group of even numbers $2\mathbf{Z}$.

32. Prove that, if p is prime number, then Galois group for equation $x^p - 1$ is the $(p - 1)$ -cycle cyclic group C_{p-1} .

The p roots of $x^p - 1$ are the points along the unit circle in complex plane, i.e. $1, \omega, \omega^2, \dots, \omega^{p-1}$. They can be expressed in form of $e^{2\pi k i/p}$. The splitting field is $Q[\omega]$.

Consider automorphism f as an element in Galois group $Gal(Q[\omega]/Q)$. According to the definition of automorphism, we have:

$$f(\omega)^k = f(\omega^k) = 1 \iff \omega^k = 1$$

It means $f(\omega)$ is also a p -th root of unity (a root of equation $x^p - 1 = 0$ ⁸). Denote:

⁸If p is not a prime number, but an integer n greater than 1, then the k -th power of the m -th root is $\zeta_m^k = e^{2\pi m k i/n}$. There may exist some $k < n$, such that $\zeta_m^k = 1$. Actually, it holds as far as n divides mk . However, if n is a prime number p , then k can't be less than p , but must be multiplicand of p .

$$f(\omega) = h_i(\omega) = \omega^i$$

If $f(\omega)$ is the i -th root of unity, we name it as h_i , where $1 \leq i \leq p - 1$ (why i can't be 0?). In this way, we establish a one-to-one mapping from Galois group to cyclic group C_{p-1} :

$$Gal(Q[\omega]/Q) \xrightarrow{\sigma} C_{p-1} : \sigma(h_i) = i$$

Where the Galois group contains $p - 1$ automorphisms $\{h_1, h_2, \dots, h_{p-1}\}$, hence it has the same order as the cyclic group C_{p-1} .

Next we prove this is a group isomorphism.

$$(h_i \cdot h_j)(\omega) = h_i(h_j(\omega)) = h_i(\omega^j) = \omega^{ji} = \omega^{ij}$$

hence

$$\sigma(h_i \cdot h_j) = ij = \sigma(h_i) \cdot \sigma(h_j)$$

And $h_1(\omega)$ is the generator of this cyclic Galois group.

Here are two different groups that often cause confusion. The first one is the group of integers under addition modulo n . It's a cyclic group, containing the residue class modulo n of $\{0, 1, 2, \dots, n - 1\}$, total n elements. This group is often denoted as $\mathbf{Z}/n\mathbf{Z}$. It is isomorphic to the group formed with the n roots of equation $x^n - 1 = 0$. The elements are the n -th root of unity $\{1 = \zeta_n^0, \zeta_n^1, \zeta_n^2, \dots, \zeta_n^{n-1}\}$. The binary operation is multiplication.

The second group is integers under multiplication modulo n . The group elements are not all the residues from 0 to $n - 1$, but all the ones coprime to n . The group operation is modulo multiplication. Denoted as $(\mathbf{Z}/n\mathbf{Z})^\times$. There are total $\phi(n)$ elements, where ϕ is Euler's totient function. When n is prime p , there are $\{1, 2, \dots, p - 1\}$, total $p - 1$ group elements. However, the multiplicative group modulo n is not necessary cyclic. Luckily it is cyclic when n is prime. An interesting fact is that $(\mathbf{Z}/p\mathbf{Z})^\times$ is isomorphic to the additive group $\mathbf{Z}/(p - 1)\mathbf{Z}$.

This exercise tells us: For the Galois group in rational field extension, if it is generated by n -th root of unity, then this group is isomorphic to the multiplicative group of integers modulo n , i.e. $(\mathbf{Z}/n\mathbf{Z})^\times$. For example, the cubic equation $x^3 - 1 = 0$ has three roots $\{1, \frac{-1 \pm i\sqrt{3}}{2}\}$. Its Galois group contains two automorphisms. One is $f(x) = x$, which is corresponding to $h_1(\omega) = \omega^1$; the other is $g(a + bi) = a - bi$, which is corresponding to $h_2(\omega) = \omega^2$. The effect of h_2 is transform the order of the three roots from 1, 2, 3 to 1, 3, 2.

$$\begin{aligned} 1 &\mapsto 1^2 = 1 \\ \omega &\mapsto \omega^2 \\ \omega^2 &\mapsto (\omega^2)^2 = \omega^3 \omega = \omega \end{aligned}$$

33. The 5th degree equation $x^5 - 1 = 0$ is radical solvable. What's its Galois subgroup chain?

From previous exercise, we know the 5 roots are points along the unit circle in the complex plane: $\{1, \zeta, \zeta^2, \zeta^3, \zeta^4\}$, where $\zeta = e^{2\pi i/5} = \frac{\sqrt{5} - 1 + i\sqrt{10 + 2\sqrt{5}}}{4}$. The Galois group in rational field is a cyclic group of order 4: $G(Q[\zeta]/Q) = C_4$. It is isomorphic to the multiplicative group modulo 5: $(\mathbf{Z}/5\mathbf{Z})^\times = \{1, 2, 3, 4\}_5$. There is no intermediate field extension. The splitting field is $Q[\zeta]$. The Galois group in splitting field is $\{1\}$.

Obviously, $\{1\}$ is the normal subgroup of C_4 . The quotient group $C_4/\{1\}$ is cyclic too. In previous exercise, we proved that cyclic group is abelian, hence the equation is radical solvable.

.5 categories

1. Prove that the identity arrow is unique (hint: refer to the uniqueness of identity element for groups in previous chapter).

Suppose there exists another identity arrow id'_A , pointed A from itself: $A \xrightarrow{id'_A} A$.

Consider every arrow from A to B : $A \xrightarrow{f} B$, from the definition of identity arrow, $f \circ id_A = f$ holds. When replace B with A , and replace f with id'_A , then:

$$id'_A \circ id_A = id'_A$$

Similar, for every arrow from B to A : $B \xrightarrow{g} A$, according to the definition of identity arrow, $id'_A \circ g = g$ holds. When replace B with A , replace g with id_A , then:

$$id'_A \circ id_A = id_A$$

Summarize these two result, we obtain $id_A = id'_A$, hence the identity arrow is unique.

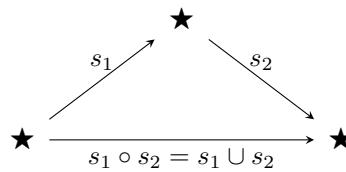
2. Verify the monoid (S, \cup, \emptyset) (the elements are sets, the binary operation is set union, the identity element is empty set) and $(N, +, 0)$ (elements are natural numbers, the binary operation is add, the identity element is zero) are all categories that contain only one object.

The key idea is that every monoid is a category contains only one object. It's a bit difficult to answer: what is the object in this category? In fact, it does not matter what this object is. The object is not necessary the monoid, or any given set. It even need not contain any elements. To avoid bother with concrete object, we give it notation \star .

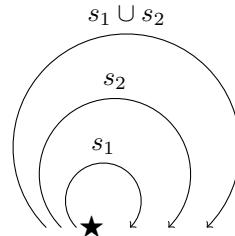
Let's first see the set monoid under union. Every set as an element in the monoid $s \in S$ can be used to define an arrow:

$$\star \xrightarrow{s} \star$$

Note there is no any inner structure (of the monoid) involved. The arrow composition is exactly set union.



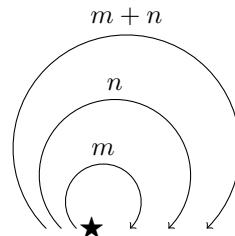
Because set union is associative, the arrows are also associative. The empty set is the unit of this monoid, it defines the identity arrow. As the empty set unions any set equals to that set itself. It serves as the identity arrow. The set monoid under union does form a category containing only one object.



Next let us see the additive monoid for natural numbers. Every number n can define an arrow:

$$\star \xrightarrow{n} \star$$

The arrow composition is addition. Since addition is associative, hence arrow composition is also associative. Zero, 0 defines the identity arrow. This is because 0 adds to any number equals to this number itself. Therefore, the additive monoid for natural numbers does form a category.



3. In chapter 1, we introduced Peano's axioms for natural numbers and isomorphic structures to Peano arithmetic, like the linked-list etc. They can be described in categories. This was found by German mathematician Richard Dedekind although the category theory was not established by his time. We named this category as Peano category, denoted as **Pno**. The objects in this category is (A, f, z) , where A is a set, for example natural numbers N ; $f : A \rightarrow A$ is a successor function. It is *succ* for natural numbers; $z \in A$ is the starting element, it is zero for natural numbers. Given any two Peano objects (A, f, z) and (B, g, c) , define the morphism from A to B as:

$$A \xrightarrow{\phi} B$$

It satisfies:

$$\phi \circ f = g \circ \phi \quad \text{and} \quad \phi(z) = c$$

Verify that **Pno** is a category.

An object in Peano category is a tuple of (A, f, z) . An arrow is a map ϕ that preserve the tuple structure. Arrow composition is function composition:

$$\begin{array}{c} A \xrightarrow{\phi} B \xrightarrow{\psi} C \\ A \xrightarrow{\psi \circ \phi} C \end{array}$$

Because function composition is associative, hence arrow composition is associative.

For identity arrow:

$$A \xrightarrow{id_A} A$$

It satisfies $id_A(z) = z$, and $id_A \circ f = f \circ id_A$.

Obviously, the tuple $(N, succ, 0)$ is an object in Peano category. It's interesting that, for every object (A, f, z) in Peano category, there is a unique arrow:

$$(N, succ, 0) \xrightarrow{\sigma} (A, f, z)$$

where:

$$\sigma(n) = f^n(z)$$

which maps any natural number n to the result of applying f to z for n times.

4. For the list functor, define the arrow map with *foldr*.

It's about to define list map with *foldr* essentially:

$$fmap f = foldr f Nil$$

5. Verify that the composition of maybe functor and list functor **Maybe** \circ **List** and **List** \circ **Maybe** are all functors.

We only prove **Maybe** \circ **List** is a functor. The other proof is similar. Any object A is sent to **Maybe(List A)**. For arrows, let us first see the case of identity arrow:

$$\begin{aligned} (\text{Maybe} \circ \text{List}) id &= \text{Maybe(List id)} && \text{functor composition} \\ &= \text{Maybe id} && \text{identity arrow for list functor} \\ &= id && \text{identity arrow for maybe functor} \end{aligned}$$

Next is about arrow composition:

$$\begin{aligned} (\text{Maybe} \circ \text{List}) (f \circ g) &= \text{Maybe(List (f \circ g))} && \text{functor composition} \\ &= \text{Maybe}((\text{List } f) \circ (\text{List } g)) && \text{composition for list functor} \\ &= (\text{Maybe (List } f)) \circ (\text{Maybe (List } g)) && \text{composition for maybe functor} \\ &= ((\text{Maybe} \circ \text{List}) f) \circ ((\text{Maybe} \circ \text{List}) g) && \text{functor composition} \end{aligned}$$

6. Proof that the composition for any functors $\mathbf{G} \circ \mathbf{F}$ is still a functor.

Similar to previous exercise, we need prove functor composition satisfies identity arrow and arrow composition. First for identity arrow:

$$\begin{aligned} (\mathbf{G} \circ \mathbf{F}) \text{ id} &= \mathbf{G}(\mathbf{F} \text{ id}) && \text{functor composition} \\ &= \mathbf{G} \text{ id} && \text{identity arrow for functor } \mathbf{F} \\ &= \text{id} && \text{identity arrow for functor } \mathbf{G} \end{aligned}$$

Then for arrow composition:

$$\begin{aligned} (\mathbf{G} \circ \mathbf{F}) (\phi \circ \psi) &= \mathbf{G}(\mathbf{F} (\phi \circ \psi)) && \text{functor composition} \\ &= \mathbf{G}((\mathbf{F} \phi) \circ (\mathbf{F} \psi)) && \text{arrow composition for functor } \mathbf{F} \\ &= (\mathbf{G} (\mathbf{F} \phi)) \circ (\mathbf{G} (\mathbf{F} \psi)) && \text{arrow composition for functor } \mathbf{G} \\ &= ((\mathbf{G} \circ \mathbf{F}) \phi) \circ ((\mathbf{G} \circ \mathbf{F}) \psi) && \text{functor composition} \end{aligned}$$

7. Give an example functor for preset.

The functor for preset category is a monotone function.

8. For the binary tree defined in chapter 2, define the functor for it.

Consider an object A in set total function category, the binary tree functor sends it to:

data Tree A = Empty | Branch (Tree A) A (Tree A)

For arrow $A \xrightarrow{f} B$, the binary tree functor maps it to:

$$\begin{aligned} \text{fmap } f \text{ Empty} &= \text{Empty} \\ \text{fmap } f (\text{Branch } l x r) &= \text{Branch} (\text{fmap } f l) (f x) (\text{fmap } f r) \end{aligned}$$

Or we can use the *mapt* defined in chapter 2:

$$\text{fmap} = \text{mapt}$$

9. For any two objects in a poset, what is their product? what is their coproduct?

In chapter 3, we mentioned any poset is a category, every element is the poset is an object, there is at most one arrow between two objects (the arrow exists if they have ordering relation). For two elements (objects) a and b , if they have arrows both to the up and down stream, then:

$$\text{meet } a \wedge b \quad \text{join } a \vee b$$

is the

$$\text{product} \quad \text{coproduct}$$

for this pair of objects.

Where meet is the least upper bound (also called supremum) of the two objects, and join is the greatest lower bound (infimum) of them. In general, the join and meet of a subset of a partially ordered set need not exist, hence the product and coproduct of a poset need not exist too.

10. Prove the absorption law for coproduct, and verify the coproduct functor satisfies composition condition.

The absorption law for coproduct states:

$$[p, q] \circ (f + g) = [p \circ f, q \circ g]$$

Proof.

$$\begin{aligned}
 & [p, q] \circ (f + g) \\
 &= [p, q] \circ [\text{left} \circ f, \text{right} \circ g] && \text{definition of } + \\
 &= [[p, q] \circ (\text{left} \circ f), [p, q] \circ (\text{right} \circ g)] && \text{fusion law} \\
 &= [[p, q] \circ \text{left} \circ f, [p, q] \circ \text{right} \circ g] && \text{associative} \\
 &= [p \circ f, q \circ g] && \text{cancellation law}
 \end{aligned}$$

□

The composition condition for coproduct states:

$$(f + g) \circ (f' + g') = f \circ f' + g \circ g'$$

Proof. Let $p = \text{left} \circ f$, and $q = \text{right} \circ g$

$$\begin{aligned}
 & (f + g) \circ (f' + g') \\
 &= [\text{left} \circ f, \text{right} \circ g] \circ (f' + g') && \text{definition of } + \\
 &= [p, q] \circ (f' + g') && \text{substitute with } p, q \\
 &= [p \circ f', q \circ g'] && \text{absorption law} \\
 &= [\text{left} \circ f \circ f', \text{right} \circ g \circ g'] && \text{substitute } p, q \text{ back} \\
 &= [\text{left} \circ (f \circ f'), \text{right} \circ (g \circ g')] && \text{associative law} \\
 &= f \circ f' + g \circ g' && \text{reverse of } +
 \end{aligned}$$

□

11. Prove that swap satisfies the natural transformation condition $(g \times f) \circ \text{swap} = \text{swap} \circ (f \times g)$

For $A \xrightarrow{f} C$ and $B \xrightarrow{g} D$, we need prove the below diagram commutes.

$$\begin{array}{ccc}
 (A, B) & \xrightarrow{\text{swap}_{A,B}} & (B, A) \\
 f \times g \downarrow & & \downarrow g \times f = \text{swap } f \times g \\
 (C, D) & \xrightarrow{\text{swap}_{C,D}} & (D, C)
 \end{array}$$

Proof.

$$\begin{aligned}
 & ((g \times f) \circ \text{swap}) (A, B) \\
 &= (g \times f) (\text{swap} (A, B)) && \text{definition of composition} \\
 &= (g \times f) \circ (B, A) && \text{definition of } \text{swap} \\
 &= (g B, f A) && \text{product of arrows} \\
 &= (D, C) && \text{definition of } g, f \\
 &= \text{swap} (C, D) && \text{reverse of } \text{swap} \text{ definition} \\
 &= \text{swap} (f A, g B) && \text{reverse of } f, g \text{ definition} \\
 &= \text{swap} ((f \times g) (A, B)) && \text{product of arrow} \\
 &= (\text{swap} \circ (f \times g)) (A, B) && \text{reverse of composition}
 \end{aligned}$$

□

12. Prove that the polymorphic function *length* is a natural transformation. It is defined as the following:

$$\begin{aligned} \text{length} &: [A] \rightarrow \mathbf{Int} \\ \text{length} [] &= 0 \\ \text{length} (x : xs) &= 1 + \text{length} xs \end{aligned}$$

For any object A , the arrow *length* indexed by A is:

$$[A] \xrightarrow{\text{length}_A} \mathbf{K}_{\mathbf{Int}} A$$

where $\mathbf{K}_{\mathbf{Int}}$ is constant functor. It sends every object to \mathbf{Int} , and sends every arrow to identity arrow $id_{\mathbf{Int}}$. For arrow $A \xrightarrow{f} B$, we need prove below diagram commutes.

$$\begin{array}{ccc} A & & [A] \xrightarrow{\text{length}_A} \mathbf{K}_{\mathbf{Int}} A \\ f \downarrow & \text{List}(f) \downarrow & \downarrow \mathbf{K}_{\mathbf{Int}}(f) \\ B & & [B] \xrightarrow{\text{length}_B} \mathbf{K}_{\mathbf{Int}} B \end{array}$$

From the definition of constant functor, this diagram is equivalent to:

$$\begin{array}{ccc} A & & [A] \xrightarrow{\text{length}_A} \mathbf{Int} \\ f \downarrow & \text{List}(f) \downarrow & \downarrow id \\ B & & [B] \xrightarrow{\text{length}_B} \mathbf{Int} \end{array}$$

We are about to prove:

$$id \circ \text{length}_A = \text{length}_B \circ \text{List}(f)$$

Which means: $\text{length}_A = \text{length}_B \circ \text{List}(f)$

Proof. Use mathematical induction, we first consider the empty list case:

$$\begin{aligned} & \text{length}_B \circ \text{List}(f) [] \\ = & \text{length}_B [] & \text{definition of list functor} \\ = & 0 & \text{definition of length} \\ = & \text{length}_A [] & \text{reverse of length definition} \end{aligned}$$

Next suppose $\text{length}_B \circ \text{List}(f) as = \text{length}_A as$ holds, we have:

$$\begin{aligned} & \text{length}_B \circ \text{List}(f)(a : as) \\ = & \text{length}_B (f(a) : \text{List}(f) as) & \text{definition of list functor} \\ = & 1 + \text{length}_B (\text{List}(f) as) & \text{definition of length} \\ = & 1 + \text{length}_B \circ \text{List}(f) as & \text{arrow composition} \\ = & 1 + \text{length}_A as & \text{induction assumption} \\ = & \text{length}_A (a : as) & \text{reverse of length definition} \end{aligned}$$

□

13. Natural transformation is composable. Consider two natural transformations $\mathbf{F} \xrightarrow{\phi} \mathbf{G}$ and $\mathbf{G} \xrightarrow{\psi} \mathbf{H}$. For any arrow $A \xrightarrow{f} B$, draw the diagram for their composition, and list the commutative condition.

$$\begin{array}{ccccc}
 \mathbf{F}A & \xrightarrow{\phi_A} & \mathbf{G}A & \xrightarrow{\psi_A} & \mathbf{H}A \\
 \mathbf{F}(f) \downarrow & & \mathbf{G}(f) \downarrow & & \mathbf{H}(f) \downarrow \\
 \mathbf{F}B & \xrightarrow{\phi_B} & \mathbf{G}B & \xrightarrow{\psi_B} & \mathbf{H}B
 \end{array}$$

The commutative condition is:

$$\mathbf{H}(f) \circ (\psi_A \circ \phi_A) = (\psi_B \circ \phi_B) \circ \mathbf{F}(f)$$

14. In the poset example, we say if there exists the minimum (or the maximum) element, then the minimum (or the maximum) is the initial object (or the final object). Consider the category of all posets **Poset**, if there exists the initial object, what is it? If there exists the final object, what is it?

For the **Poset** category, the objects are posets, the arrows are monotone functions. For two posets P, Q , arrow $P \xrightarrow{h} Q$ means for any two ordered elements $a \leq b$ in P , $h(a) \leq h(b)$ holds.

The initial object in this category is the empty poset $0 = \emptyset$. There is unique arrow from it to any poset P :

$$\emptyset \longrightarrow P$$

The final object is the singleton poset $1 = \{\star\}$, the order relationship is $R = \{\star, \star\}$, i.e. $\star \leq \star$. From any poset P , there is unique arrow to 1 :

$$\begin{array}{ccc}
 P & \longrightarrow & \{\star\} \\
 p & \mapsto & \star
 \end{array}$$

15. In the Peano category **Pno** (see exercise 2 in section 1), what is the initial object in form (A, f, z) ? What is the final object?

The initial object is $(\mathbf{N}, \text{succ}, 0)$. There is unique arrow from it to any object:

$$(\mathbf{N}, \text{succ}, 0) \xrightarrow{\sigma} (A, f, z) : \sigma(n) = f^n(z)$$

The final object is a singleton $1 = (\{\star\}, \star, \text{id})$. There is unique arrow from any object (A, f, z) to the final object:

$$(A, f, z) \xrightarrow{\sigma} 1 : \sigma(a) = \star$$

16. Verify that **Exp** is a category. What is the *id* arrow and arrow composition in it?

Let us first verify the *id* arrow $h \xrightarrow{\text{id}} h$, such that the following diagram commutes:

$$\begin{array}{ccc}
 A & & A \times B \\
 id_A \downarrow & & id_A \times id_B \downarrow \\
 A & & A \times B \xrightarrow{h} C
 \end{array}$$

Next is the arrow composition:

The composition of $h \xrightarrow{i} k$ and $k \xrightarrow{j} m$ is $j \circ i$ such that below diagram commutes:

$$\begin{array}{ccc}
 A & & A \times B \\
 f \downarrow & & f \times id_B \downarrow \\
 D & & D \times B \\
 g \downarrow & & g \times id_B \downarrow \\
 E & & E \times B \\
 & & \xrightarrow{k} C \\
 & & \xrightarrow{m} C
 \end{array}$$

For arrow $h \xrightarrow{j} k$, it means $id_k \circ j = j = j \circ id_h$ holds. And the associative law hold for any three arrows.

17. In the reflection law $curry\ apply = id$, what is the subscript of the id arrow? Please prove it with another method.

The subscript of the id arrow is the type of the binary arrow: $A \times B \rightarrow C$.

Proof.

$$\begin{aligned}
 & curry \circ apply\ f\ a\ b \\
 = & curry\ (apply\ f)\ a\ b & \text{definition of composition} \\
 = & (apply\ f)\ (a, b) & \text{definition of } curry \\
 = & f(a, b) & \text{definition of } apply \\
 = & id_{A \times B \rightarrow C}\ f(a, b)
 \end{aligned}$$

□

18. We define the equation

$$(curry\ f) \circ g = curry(f \circ (g \times id))$$

as the fusion law for Currying. Draw the diagram and prove it.

$$\begin{array}{ccc}
 D & & D \times B \\
 g \downarrow & & g \times id \downarrow \\
 A & & A \times B \\
 \text{curry } f \downarrow & & f \circ (g \times id) \downarrow \\
 C^B & & C^B \times B \\
 & & \xrightarrow{apply} C
 \end{array}$$

Observe the triangle of $D \times B$, $A \times B$, and C . We know the arrow of $D \times B \rightarrow C$ is $f \circ (g \times id)$.

According to the definition of exponentials and transpose arrow, we have:

$$apply \circ (curry\ f) \circ g = f \circ (g \times id)$$

According to the universal property of *curry* and *apply*:

$$(curry\ f) \circ g = curry(f \circ (g \times id))$$

19. Draw the diagram to illustrate the reverse element axiom for group.

The reverse element axiom can be formalized as: $m \circ (id, i) = m \circ (i, id) = e$

$$\begin{array}{ccccc} G & \xrightarrow{(id, i)} & G \times G & \xleftarrow{(i, id)} & G \\ \downarrow & & m \downarrow & & \downarrow \\ 1 & \xrightarrow{e} & G & \xleftarrow{e} & 1 \end{array}$$

20. Let p be a prime. Use the F-algebra to define the α arrow for the multiplicative group for integers modulo p (refer to the previous chapter for the definition of this group).

According to the α arrow defined for group:

$$\mathbf{F}A \xrightarrow{\alpha = e + m + i} A$$

The multiplicative group for integers modulo p is defined as:

$$\begin{array}{lll} e() = 1 & & 1 \text{ is the unit} \\ m(a, b) = ab \bmod p & & \text{multiplication modulo } p \\ i(a) = a^{p-2} \bmod p & & \text{Fermat's little theorem } a^{p-1} \equiv 1 \pmod p \end{array}$$

21. Define F-algebra for ring (refer to the previous chapter for definition of ring).

The algebraic structure of ring contains three parts:

- Carrier object R , the set that carries the algebraic structure of ring;
- Polynomial functor $\mathbf{F}A = 1 + 1 + A \times A + A \times A + A$;
- Arrow $\mathbf{F}A \xrightarrow{\alpha = z + e + p + m + n} A$, consists of the unit of addition z , the unit of multiplication e , addition p , multiplication m , and negation n .

These define the *F*-algebra (R, α) for ring. When the carrier object is integers for example, the ring is defined as below under standard arithmetic:

$$\begin{aligned} z() &= 0 \\ e() &= 1 \\ p(a, b) &= a + b \\ m(a, b) &= ab \\ n(a) &= -a \end{aligned}$$

22. What is the id arrow for F -algebra category? What is the arrow composition?

The id arrow is the homomorphism from F -algebra (A, α) to itself. The arrow composition is the composition of F -morphisms. The arrow between carrier object $A \xrightarrow{f} B \xrightarrow{g} C$ makes the following diagram commute:

$$\begin{array}{ccccc}
 & \mathbf{F}(f) & & \mathbf{F}(g) & \\
 \mathbf{F}A & \xrightarrow{\quad} & \mathbf{F}B & \xrightarrow{\quad} & \mathbf{F}C \\
 \alpha \downarrow & & \beta \downarrow & & \gamma \downarrow \\
 A & \xrightarrow{f} & B & \xrightarrow{g} & C
 \end{array}$$

$$g \circ f \circ \alpha = \gamma \circ \mathbf{F}(g) \circ \mathbf{F}(f) = \gamma \circ \mathbf{F}(g \circ f)$$

23. Someone write the natural number like functor as the below recursive form. What do you think about it?

```
data NatF A = ZeroF | SuccF (NatF A)
```

No, such definition does not work. Consider carrier object A , Functor **NatF** is recursive, it does not send A to a determined object. In fact, we expect it is mapped to a object in Peano category (A, f, z) .

24. We can define an α arrow for $\mathbf{NatF}Int \rightarrow Int$, named *eval*:

$$\begin{aligned}
 eval &: \mathbf{NatF}Int \rightarrow Int \\
 eval \ ZeroF &= 0 \\
 eval \ (SuccF \ n) &= n + 1
 \end{aligned}$$

We can recursively substitute $A' = \mathbf{NatF}A$ to functor **NatF** by n times. We denote the functor obtained as $\mathbf{NatF}^n A$. Can we define the following α arrow?

$$eval : \mathbf{NatF}^n Int \rightarrow Int$$

$$\begin{aligned}
 eval &: \mathbf{NatF}^n Int \rightarrow Int \\
 eval \ ZeroF &= 0 && ZeroF \text{ is an object of } \mathbf{NatF}^n Int \\
 eval \ (SuccF \ ZeroF) &= 1 && ZeroF \text{ is an object of } \mathbf{NatF}^{n-1} Int \\
 eval \ (SuccF \ (SuccF \ ZeroF)) &= 2 && ZeroF \text{ is an object of } \mathbf{NatF}^{n-2} Int \\
 &\vdots && \\
 eval \ (SuccF^{n-1} \ ZeroF) &= n - 1 && ZeroF \text{ is an object of } \mathbf{NatF} Int \\
 eval \ (SuccF^n \ m) &= m + n &&
 \end{aligned}$$

25. For the binary tree functor **TreeF** $A \ B$, fix A , use the fixed point to prove that $(\mathbf{Tree} \ A, [nil, branch])$ is the initial algebra

Let $B' = \mathbf{TreeF} \ A \ B$. We recursively apply to itself, and call this result as **Fix** (**TreeF** A).

$$\begin{aligned}
 \mathbf{Fix} \ (\mathbf{TreeF} \ A) &= \mathbf{TreeF} \ A \ (\mathbf{Fix} \ (\mathbf{TreeF} \ A)) && \text{definition of fixed point} \\
 &= \mathbf{TreeF} \ A \ (\mathbf{TreeF} \ A \ (\dots)) && \text{expand} \\
 &= \mathbf{NilF} \mid \mathbf{BrF} \ A \ (\mathbf{TreeF} \ A \ (\dots)) \ (\mathbf{TreeF} \ A \ (\dots)) && \text{definition of binary tree functor} \\
 &= \mathbf{NilF} \mid \mathbf{BrF} \ A \ (\mathbf{Fix} \ (\mathbf{TreeF} \ A)) \ (\mathbf{Fix} \ (\mathbf{TreeF} \ A)) && \text{reverse of fixed point}
 \end{aligned}$$

Compare with the definition of *TreeA*:

```
data Tree A = Nil | Br A (Tree A) (Tree A)
```

Hence $\mathbf{Tree} \ A = \mathbf{Fix} \ (\mathbf{TreeF} \ A)$. The initial algebra is $(\mathbf{Tree} \ A, [nil, branch])$.

.6 Fusion

1. Verify that folding from left can also be defined with *foldr*:

$$\text{foldl } f z xs = \text{foldr } (b \ g \ a \mapsto g (f a b)) \text{ id } xs \ z$$

To make it easy, we rewrite it to:

$$\begin{aligned} \text{foldl } f z xs &= \text{foldr } \text{step id } xs \ z \\ \text{where: } \text{step } x g a &= g (f a x) \end{aligned}$$

$$\begin{aligned} &\text{foldl } f z [x_1, x_2, \dots, x_n] \\ &= (\text{foldr } \text{step id } [x_1, x_2, \dots, x_n]) z \\ &= (\text{step } x_1 (\text{step } x_2 (\dots (\text{step } x_n \text{ id}))) \dots) z \\ &= (\text{step } x_1 (\text{step } x_2 (\dots (a_n \mapsto \text{id } (f a_n x_n))) \dots) z \\ &= (\text{step } x_1 (\text{step } x_2 (\dots (a_{n-1} \mapsto (a_n \mapsto \text{id } (f a_n x_n)) (f a_{n-1} x_{n-1}))) \dots) z \\ &= (a_1 \mapsto (a_2 \mapsto (\dots (a_n \mapsto \text{id } (f a_n x_n)) (f a_{n-1} x_{n-1})) \dots (f a_2 x_2)) (f a_1 x_1)) z \\ &= (a_1 \mapsto (a_2 \mapsto (\dots (a_n \mapsto f a_n x_n) (f a_{n-1} x_{n-1}))) \dots) (f a_1 x_1)) z \\ &= (a_1 \mapsto (a_2 \mapsto (\dots (a_{n-1} \mapsto f (f a_{n-1} x_{n-1}) x_n) \dots)) (f a_1 x_1)) z \\ &= (a_1 \mapsto f (f (\dots (f a_1 x_1) x_2) \dots) x_n) z \\ &= f (f (\dots (f z x_1) x_2) \dots) x_n \end{aligned}$$

We can further write *f* as an infix of \oplus to highlight the difference between *foldl* and *foldr*:

$$\text{foldl } \oplus f z = ((\dots(z \oplus x_1) \oplus x_2) \dots) \oplus x_n$$

2. Prove the below build...foldr forms hold:

$$\begin{aligned} \text{concat } xss &= \text{build } (f z \mapsto \text{foldr } (xs x \mapsto \text{foldr } f x xs) z xss) \\ \text{map } f xs &= \text{build } (\oplus z \mapsto \text{foldr } (y ys \mapsto (f y) \oplus ys) z xs) \\ \text{filter } f xs &= \text{build } (\oplus z \mapsto \text{foldr } (x xs' \mapsto \begin{cases} f(x) : & x \oplus xs' \\ \text{otherwise} : & xs' \end{cases}) z xs) \\ \text{repeat } x &= \text{build } (\oplus z \mapsto \text{let } r = x \oplus r \text{ in } r) \end{aligned}$$

First for the list *concat*:

Proof.

$$\begin{aligned} &\text{build } (f z \mapsto \text{foldr } (xs x \mapsto \text{foldr } f x xs) z xss) \\ &= (f z \mapsto \text{foldr } (xs x \mapsto \text{foldr } f x xs) z xss) (:) [] \quad \text{definition of build} \\ &= \text{foldr } (xs x \mapsto \text{foldr } (:) x xs) [] xss \quad \beta\text{-reduction} \\ &= \text{foldr } \text{++} [] xss \quad \text{concatenate two lists} \\ &= \text{concat } xss \quad \text{concatenate multiple lists} \end{aligned}$$

□

Next for list *map*

Proof.

$$\begin{aligned}
 & build (\oplus z \mapsto foldr (y ys \mapsto (f y) \oplus ys) z xs) \\
 &= (\oplus z \mapsto foldr (y ys \mapsto (f y) \oplus ys) z xs) (:) [] \quad \text{definition of } build \\
 &= foldr (y ys \mapsto f(y) : ys) [] xs \quad \beta\text{-reduction} \\
 &= foldr (x ys \mapsto f(x) : ys) [] xs \quad \alpha \text{ transformation, change name} \\
 &= map f xs \quad \text{definition of list map}
 \end{aligned}$$

□

Next for *filter*

Proof.

$$\begin{aligned}
 & build (\oplus z \mapsto foldr (x xs' \mapsto \begin{cases} f(x) : & x \oplus xs' \\ \text{otherwise :} & xs' \end{cases}) z xs) \\
 &= (\oplus z \mapsto foldr (x xs' \mapsto \begin{cases} f(x) : & x \oplus xs' \\ \text{otherwise :} & xs' \end{cases}) z xs) (:) [] \quad \text{definition of } build \\
 &= foldr (x xs' \mapsto \begin{cases} f(x) : & x : xs' \\ \text{otherwise :} & xs' \end{cases}) [] xs \quad \beta\text{-reduction} \\
 &= filter f xs \quad \text{definition of filter}
 \end{aligned}$$

□

Last for *repeat*

Proof.

$$\begin{aligned}
 & build (\oplus z \mapsto let r = x \oplus r \text{ in } r) \\
 &= (\oplus z \mapsto let r = x \oplus r \text{ in } r) (:) [] \quad \text{definition of } build \\
 &= (let r = x : r \text{ in } r) \quad \beta\text{-reduction} \\
 &= repeat x \quad \text{definition of repeat}
 \end{aligned}$$

□

3. Simplify the quick sort algorithm.

$$\begin{cases} qsort [] = [] \\ qsort (x : xs) = qsort [a | a \in xs, a \leq x] ++ [x] ++ qsort [a | a \in xs, x < a] \end{cases}$$

First, we can transform the ZF-expression to *filter*, and combine the two rounds of list filtering to one pass:

$$\begin{cases} qsort [] = [] \\ qsort (x : xs) = qsort as ++ [x] ++ qsort bs \end{cases}$$

where:

$$\begin{aligned}
 (as, bs) &= foldr h ([][], []) xs \\
 h y (as', bs') &= \begin{cases} y \leq x : & (y : as', bs') \\ \text{otherwise :} & (as', y : bs') \end{cases}
 \end{aligned}$$

Next we further simplify the list concatenation:

$$\begin{aligned}
 & qsort as ++ [x] ++ qsort bs \\
 &= qsort as ++ (x : qsort bs) \\
 &= foldr(:) (x : qsort bs)(qsort as)
 \end{aligned}$$

4. Verify the type constraint of fusion law with category theory. Hint: consider the type of the catamorphism.

As shown in below diagram:

$$\begin{array}{ccc}
 \mathbf{ListFA} [A] & \xrightarrow{(:) + []} & [A] \\
 \mathbf{ListFA}(h) \downarrow & & \downarrow h = (\alpha) \\
 \mathbf{ListFA} B & \xrightarrow{\alpha = f + z} & B
 \end{array}$$

The catamorphism (α) is abstracted to build some algebraic structure from α , i.e. $g \alpha$. Where g accepts the α arrow of an F -algebra, generates result of B . The α arrow is the coproduct of $f : A \rightarrow B \rightarrow B$ and $z : 1 \rightarrow B$, that the type is:

$$g : \forall A. (\forall B. (A \rightarrow B \rightarrow B) \rightarrow B \rightarrow B)$$

The definition of build is $build(g) = g (:) []$. It applies g to the α arrow of the initial algebra, and builds the object of the initial algebra, which is a list of $[A]$.

$$build : \forall A. (\forall B. (A \rightarrow B \rightarrow B) \rightarrow B \rightarrow B) \rightarrow \mathbf{List} A$$

5. Use the fusion law to optimize the expression evaluation function:

$$eval = sum \circ map (product \circ (map dec))$$

$$\begin{aligned}
 & eval \ es \\
 = & sum (map (product \circ (map dec)) es) && \text{function composition} \\
 & \{sum \text{ in fold, map in build}\} \\
 = & \mathbf{foldr} (+) 0 (\mathbf{build} (\oplus z \mapsto \mathbf{foldr} (t \ ts \mapsto (f \ t) \oplus ts) z \ es)) && \text{let } f = product \circ (map dec) \\
 = & (\oplus z \mapsto \mathbf{foldr} (t \ ts \mapsto (f \ t) \oplus ts) z \ es) (+) 0 && \text{fusion law} \\
 = & \mathbf{foldr} (t \ ts \mapsto (f \ t) + ts) 0 \ es && \beta\text{-reduction}
 \end{aligned}$$

Written in point-free form as:

$$eval = \mathbf{foldr} (t \ ts \mapsto (f \ t) + ts) 0$$

Next we simplify the $product \circ (map dec)$ part

$$\begin{aligned}
 & (product \circ (map dec)) t \\
 = & product (map dec t) && \text{function composition} \\
 & \{product \text{ in fold, map in build}\} \\
 = & \mathbf{foldr} (\times) 1 (\mathbf{build} (\oplus z \mapsto \mathbf{foldr} (d \ ds \mapsto (dec \ d) \oplus ds) z \ t)) \\
 = & (\oplus z \mapsto \mathbf{foldr} (d \ ds \mapsto (dec \ d) \oplus ds) z \ t) (\times) 1 && \text{fusion law} \\
 = & \mathbf{foldr} (d \ ds \mapsto (dec \ d) \times ds) 1 \ t && \beta\text{-reduction} \\
 = & \mathbf{foldr} ((\times) \circ \mathbf{fork} (dec, id)) 1 \ t && \text{let } \mathbf{fork}(f, g) \ x = (f \ x, g \ x)
 \end{aligned}$$

Substitute this to f , we obtain the final simplified result:

$$eval = \mathbf{foldr} (t \ ts \mapsto (\mathbf{foldr} ((\times) \circ \mathbf{fork} (dec, id)) 1 \ t) + ts) 0$$

6. How to expand all expressions from left?

When expand from left to right, there are three options for every digit d :

- Insert nothing. It means append d to the last factor of the last sub-expression of e_i . Combine $f_n \# [d]$ as a new factor. For example when e_i is $1 + 2$, d is 3, write 3 after $1 + 2$ without inserting any symbols, we obtain a new expression $1 + 23$;
- Insert \times . It means we create a new factor $[d]$, then append it to the last sub-expression of e_i . Combine $t_m \# [[d]]$ as a new sub-expression. For the same $1 + 2$ example, we write 3 after it, put a \times between 2 and 3, hence obtain a new sub-expression $1 + 2 \times 3$;
- Insert $+$. It means we create a new sub-expression $[[d]]$, then append it to e_i to obtain a new expression $e_i \# [[d]]$. For the same $1 + 2$ example, we write 3 after it, put a $+$ between 2 and 3, hence obtain a new expression $1 + 2 + 3$.

We need define the append function to add an element after a list:

$$\text{append } x = \text{foldr } (:) [x]$$

Then we define a function $\text{onLast}(f)$, which applies f to the last element of a list:

$$\begin{aligned} \text{onLast } f &= \text{foldr } h [] \\ \text{where : } &\begin{cases} h \ x \ [] &= [(f \ x)] \\ h \ x \ xs &= x : xs \end{cases} \end{aligned}$$

Then we implement the above three expansion options:

```
add d exp = [((append d) `onLast`) `onLast` exp,
              (append [d]) `onLast` exp,
              (append [[d]]) exp]
```

7. The following definition converts expression to string:

$$str = (join "+") \circ (map ((join "\times") \circ (map (show \circ dec))))$$

Where $show$ converts number to string. Function $join(c, s)$ concatenates multiple strings s with delimiter c . For example: $join("#", ["abc", "def"]) = "abc#def"$. Use the fusion law to optimize str .

We defined $join(ws)$ in chapter 5. It insert space between every two strings. We can extract the space as a parameter to define $join(c, s)$:

$$join \ c = \text{foldr } (w \ b \mapsto \text{foldr } (:) (c : b) w) []$$

Observe the definition of str . It contains embedded $(join \ c) \circ (map \ f)$ as:

$$\begin{aligned} str &= (join \ c) \circ (map \ f) \\ \text{where : } f &= (join \ d) \circ (map \ g) \end{aligned}$$

where $c = '+'$, $d = '\times'$, and $g = show \circ dec$. What we need is to simplify $(join \ c) \circ (map \ f)$.

$$\begin{aligned}
& (join c) \circ (map f) es \\
& \quad \{join in fold, map in build\} \\
= & \textbf{foldr} (w b \mapsto \text{foldr} (:) (c : b) w) [] (\textbf{build} (\oplus z \mapsto \text{foldr} (y ys \mapsto (f y) \oplus ys) z es)) \\
& \quad \{\text{fusion law}\} \\
= & (\oplus z \mapsto \text{foldr} (y ys \mapsto (f y) \oplus ys) z es) (w b \mapsto \text{foldr} (:) (c : b) w) [] \\
& \quad \{\beta\text{-reduction}\} \\
= & \text{foldr} (y ys \mapsto \text{foldr} (:) (c : ys) (f y)) [] es
\end{aligned}$$

Substitute the $+$, \times , and $show \circ dec$ in, we obtain the final result:

$$\begin{aligned}
str = & \text{foldr} (x xs \mapsto \text{foldr} (:) ('+' : xs) (\\
& \quad \text{foldr} (y ys \mapsto \text{foldr} (:) (' \times' : ys) (show \circ dec y)) [])) []
\end{aligned}$$

.7 Infinity

1. In chapter 1, we realized Fibonacci numbers by folding. How to define Fibonacci numbers as potential infinity with *iterate*?

$$F = (fst \circ unzip) (\text{iterate} ((m, n) \mapsto (n, m + n)) (1, 1))$$

For example *take* 100 F gives the first 100 Fibonacci numbers

2. Define *iterate* by folding.

Consider the infinite stream $\text{iterate } f x$. After applying f to each element, and prepend x as the first one, we obtain this infinite stream again. Based on this fact, we can define it as:

$$\text{iterate } f x = x : \text{foldr} (y ys \mapsto (f y) : ys) [] (\text{iterate } f x)$$

For example:

```
take 10 $ iter (+1) 0
[0,1,2,3,4,5,6,7,8,9]
```

3. Use the definition of the fixed point in chapter 4, prove $Stream$ is the fixed point of $StreamF$.

Let $A' = \text{StreamF } E A$, then apply it to itself repeatedly. We call this result **Fix (StreamF E)**

$$\begin{aligned}
\text{Fix (StreamF E)} &= \text{StreamF } E (\text{Fix (StreamF E)}) && \text{definition of fixed point} \\
&= \text{StreamF } E (\text{StreamF } E (...)) && \text{expand recursively} \\
&= \text{Stream } E (\text{Stream } E (...)) && \text{change name} \\
&= \text{Stream } E && \text{reverse of Stream}
\end{aligned}$$

Therefore, $Stream$ is the fixed point of $StreamF$.

4. Define *unfold*.

We often use *Maybe* to define the terminate condition:

```
unfold :: (b -> Maybe (a, b)) -> (b -> [a])
unfold f b = case f b of
  Just (a, b') -> a : unfold f b'
  Nothing -> []
```

5. The fundamental theorem of arithmetic states that, any integer greater than 1 can be unique represented as the product of prime numbers. Given a text T , and a string W , does any permutation of W exist in T ? Solve this programming puzzle with the fundamental theorem and the stream of prime numbers.

Our idea is to map every unique character to a prime number, for example, $a \rightarrow 2$, $b \rightarrow 3$, $c \rightarrow 5$, ... Given any string W , no matter it contains repeated characters or not, we can convert it to a product of prime numbers:

$$F = \prod p_c, c \in W$$

We call it the number theory finger print F of string W . When W is empty, we define its finger print as 1. Because multiplication of integers is commutative, the finger print is same for all permutations of W , and according to the fundamental theorem of arithmetic, the finger print is unique. We can develop an elegant solution based on this: First, we calculate F of string W , then slide a window of length $|W|$ along T from left to right. When start, we also need to compute the finger print within this window of T , and compare it with F . If they are equal, it means T contains some permutation of W . Otherwise, we slide the window to the right by a character. We can easily compute the updated finger print value for this new window position: divide the product by the prime number of the character slides out, and multiply the prime number of the character slides in. Whenever the finger print equals to F , we find a permutation. In order to map different characters to prime numbers, we can use sieve of Eratosthenes to generate a series of prime numbers. Below is the example algorithm accordingly.

```

function CONTAINS?( $W, T$ )
   $P \leftarrow$  ana era [2, 3, ...] ▷ prime numbers
  if  $W = \phi$  then
    return True
  if  $|T| < |W|$  then
    return False
   $m \leftarrow \prod P_c, c \in W$ 
   $m' \leftarrow \prod P_c, c \in T[1...|W|]$ 
  for  $i \leftarrow |W| + 1$  to  $|T|$  do
    if  $m = m'$  then
      return True
     $m' \leftarrow m' \times P_{T_i} / P_{T_{i-|W|}}$ 
  return  $m = m'$ 

```

6. We establish the 1-to-1 correspondence between the rooms and guests. For guest i in group j , which room number should be assigned? Which guest in which group will live in room k ?

Use the convention to count from zero, and use pair (i, j) to denote the j -th guest in the i -th group. Let us list the first several guests and their rooms:

(i, j)	(0, 0)	(0, 1)	(1, 0)	(2, 0)	(1, 1)	(0, 2)	(0, 3)	(1, 2)	(2, 1)	(3, 0)	...
k	0	1	2	3	4	5	6	7	8	9	...
$i + j$	0	1	1	2	2	2	3	3	3	3	...

Writing down the values $i + j$, we can find the pattern. There are 1 instance of number 0, 2 instances of number 1, 3 instances of number 2, 4 instances of number 3, ... They are exactly the triangle numbers found by Pythagoreans. Let $m = i + j$,

there are total $\frac{m(m+1)}{2}$ grid points along the diagonals on the left-bottom side of a given grid.

For the diagonal where this point belongs to, if m is odd, then the room number increases along the left up direction, i increases, and j decreases; if m is even, then the direction is right-bottom. Summarize the two cases gives the following result:

$$k = \frac{m(m+1)}{2} + \begin{cases} m - j : m \text{ is odd} \\ j : m \text{ is even} \end{cases}$$

Further, we can use $(-1)^m$ to simplify the conditions:

$$k = \frac{m(m+2) + (-1)^m(2j - m)}{2}$$

7. For Hilbert's Grand hotel, there are multiple solutions for the problem on the third day. Can you give a different numbering scheme based on the cover page of the book *Proof without word*?

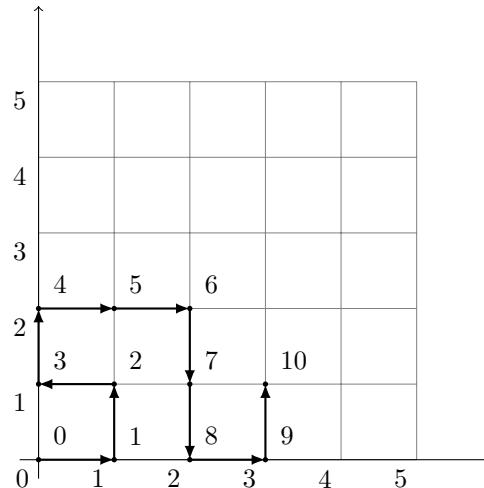


Figure 21: Another numbering scheme for infinity of infinity

As shown in figure 21, we count along the gnomon shaped path. There are odd number of grid points along every gnomon.

8. Let $x = 0.9999\dots$, then $10x = 9.9999\dots$, subtract them gives $10x - x = 9$. Solving this equation gives $x = 1$. Hence $1 = 0.9999\dots$ Is this proof correct?

Yes, it's correct.

9. Light a candle between two opposite mirrors, what image can you see? Is it potential or actual infinity?

The candle image reflects between the two mirrors endlessly, generates infinite many images. If we consider the speed of light is limited, then it is potential infinity from physics viewpoint.

.8 Paradox

1. We can define numbers in natural language. For example “the maximum of two digits number” defines 99. Define a set containing all numbers that cannot be described within 20 words. Consider such an element: “The minimum number that cannot be described within 20 words”. Is it a member of this set?

This is an instance of Russell’s paradox. Whether it is a member, all lead to contradiction.

2. “The only constant is change” said by Heraclitus. Is this Russell’s paradox?

Yes, this is an instance of Russell’s paradox.

3. Is the quote saying by Socrates (the beginning of chapter 7) Russell’s paradox?

Yes, it is an instance of Russell’s paradox.

4. Translate Fermat’s last theorem into a TNT string.

We need define power operation first.

$$\begin{cases} \forall a : e(a, 0) = S0 & \text{0-th power is 1} \\ \forall a : \forall b : e(a, Sb) = a \cdot e(a, b) & \text{recursion} \end{cases}$$

We can then define Fermat’s last theorem atop it.

$$\forall d : \neg \exists a : \exists b : \exists c : \neg(d = 0 \vee d = S0 \vee d = SS0) \rightarrow e(a, d) + e(b, d) = e(c, d)$$

5. Prove the associative law of addition with TNT reasoning rules.

Surprisingly, we can prove every theorem below:

$$\begin{aligned} a + b + 0 &= a + (b + 0) \\ a + b + S0 &= a + (b + S0) \\ a + b + SS0 &= a + (b + SS0) \\ &\dots \end{aligned}$$

For example:

$$a + b + 0 = a + b = a + (b + 0)$$

And:

$$\begin{aligned} a + b + SS0 &= SS(a + b + 0) \\ &= SS(a + b) \\ &= a + SSb \\ &= a + (b + SS0) \end{aligned}$$

However, we cannot prove: $\forall c : a + b + c = a + (b + c)$.

To do that, we has to introduce mathematical induction.

6. Prove that $\forall a : (0 + a) = a$ with the newly added rule of induction.

First for the case of 0:

$$0 + 0 = 0$$

Next suppose $(0 + a) = a$ holds, we have:

$$\begin{aligned} (0 + Sa) &= S(0 + a) && \text{axiom 3} \\ &= Sa && \text{induction hypothesis} \end{aligned}$$

From the rule of induction, we obtain: $\forall a : (0 + a) = a$

Proof of commutative law of addition

To prove the commutative law of addition $a + b = b + a$, we prepare three things. The first states that, for any natural number a , we have:

$$0 + a = a \quad (1)$$

It means the zero on the left side can be cancelled for addition. When $a = 0$, according to the first rule of addition, it holds:

$$0 + 0 = 0$$

As the induction, suppose $0 + a = a$ holds, we are going to show that $0 + a' = a'$.

$$\begin{aligned} 0 + a' &= (0 + a)' && \text{second rule of addition} \\ &= a' && \text{induction assumption} \end{aligned}$$

Next we define the successor of 0 be 1, and prove the second fact:

$$a' = a + 1 \quad (2)$$

It means the successor of any natural number is this number plus 1. This is because:

$$\begin{aligned} a' &= (a + 0)' && \text{first rule of addition} \\ &= a + 0' && \text{second rule of addition} \\ &= a + 1 && 1 \text{ is defined as the successor of 0} \end{aligned}$$

The third thing we are going to prove is the starting case:

$$a + 1 = 1 + a \quad (3)$$

When $a = 0$, we have:

$$\begin{aligned} 0 + 1 &= 1 && \text{We proved the left 0 can be cancelled} \\ &= 1 + 0 && \text{first rule of addition} \end{aligned}$$

For induction case, suppose $a + 1 = 1 + a$ holds, we are going to show $a' + 1 = 1 + a'$.

$$\begin{aligned} a' + 1 &= a' + 0' && 1 \text{ is the successor of 0} \\ &= (a' + 0)' && \text{first rule of addition} \\ &= ((a + 1) + 0)' && \text{second result we proved. (2)} \\ &= (a + 1)' && \text{second rule of addition} \\ &= (1 + a)' && \text{induction assumption} \\ &= 1 + a' && \text{second rule of addition} \end{aligned}$$

On top of these three results, we can prove the commutative law of addition. We first show that when $b = 0$ it holds. According to the first rule of addition, we have $a + 0 = a$; While from the first result we proved, $0 + a = a$ holds too. Hence $a + 0 = 0 + a$. Then we prove the induction case. Suppose $a + b = b + a$ holds, we are going to show $a + b' = b' + a$.

$$\begin{aligned}
 a + b' &= (a + b)' && \text{second rule of addition} \\
 &= (b + a)' && \text{induction assumption} \\
 &= b + a' && \text{second rule of addition} \\
 &= b + a + 1 && \text{second result we proved. (2)} \\
 &= b + 1 + a && \text{third result we proved. (3)} \\
 &= (b + 1) + a && \text{associative law proved in chapter 1} \\
 &= b' + a && \text{third result we proved. (3)}
 \end{aligned}$$

Therefore we proved commutative law of addition with Peano's axioms([10], p147 - 148).

Uniqueness of product and coproduct

Are product and coproduct unique? The following theorem answers this question.

Lemma .8.1. For any pair of object A, B of category \mathbf{C} , let the objects and arrows in below diagram

$$\begin{array}{ccc}
 \begin{array}{c} P \\ \swarrow p_A \quad \searrow p_B \\ A \\ \uparrow q_A \quad \downarrow q_B \\ B \\ \swarrow q_B \quad \searrow q_A \\ Q \end{array} & \quad &
 \begin{array}{c} I \\ \nearrow i_A \quad \searrow i_B \\ A \\ \uparrow j_A \quad \downarrow j_B \\ B \\ \swarrow j_B \quad \searrow j_A \\ J \end{array}
 \end{array}$$

be a pair of

product coproduct

wedges, then

$$P, Q \quad I, J$$

are isomorphic wedges. There are unique arrows:

$$\begin{array}{ccc}
 \begin{array}{c} P \\ \downarrow f \quad \uparrow g \\ Q \end{array} & \quad &
 \begin{array}{c} I \\ \downarrow f \quad \uparrow g \\ J \end{array}
 \end{array}$$

Such that:

$$\begin{cases} p_A = q_A \circ f & p_B = q_B \circ f \\ q_A = p_A \circ g & q_B = p_B \circ g \end{cases} \quad \begin{cases} i_A = g \circ j_A & i_B = g \circ j_B \\ j_A = f \circ i_A & j_B = f \circ i_B \end{cases}$$

Where f and g are inverse pair of isomorphisms.

Proof. We only prove the left side part. The right side can be proved in a similar way. Given A, B , Object Q and the pair q_A, q_B form a product wedge. Object P and the pair p_A, p_B form another wedge. From the definition of product, there is a unique mediator f satisfying:

$$p_A = q_A \circ f \quad \text{和} \quad p_B = q_B \circ f$$

By reversing the role of P and Q (Let P be product, and Q be arbitrary wedge), we have:

$$q_A = p_A \circ g \quad \text{和} \quad q_B = p_B \circ g$$

Hence we have:

$$\begin{cases} p_A \circ g \circ f = q_A \circ f = p_A \\ p_B \circ g \circ f = q_B \circ f = p_B \end{cases}$$

and:

$$\begin{cases} q_A \circ f \circ g = p_A \circ g = q_A \\ q_B \circ f \circ g = p_B \circ g = q_B \end{cases}$$

Therefore:

$$g \circ f = id_P \quad f \circ g = id_Q$$

□

This proved if two objects have product (or coproduct), then it is unique.

Cartesian product and disjoint union of sets form product and coproduct

Proof. We prove this by construction. The Cartesian product $A \times B$ contains all combinations from the two sets.

$$\{(a, b) | a \in A, b \in B\}$$

We define two special arrows (functions) as p_A and p_B :

$$\begin{cases} fst(a, b) = a \\ snd(a, b) = b \end{cases}$$

Consider an arbitrary wedge

$$\begin{array}{ccc} & & A \\ & \nearrow p & \\ X & & \\ & \searrow q & \\ & & B \end{array}$$

Where $p x = a, q x = b, x \in X$. For example, let X be Int , A be Int , and B be $Bool$, the two functions p and q are defined as:

$$\begin{cases} p(x) = -x \\ q(x) = even(x) \end{cases}$$

Such that p negates an integer, and q examines if it is even. We define the function $X \xrightarrow{m} A \times B$ as below:

$$m(x) = (a, b)$$

For this example, we have:

$$m(x) = (-x, even(x))$$

Such that, the following diagram commutes:

$$\begin{array}{ccc}
 & A & \\
 p \nearrow & \uparrow fst & \\
 X \xrightarrow{m} A \times B & & \downarrow snd \\
 q \searrow & & B
 \end{array}$$

Let us verify it:

$$\begin{aligned}
 (fst \circ m)(x) &= fst m(x) && \text{function composition} \\
 &= fst (a, b) && \text{definition of } m \\
 &= a && \text{definition of } fst \\
 &= p(x) && \text{reverse of } p
 \end{aligned}$$

and:

$$\begin{aligned}
 (snd \circ m)(x) &= snd m(x) && \text{function composition} \\
 &= snd (a, b) && \text{definition of } m \\
 &= b && \text{definition of } snd \\
 &= q(x) && \text{reverse of } q
 \end{aligned}$$

For the above example, we have:

$$\begin{cases} (fst \circ m)(x) = fst (-x, even(x)) = -x = p(x) \\ (snd \circ m)(x) = snd (-x, even(x)) = even(x) = q(x) \end{cases}$$

We also need prove the uniqueness of m . Suppose there exists another function $x \xrightarrow{h} A \times B$, such that the diagram also commutes:

$$fst \circ h = p \quad \text{and} \quad snd \circ h = q$$

We have:

$$\begin{aligned}
 (a, b) &= (p(x), q(x)) && \text{definition of } p, q \\
 &= ((fst \circ h)(x), (snd \circ h)(x)) && \text{commute} \\
 &= (fst h(x), snd h(x)) && \text{function composition} \\
 &= (fst (a, b), snd (a, b)) && \text{reverse of } fst, snd
 \end{aligned}$$

Hence $h(x) = (a, b) = m(x)$, which proves the uniqueness of m .

Next we prove the coproduct part. The elements in the disjoint union $A + B$ have two types. One comes from A as $(a, 0)$, the other comes from B as $(b, 1)$. We can define two special arrows (functions) as i_A and i_B :

$$\begin{cases} left(a) = (a, 0) \\ right(b) = (b, 1) \end{cases}$$

Consider the wedge of an arbitrary set X :

$$\begin{array}{ccc}
 A & & \\
 & \searrow p & \\
 & X & \\
 & \nearrow q & \\
 B & &
 \end{array}$$

We define arrow $A + B \xrightarrow{m} X$ as below:

$$\begin{cases} m(a, 0) = p(a) \\ m(b, 1) = q(b) \end{cases}$$

Such that the following diagram commutes:

$$\begin{array}{ccc} A & & \\ \downarrow \text{left} & \searrow p & \\ A + B & \xrightarrow{m} & X \\ \uparrow \text{right} & \nearrow q & \\ B & & \end{array}$$

Next we need prove the uniqueness of m . Suppose there exists another arrow $A + B \xrightarrow{h} X$, also makes the diagram commutes:

$$h \circ \text{left} = p \quad \text{and} \quad h \circ \text{right} = q$$

Taking any $a \in A, b \in B$, we have:

$$\begin{cases} h(a, 0) = h(\text{left}(a)) = (h \circ \text{left})(a) = p(a) = m(a, 0) \\ h(b, 1) = h(\text{right}(b)) = (h \circ \text{right})(b) = q(b) = m(b, 1) \end{cases}$$

Hence $h = m$, which proves the uniqueness of m . □

Bibliography

- [1] Wikipedia. “History of ancient numeral systems”. https://en.wikipedia.org/wiki/History_of_ancient_numeral_systems
- [2] Calvin C Clawson. “The Mathematical Traveler, Exploring the Grand History of Numbers”. Springer. 1994, ISBN: 9780306446450
- [3] Wikipedia. “Babylonian numerals”. https://en.wikipedia.org/wiki/Babylonian_numerals
- [4] Morris Kline “Mathematics: The Loss of Certainty”. Oxford University Press, 1980. 0-19-502754-X.
- [5] Douglas R. Hofstadter “Gödel, Escher, Bach: An Eternal Golden Braid”. Basic Books; Anniversary edition (February 5, 1999). ISBN: 978-0465026562
- [6] Richard Bird, Oege de Moor. “Algebra of Programming”. University of Oxford, Prentice Hall Europe. 1997. ISBN: 0-13-507245-X.
- [7] 顾森《浴缸里的惊叹》 People’s postal Press. 2014, ISBN: 9787115355744
- [8] 韩雪涛 “数学悖论与三次数学危机”. People’s postal Press. 2016, ISBN: 9787115430434
- [9] Morris Klein. “Mathematical thought from Ancient to Modern Times, Vol. 1”. Oxford University Press; 1972. ISBN: 9780195061352
- [10] Alexander A. Stepanov, Daniel E. Rose “From Mathematics to Generic Programming”. Addison-Wesley Professional; 1 edition (November 17, 2014) ISBN-13: 978-0321942043
- [11] Euclid, Thomas Heath (Translator) “Euclid’s Elements (The Thirteen Books) ”. DigiReads.com Publishing (December 26, 2017). ISBN-13: 978-1420956474
- [12] 韩雪涛 “好的数学——“下金蛋”的数学问题”. 湖南科学技术出版社. 2009, ISBN: 9787535756725
- [13] Wikipedia “Bézout’s identity” https://en.wikipedia.org/wiki/B%C3%A9zout%27s_identity
- [14] 刘新宇 “算法新解” 人民邮电出版社. 2017, ISBN: 9787115440358
- [15] Wikipedia “Alan Turing” https://en.wikipedia.org/wiki/Alan_Turing
- [16] by Serge Abiteboul (Author), Gilles Dowek (Author), K-Rae Nelson (Translator) “The Age of Algorithms”. Cambridge University Press (December 31, 2019) ISBN-13: 978-1108745420

- [17] Simon L. Peyton Jones. “The implementation of functional programming language”. Prentice Hall. 1987, ISBN: 013453333X
- [18] 韩雪涛 “好的数学——方程的故事”. 湖南科学技术出版社. 2012, ISBN: 9787535770066
- [19] Wikipedia “Galois theory”. https://en.wikipedia.org/wiki/Galois_theory
- [20] Wikipedia “Évariste Galois”. https://en.wikipedia.org/wiki/Évariste_Galois
- [21] Anita R Singh. “The Last Mathematical Testament of Galois” Resonance Oct 1999. pp93-100.
- [22] Sawilowsky, Shlomo S. and Cuzzocrea, John L. (2005) “Joseph Liouville’s ‘Mathematical Works Of Évariste Galois’,” Journal of Modern Applied Statistical Methods: Vol. 5 : Iss. 2 , Article 32. DOI: 10.22237/jmasm/1162355460
- [23] Wikipedia “Rubik’s Cube group”. https://en.wikipedia.org/wiki/Rubik's_Cube_group
- [24] 张禾瑞 “近世代数基础”. 高等教育出版社. 1978, ISBN: 9787040012224
- [25] M.A. Armstrong “Groups and Symmetry”. Springer. 1988. ISBN: 0387966757.
- [26] Wikipedia “Joseph-Louis Lagrange”. https://en.wikipedia.org/wiki/Joseph-Louis_Lagrange
- [27] Wikipedia “Proofs of Fermat’s little theorem”. https://en.wikipedia.org/wiki/Proofs_of_Fermat's_little_theorem
- [28] Wikipedia “Leonhard Euler”. https://en.wikipedia.org/wiki/Leonhard_Euler
- [29] Wikipedia “Carmichael number”. https://en.wikipedia.org/wiki/Carmichael_number
- [30] Sanjoy Dsgupta, Christos Papadimitriou, Umesh Vazirani. “Algorithms”. McGraw-Hill Education. Sept. 2006. ISBN: 9780073523408
- [31] Wikipedia “Miller-Rabin primality test”. https://en.wikipedia.org/wiki/Miller-Rabin_primality_test
- [32] Wikipedia “Emmy Noether”. https://en.wikipedia.org/wiki/Emmy_Noether
- [33] 章璞. “伽罗瓦理论：天才的激情”. 高等教育出版社. 2013. ISBN: 9787040372526
- [34] John Stillwell. “Galois Theory for Beginners”. The American Mathematical Monthly, Vol. 101, No. 1 (Jan., 1994), pp. 22-2
- [35] Dan Goodman. “An Introduction to Galois Theory”. <https://nrich.maths.org/1422>
- [36] Michael Artin. “Algebra (Second Edition)”. Pearson. Feb. 2017. ISBN: 9780134689609
- [37] Hermann Weyl. “Symmetry”. Princeton University Press; Reprint edition (October 4, 2016). ISBN: 978-0691173252

- [38] Jean Dieudonne. "Mathematics —The Music of Reason". Springer Science and Business Media, Jul 20, 1998. ISBN: 9783540533467
- [39] Haskell Wiki. "Monad". <https://wiki.haskell.org/Monad>
- [40] Wikipedia. "Samuel Eilenberg". https://en.wikipedia.org/wiki/Samuel_Eilenberg
- [41] Wikipedia. "Saunders Mac Lane". https://en.wikipedia.org/wiki/Saunders_Mac_Lane
- [42] Harold Simmons. "An introduction to Category Theory". Cambridge University Press; 1 edition, 2011. ISBN: 9780521283045
- [43] Wikipedia. "Tony Hoare". https://en.wikipedia.org/wiki/Tony_Hoare
- [44] Wadler Philip. "Theorems for free!". Functional Programming Languages and Computer Architecture, pp. 347-359. Asociation for Computing Machinery. 1989.
- [45] Bartosz Milewski. "Category Theory for Programmers". <https://bartoszmilewski.com/2014/10/28/category-theory-for-programmers-the-preface/>
- [46] Peter Smith. "Category Theory - A Gentle Introduction". http://www.academia.edu/21694792/A_Gentle_Introduction_to_Category_Theory_Jan_2018_version_
- [47] Wikipedia. "Exponential Object". https://en.wikipedia.org/wiki/Exponential_object
- [48] Manes, E. G. and Arbib, M. A. "Algebraic Approaches to Program Semantics". Texts and Monographs in Computer Science. Springer-Verlag. 1986.
- [49] Lambek, J. "A fixpoint theorem for complete categories". Mathematische Zeischrift, 103, pp.151-161. 1968.
- [50] Wikibooks. "Haskell/Foldable". <https://en.wikibooks.org/wiki/Haskell/Foldable>
- [51] Mac Lane. "Categories for working mathematicians". Springer-Verlag. 1998. ISBN: 0387984038.
- [52] Andrew Gill, John Launchbury, Simon L. Peyton Jones. "A Short Cut to Deforestation". Functional programming languages and computer architecture. pp. 223-232. 1993.
- [53] Richard Bird. "Pearls of Functional Algorithm Design". Cambridge University Press; 1 edition November 1, 2010. ISBN: 978-0521513388.
- [54] Ralf Hinze, Thomas Harper, Daniel W. H. James. "Theory and Practice of Fusion". 2010, 22nd international symposium of IFL (Implementation and application of functional languages). pp.19-37.
- [55] Akihiko Takano, Erik Meijer. "Shortcut Deforestation in Calculational Form". Functional programming languages and computer architecture. pp. 306-313. 1995.
- [56] Donald Knuth. "The Art of Computer Programming, Volume 4, Fascicle 4: Generating All Trees." Reading, MA: Addison-Wesley. ISBN: 978-0321637130. 2006.

- [57] Jean-Pierre Luminet, Marc Lachièze-Rey. “De l'infini - Horizons cosmiques, multivers et vide quantique”. DUNOD, 2016. ISBN: 9782100738380
- [58] 野口哲典. “数学的センスが身につく練習帳”. ソフトバンククリエイティブ 2007. ISBN: 9784797339314
- [59] Wikipedia. “Googol”. <https://en.wikipedia.org/wiki/Googol>
- [60] Wikipedia. “Zeno's Paradoxes”. https://en.wikipedia.org/wiki/Zeno's_paradoxes
- [61] 张锦文, 王雪生 “连续统假设”. 世界数学名题欣赏丛书。辽宁教育出版社. 1988. ISBN: 7-5382-0436-9/G·445
- [62] Richard Courant, Herbert Robbins, Reviewed by Ian Stewart. “What Is Mathematics? An Elementary Approach to Ideas and Methods 2nd Edition”. Oxford University Press, 1996, ISBN: 978-0195105193.
- [63] Henri Poincaré. “Science and Hypothesis”. Franklin Classics. 2018. ISBN: 9780342349418
- [64] Leon A. Gatys, Alexander S. Ecker, Matthias Bethge. “A Neural Algorithm of Artistic Style.” 2015. arXiv:1508.06576 [cs.CV] IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 2017.
- [65] 顾森《思考的乐趣——Matrix67 数学笔记》人民邮电出版社. 2012, ISBN: 9787115275868
- [66] Harold Abelson, Gerald Jay Sussman, Julie Sussman. “Structure and Interpretation of Computer Programs”. MIT Press, 1984; ISBN 0-262-01077-1
- [67] Henri Poincaré. “The Value of Science”. Modern Library, 2001. ISBN: 978-0375758485
- [68] Constance Ried. “Hilbert”. Springer, 1st Printing edition, 1996, ISBN: 978-0387946740
- [69] Paul Lockhart. “Measurement”. Belknap Press: An Imprint of Harvard University Press; Reprint edition 2014, ISBN: 978-0674284388

Index

- ω incomplete, 241
- absorption law, 127
- acid rain law, 170
- actual infinity, 182
- algebraic numbers, 197
- AlphaGo, 222
- anamorphism, 191
- apply, 139
- Archimedes, 185
- Aristotle, 183
- arrow, 107
- automorphism, 60, 94
- axiom of Archimedes, 185
- axiom of choice, 234
- axiom of reducibility, 229
- Banach-Tarski paradox, 235
- Barber paradox, 226
- Bicartesian closed, 141
- bifunctor, 125
- bimap, 127
- binary functor, 125
- Brouwer, 230
- Cancellation law, 124
- Cantor, 198
- Cantor's theorem, 208
- cardinal number, 209
- Carl Friedrich Gauss, 48
- Cartesian closed, 141
- Cartesian product, 119
- catamorphism, 151
- category, 107
- category of partial order set, 110
- category of pre-order set, 110
- Category of sets under total functions, 109
- CH, 211
- coalgebra, 190
- commutative ring, 88
- concatMap, 168
- continuum hypothesis, 211
- contravariance, 113
- coproduct, 121
- countable set, 203
- covariance, 113
- Currying, 140
- cyclic group, 67
- Dedekind, 200
- Dedekind cut, 206
- Deep Blue, 221
- deforestation, 170
- Descartes, 119
- diagonal method, 203
- division ring, 90
- Emmy Noether, 86
- equational theory, 141
- Euler, 82
- Euler function, 80
- Euler Theorem, 80
- exponential, 137
- exponential object), 139
- exponentials, 139
- F-algebra, 147
- F-coalgebra, 147
- Fermat's little theorem, 77
- field, 90
- field extension, 92
- flatMap, 168
- fmap, 115
- foldr/build fusion law, 164
- Fraenkel, 234
- Frege, 228
- functor, 113
 - constant functor, 113
 - id functor, 113
 - identity functor, 113
 - list functor, 116
 - maybe functor, 113
- Fundamental theorem of algebra, 48
- Fundamental theorem of Galois theory, 95
- fusion law, 125
- Galileo's paradox, 193
- Galois, 49

- Galois group, 94
- GCH, 211
- Gerolamo Cardano, 48
- Group, 53
- Gödel, 236
- Gödel numbering, 241
- Gödel's incompleteness theorems, 237
- Hamming numbers, 202
- heap, 56
- Hilbert, 231
- homomorphism, 59
- index of subgroup, 73
- initial algebra, 151
- initial object, 132
- integral domain, 90
- invariant subgroup, 73
- isomorphism, 60
- kernel, 73
- Lagrange, 74
- Lagrange's theorem, 76
- Lambek theorem, 151
- lazy evaluation, 188
- left coset, 72
- liar paradox, 225
- mediating arrow, 122
- mediator, 122
- method of exhaustion, 184
- monoid, 55
- Monoid category, 109
- natural isomorphism, 131
- natural transformation, 128
- Non-Euclidean geometry, 213
- normal subgroup, 73
- Object arithmetic, 141
- ordinal number, 208
- pairing heap, 57
- partial order set, 109
- Pierre de Fermat, 79
- polynomial functors, 143
- poset, 109
- potential infinity, 182
- power set, 208
- pre-order set, 109
- preset, 109
- primitive recursive function, 243
- product, 121
- quotient group, 73
- rank-2 type polymorphic, 168
- reflection law, 124
- regular numbers, 202
- right coset, 71
- ring, 88
- root field, 92
- RSA cryptosystem, 84
- Russell, 227
- Russell's paradox, 226
- Samuel Eilenberg, 104
- Saunders Mac Lane, 104
- selection arrow, 135
- semigroup, 57
- semiring, 90
- shortcut fusion, 168
- skew heap, 56
- splitting field, 92
- subgroup, 69
- terminal object, 132
- The order of the group element, 59
- theory of types, 229
- TNT system, 238
- transfinite numbers, 208
- transformation group, 62
- Turning's halting problem, 224
- Typographical Number Theory, 238
- uncountable set, 204
- wedge, 121
- Zeno, 179
- Zeno's paradox, 179
- Zermelo, 234
- zero divisor, 89
- zero object, 133
- ZF system, 234
- ZFC system, 235

GNU Free Documentation License

Version 1.3, 3 November 2008

Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc.

<http://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document “free” in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of “copyleft”, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The “**Document**”, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as “**you**”. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A “**Modified Version**” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A “**Secondary Section**” is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a

textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The “**Invariant Sections**” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The “**Cover Texts**” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A “**Transparent**” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not “Transparent” is called “**Opaque**”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The “**Title Page**” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

The “**publisher**” means any person or entity that distributes copies of the Document to the public.

A section “**Entitled XYZ**” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as “**Acknowledgements**”, “**Dedications**”, “**Endorsements**”, or “**History**”.) To “**Preserve the Title**” of such a section when you modify the Document means that it remains a section “Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- H. Include an unaltered copy of this License.
- I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
- N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
- O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's

Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled “Acknowledgements”, “Dedications”, or “History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License “or any later version” applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can

be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

“Massive Multiauthor Collaboration Site” (or “MMC Site”) means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A “Massive Multiauthor Collaboration” (or “MMC”) contained in the site means any set of copyrightable works thus published on the MMC site.

“CC-BY-SA” means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

“Incorporate” means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is “eligible for relicensing” if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright © YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled “GNU Free Documentation License”.

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the “with ... Texts.” line with this:

with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.