Proceedings of the 3rd International Conference on Computing Innovation and Applied Physics
DOI: 10.54254/2753-8818/9/20240729

An approach to simulate basic concepts of mathematics with object-oriented programming

Siyang Dai

Huazhong University of Science and Technology, Wuhan, 430074, China

siyangdai@hust.edu.cn

Abstract. Traditional approaches to using data structures mainly focus on the improvement in algorithm efficiency but rarely take advantage of object-oriented programming that can simulate the definition of mathematic concepts in a way that is similar to human thinking mode, so traditional approaches cannot satisfy the need of simulating sets and their features and operations for mathematics studies. In this paper, the author pointed out the disadvantages of traditional ways, proposed a series of hypotheses describing the relationship between sets and classes such as inheriting and inclusion relationships and a way based on those hypotheses using features of object-oriented programming, and used C#, one of the best object-oriented programming languages, to simulate sets and their features. In addition, for each hypothesis, the author raised examples with C# codes that realize the theory in this paper, clearly showing the approach proposed in this paper and why it is both efficient and elegant.

Keywords: Set Theory, Object-Oriented Programming, C Sharp.

1. Introduction

As the most popular programming mode, object-oriented programming (OOP) is widely used in various industries. In mathematics, it is a subject that is highly related to programming and computer science. In this paper, the author proposed a way to describe those mathematical concepts with OOP language. First, to raise their disadvantages, traditional ways of expressing a set [1] in programming language were shown. Then, the author proposed a new way to simulate sets with classes. At last, the author explained how to simulate concepts related to sets such as relations and algebra operations through methods of classes. For each hypothesis, at least one example in which the hypothesis is true was given. This paper proposed a new perspective to review the meaning of OOP and the relationship between programming and mathematics. Additionally, it may provide programmers with an easier way to learn and understand mathematics with programming knowledge.

This paper chose C# as the example of OOP language and all codes were provided in C#. This choice was made because the author considered that C# is not only widely used in different fields such as the auto-controlling system [2], as well as some subjects like geochemistry, to simplify the calculation and improve efficiency [3], but also some of its language features are suitable for doing mathematics work in this paper. At the same time, it should be noticed that all the codes in this paper should be easily transferred to other OOP languages such as C++ and Java.

DOI: 10.54254/2753-8818/9/20240729

2. Traditional ways to express sets and their problems

"Set" is one of the most basic concepts in mathematics. From the first day an object-oriented programming language was created, there should have been a data structure corresponding to this concept in those languages' internal libraries. There are widely used "unordered_set" and "set" in the standard template library (STL) of C++ [4]. Research in advance showed that the use rate of a set is up to 7.35% in all uses of the C++ STL and that of an unordered set is 0.024% [5]. There are "HashSet" and "SortedSet" in C# [6] with "HashSet" and "TreeSet" in Java [7], and they do similar work.

2.1. Example 1

Suppose that Alice has four lucky numbers. They are 5, 7, 9, and 11. Now Alice wants to get a set containing these four numbers. In mathematics, this set can be described as follows:

$\{5,7,9,11\}$

Or as follows:

$\{x \in Z \mid x \nmid 2 \wedge x \geq 5 \wedge \leq 11\}$ ○1

And in traditional ways, the code is probably as follows:

```
HashSet<int> luckyNumberSet1=new(){5,7,9,11};
```

Or as follows:

```
HashSet<int> luckyNumberSet1=new();
luckyNumberSet1.Add(5);
luckyNumberSet1.Add(7);
luckyNumberSet1.Add(9);
luckyNumberSet1.Add(11);
```

Notice that since these numbers follow a regularity, the code can also be as follows:

```
HashSet<int>luckyNumberSet2=new();
for (int i=5; i<=11; i+=2)
{
    luckyNumberSet2.Add(i);
}
```

Although these data structures are perfectly designed, and high-level languages can provide interfaces like "ISet" in C# [6] and "Set" in Java [7] and give the freedom of customizing users' own sets, still, they cannot satisfy some needs. For example, they cannot describe a set which contains infinite elements.

2.2. Example 2

Suppose that Bob's lucky numbers are all odd numbers that are greater than or equal to 5. Now Bob wants to get a set containing these numbers. In mathematics, this set can be described as follows:

$\{5,7,9,11, \ldots\}$

Here "…" means infinite elements after 11 and these elements follow the same law as the four explicitly written numbers.

Or as follows:

$\{x \in Z \mid x \nmid 2 \wedge x \geq 5\}$ ○2

And in traditional ways, the code can be written as follows:

```
HashSet<int>luckyNumberSet3=new();
for (int i=5; i+=2)
{
    luckyNumberSet3.Add(i);
}
```

If Bob runs this code, Bob will meet some trouble: this loop will never end until all the memory of Bob's computer is occupied and finally ends with an error. And of course, if there are any further operations that should be done after this set initializing is complete, they are unable to be done since this

DOI: 10.54254/2753-8818/9/20240729

set can never complete its initializing. In one word, it does not work as Bob expects. Another problem is that these data structures do not obey the Axiom of Regularity. See the code below:

```
HashSet<Object> S=new();
S.Add(S);
```

The code above adds S itself as an element of S. Both compiler and runtime do not check this. This may provide some convenience when programming but can also cause some problems. That is why in this paper, the author considers naming these data structures as "set" confusing—in fact, there is something that is more suitable to describe "set" in mathematics, which is "class", the basic concept in OOP. In other words, "class" in OOP is nearer to the concept of "set" in mathematics instead of those data structures ("HashSet", "TreeSet", etc.).

2.3. Example 3

For the case in Example 2, Bob can write codes as follows:

```
class ElementOfLuckyNumberSet4
{
    private int value;
    public ElementOfLuckyNumberSet4(dynamic x)
    {
        if (x is int && x % 2 != 0 && x >= 5)
        {
            this.value =x;
        }
        else
        {
            throw new ArgumentException("Invalid value");
        }
    }
}
```

The codes above define a class called "ElementOfLuckyNumberSet6". There is only one constructor method receiving a dynamic type x as a parameter and checking the conditions in ○2. If the conditions are not satisfied, then it throws an exception. Now, Bob can assume that the class "ElementOfLuckyNumberSet6" represents the set which is what Bob wants in Example 2. Notice that Bob does not use any data structures and this "set" is not saved in the runtime memory as usual but saved in the code (or IL code of the .Net structure) and becomes static. However, the throw statement makes the exception happen in runtime. This asynchronism seems not good but there is no need to worry because modern object-oriented programming languages provide the users with powerful tools such as Emit for dynamic reflection, and users can just use it to move the definition of their classes to runtime. However, to make this paper easily understood, those classes will be written in the style of Example 3 instead of Emit statements.

The code in Example 3 seems not correct in some cases (if Bob put any x which is greater than the Int32.MaxValue=2147483648 [6] as input, then it will go overflow) and is not elegant enough. This paper will discuss how to improve it later. But at first, some hypotheses should be introduced. Considering the original intention of the concept "class" in OOP is to simulate the concept "class" in Logic and the Logic's "class" is a concept beyond the concept "set" in mathematics, thus, if some restrictions are added to the "class" in OOP, it is able to simulate the concept "set" in mathematics.

3. A new way to simulate sets with classes

3.1. Definition 1

If a (OOP's) class contains exactly one protected field and exactly one public constructor, and this constructor has exactly one dynamic parameter, then it is a f1c1class. Since it has only one field, this

DOI: 10.54254/2753-8818/9/20240729

field will be called “the class (or the object)’s field” without specifying the name of the field. The constructor is the same, too. It is easy to see that ElementOfLuckyNumberSet4 is a f1c1class. A f1c1classes has many interesting features, and this paper will discuss them later.

3.2. Definition 2

A “possible instance” of a f1c1class is an instance whose life cycle can last outside of the expression where the new operator [8] is located. Every instance of a f1c1class that is not a possible instance is an impossible instance, which means it will be collected by the garbage collector [9] before the construction finish, so that it cannot be used in any other part of the code. Obviously, programmers can intentionally make an instance impossible by throwing an exception in the constructor.

3.3. Hypothesis 1

Every f1c1class A can simulate a set SA and every set SA can be simulated by a f1c1class A. That is to say, f1c1class-set mapping is a bijection. It is a mapping from the set of all possible f1c1classes which can be written in an object-oriented programming language to the set of all the sets with a rule that for each f1c1class A, all of its different possible instances form a set. And if A simulates SA or SA is simulated by A, each of the different possible instances of A can simulate one different element in the set SA, and vice versa. Hypothesis 1 may be hard to prove, but between some features of f1c1classes and sets, there are some similarities. For example, the inheritance relationship of f1c1classes is very similar to the relationship between subsets and supersets. Although a set must have a subset that is the set itself, while a class cannot inherit from itself, it is still safe to see inheritance the same as inclusion. Consider that there is a f1c1class A. B is a subclass of A, and it is shaped like :

```
class B:A
{
public B(dynamic x):base((object)x){}
}
```

B is also a f1c1class because A is a f1c1class so that b can inherit A’s protected field while B cannot inherit A’s constructor. So, here B’s own constructor is written so that B is a f1c1class. And since the constructor of B does nothing else but only call A’s constructor and pass its parameter to A’s constructor, there is no problem to say “B equals A” because A and B are in fact the same, that is to say, the set S simulated by A is exactly the same set as what simulated by B. Notice that B is a subclass of A and S is a subset of itself. So, here is the second Hypothesis.

3.4. Hypothesis 2

If a f1c1class A has a superclass B, then A simulates a set SA which is a subset of the set SB simulated by B. Now the conditions in ○2 are analyzed. ○2 contains three conditions and each of them refers to a set. $x \in Z$ means that the universal set in the case this paper is discussing is the set of all integers and of course, x belongs to the integer set; $x \nmid 2$ means that x belongs to the set of all odd integers, which is a subset of the integer set; $x \geq 5$ means that x belongs to the set of all integers greater than or equal to 5, which is another subset of the integer set. The next parts illustrate how to write codes to express these three sets.

3.5. Example 4

C# provided the BigIntegers class [6] to store very big integers. Regardless of hardware limitation, suppose that it can store almost as large as positive and negative infinitely large integers. Here the code just takes this advantage to build a f1c1class to simulate a set of all integers, like the code below:

```
class ElementOfIntegerSet
{
    protected System.Numerics.BigInteger X;
    public ElementOfIntegerSet(dynamic x)
    {
```

DOI: 10.54254/2753-8818/9/20240729

```
            this.X = new(x);
        }
    }
```

This is a f1c1class, and it is easy to explain why this class ElementOfIntegerSet can simulate the set of all integers. For every different integer x, there exists an instance of this class whose field is named "X", which is the only field of this f1c1class, whose value equals x. And for every different instance of this class, there exists an integer x which equals the value of the only field "X" of the instance. The necessary work is to call the only constructor and pass the integer one wants as a parameter.

For anything that is not an integer, for example, a string like "apple", if it is passed as the parameter to the constructor, it will throw an exception when trying to call BigInteger's constructor. And, if a rational number which is not an integer like 2.5 is passed, then the decimal section will be ignored and the result will be an instance whose value of field is 2. Thus, there do not exist any instances of this class whose field "X" is not an integer.

In conclusion, all possible different instances of this class form a set, which has a bijection to the set of all integers. So, that is why this f1c1class ElementOfIntegerSet is "simulating" the set of all integers.

3.6. Example 5

Here is the class simulating the set of all integers greater than or equal to 5. Since it is a subset of the set of all integers, it just inherits the ElementOfIntegerSet class.

```
class ElementOfSetOfAllIntegersGreaterThanOrEqualTo5 : ElementOfIntegerSet
{
    public ElementOfSetOfAllIntegersGreaterThanOrEqualTo5(dynamic x) : base((object)x)
    {
        ElementOfIntegerSet e5 = new(5);
        if (!(this >= e5))
        {
            throw new();
        }
    }
}
```

This subclass contains a constructor. It is first called superclass's constructor and this checks whether x is an integer. Then it announced and created an ElementOfIntegerSet's instance e5 whose value of the field is 5. Then it judges whether the new instance of ElementOfSetOfAllIntegersGreaterThanOrEqualTo5 is not greater than or equal to e5. If it is not greater than or equal to e5, then it throws an exception. Obviously, it is also a f1c1class, and all possible different instances of this set are simulating the set of all integers greater than or equal to 5.

3.7. Example 6

Similarly, this is the class simulating the set of all odd integers. Since it is a subset of the set of all integers, it just inherits the ElementOfIntegerSet class.

```
class ElementOfSetOfAllOddIntegers: ElementOfIntegerSet
{
    public ElementOfSetOfAllOddIntegers(dynamic x) : base((object)x)
    {
        ElementOfIntegerSet e2 = new(2);
        ElementOfIntegerSet e1 = new(1);
        if (!(this % e2==e1))
        {
            throw new();
        }
    }
```

DOI: 10.54254/2753-8818/9/20240729

```
}
```

Obviously, it is also a f1c1class, and all possible different instances of this set are simulating the set of all odd integers. Notice that the set of lucky numbers Bob wants in Example 2 is the intersection of the two sets in Example 5 and Example 6. To find a way to create a class to simulate the intersection of sets, the hypothesis below is proposed:

3.8. Hypothesis 3

For a f1c1class C simulating the set SC which is the intersection of two sets SA which is simulated by f1c1class A and SB which is simulated by f1c1class B, C's constructor's method body is the join of A's constructor's method body and B's constructor's method body.

3.9. Example 7

More specifically, if Bob writes codes as follows:

```
class ElementOfSetOfAllOddIntegersGreaterThanOrEquals5 : ElementOfIntegerSet
{
    public ElementOfSetOfAllOddIntegersGreaterThanOrEquals5(dynamic x) : base((object)x)
    {

    }
}
```

And copy the 5th to the 9th line in the code in Example 5 together with the 5th to the 10th line in the code in Example 6 and paste them into the 5th line in the code above. Then, Bob will get:

```
class ElementOfSetOfAllOddIntegersGreaterThanOrEqualTo5 : ElementOfIntegerSet
{
    public ElementOfSetOfAllOddIntegersGreaterThanOrEqualTo5(dynamic x) : base((object)x)
    {
        ElementOfIntegerSet e5 = new(5);
        if (!(this >= e5))
        {
            throw new();
        }
        ElementOfIntegerSet e2 = new(2);
        ElementOfIntegerSet e1 = new(1);
        if (!(this % e2 == e1))
        {
            throw new();
        }
    }
}
```

Inspecting the code above, Bob can find that this class is a f1c1class and it exactly simulates the set Bob wants in Example 2.

3.10. Hypothesis 4

For a f1c1class C simulating the set SC which is the union of two sets SA which is simulated by f1c1class A, and SB which is simulated by f1c1class B, C's constructor's method body is A's constructor's method body replacing every throw statement with B's constructor's method body, or B's constructor's method body replacing every throw statement with A's constructor's method body.

3.11. Example 8

For the first case in Hypothesis 4, the first step is to code as follows:

```
class ElementOfSetOfAllOddIntegersAndAllIntegersGreaterThanOrEquals5 : ElementOfIntegerSet
```

DOI: 10.54254/2753-8818/9/20240729

```
{
    public ElementOfSetOfAllOddIntegersGreaterThanOrEquals5(dynamic x) : base((object)x)
    {

    }
}
```

Then put ElementOfSetOfAllIntegersGreaterThanOrEqualTo5's constructor's method body to the 5th line:

```
class ElementOfSetOfAllOddIntegersAndAllIntegersGreaterThanOrEquals5 : ElementOfIntegerSet
{
    public ElementOfSetOfAllOddIntegersGreaterThanOrEquals5(dynamic x) : base((object)x)
    {
        ElementOfIntegerSet e5 = new(5);
        if (!(this >= e5))
        {
            throw new();
        }
    }
}
```

There is only one throw statement here in the 8th line. So, replace the throw statement with ElementOfSetOfAllOddIntegers's constructor's method body:

```
class ElementOfSetOfAllOddIntegersAndAllIntegersGreaterThanOrEquals5 : ElementOfIntegerSet
{
    public ElementOfSetOfAllOddIntegersGreaterThanOrEquals5(dynamic x) : base((object)x)
    {
        ElementOfIntegerSet e5 = new(5);
        if (!(this >= e5))
        {
            ElementOfIntegerSet e2 = new(2);
        ElementOfIntegerSet e1 = new(1);
        if (!(this % e2==e1))
        {
            throw new();
        }
        }
    }
}
```

The code above shows a class which is also a f1c1class, and the only situation that the throw statement would be executed is that x is neither greater than 5 nor an odd integer. Such that this class ElementOfSetOfAllOddIntegersAndAllIntegersGreaterThanOrEquals5 simulates the set of all odd integers and all integers greater than or equal to 5 which is the union of two sets: the set of all odd integers and the set of all integers greater than 5.

4. Relations and algebra operations

In the 5th line of the code in Example 5, there is a judgement "this>=e5". Here, "this" is the new instance of ElementOfSetOfAllIntegersGreaterThanOrEqualTo5 which is being constructed and e5 is an instance of ElementOfIntegerSet which represents the number 5 in the integer set. However, this paper has not defined any relations including "greater than or equal to" on "Integer Set". So, by now, this judgement is meaningless. And though this paper ignored it in the sections before, in Hypothesis 1, the author mentioned "different instances" but the author has not given a way to judge whether two instances are the same. For preciseness, it is necessary to find a way to simulate the concept "relation" in set theory.

DOI: 10.54254/2753-8818/9/20240729

4.1. Hypothesis 5

In a f1c1class A simulating set SA, any relation which can be defined on SA can be simulated by a method with a return type bool and exactly two parameters whose types are both A. Of course, a customized method is usable, but C# provided users with operator overriding and overloading to realize that conveniently.

4.2. Definition 3

Two instances of f1c1class A are called "different" if and only if A's method operator== [8] returns false when passing these two instances to parameters. Notice that in Example 4 the operator== has not been overridden so it will be incorrect when judging as follows:

new ElementOfIntegerSet(3)==new ElementOfIntegerSet(3)

The statement above will be false because the left side and the right side are seen as two different instances. But in fact, they are simulating the same element in the set of integers.

4.3. Example 9

There is a need to improve the class ElementOfIntegerSet in Example 4. Add codes as follows into the class definition:

```
public static bool operator ==(ElementOfIntegerSet left, ElementOfIntegerSet right)
{
    return left.X == right.X;
}
public static bool operator !=(ElementOfIntegerSet left, ElementOfIntegerSet right)
{
    return !(left.X == right.X);
}
public static bool operator >=(ElementOfIntegerSet left, ElementOfIntegerSet right)
{
    return left.X >= right.X;
}
public static bool operator <=(ElementOfIntegerSet left, ElementOfIntegerSet right)
{
    return left.X <= right.X;
}
```

By adding operator overriding, now the operator == can be used to judge whether two instances of ElementOfIntegerSet are equal, which means they are the same element in the set of integers.

It is easy to prove that the operator== is reflexive (which means if a parameter left and right refers to the same instance, the return value is always true), symmetric (which means the return value never changes when swapping the order of two parameters), transitive (which means for any three instances of this class A, B, C if the return value is true for parameters A and B, and the return value is true for parameters B and C, then the return value must be true for parameters A and C). Also, by adding operator overloading, the operator >= and operator <= are defined and it is easy to prove that they are transitive. There is one last thing that has not been solved, which is that in the 7th line of Example 6, there is an operator "%", which remains undefined.

4.4. Hypothesis 6

In a f1c1class A simulating set SA, any algebra operation which can be defined on A can be simulated by a method with a return type A and exactly two parameters whose types are both A. Still, consider using operation overloading to realize that.

4.5. Example 10

Add the overloading of operation "%" in class ElementOfIntegerSet as follows.

DOI: 10.54254/2753-8818/9/20240729

```
public static ElementOfIntegerSet operator "%" (ElementOfIntegerSet left, ElementOfIntegerSet right)
{
return new(left.X % right.X);
}
```

Now the class is completed and can satisfy all the needs above so the codes in Example 5,6,7,8 become correct. It is provable that for each call of the operator "%", the parameters are two ElementOfIntegerSet instances which simulate two elements from the set of integers, and the return type is also ElementOfIntegerSet so that the return value is also an instance of ElementOfIntegerSet simulating an element from the set of integers, which means that this operator "%" method has closure on the set of all integers. Notice that the operator "%", which is the modular arithmetic, is very important when expanding the conclusions in this paper to further algebra concepts such as groups [10] so that it should be carefully overloaded. The example above is just one of the simplest realizations and may not be precise in future works.

5. Conclusion

In this paper, the author proposed an approach to use classes and their methods to simulate sets, relations and algebra operations, and a way to simulate the inclusion relation by using inheriting of classes. Also, the author proposed that the union and intersection operations can also be simulated by moving the method body of the constructor of classes. Limited by time and the knowledge, there are still many drawbacks in this paper. Precise proofs of the hypotheses are not given, and this paper did not mention mapping, which is also essential to the topic. The author considers that it is valuable for further studies to see whether this theory can be extended to some subjects like abstract algebra.

References

[1] Yang, Z. (2020) Basic concepts. In: Yang, Z. (Eds.) Modern Algebra (4th edition). Higher Education Press Inc., Peking. pp. 2-18.
[2] Cai, Z. (2023) Design and Realization of Automatic Stereoscopic Warehouse Control and Management System Based on C#. Ningxia University.
[3] Yang, F., Xu, H., Chen, S. (2017) About the Application of C# in the Calculation of Background Value, Line Metal Quantity and Reserves. In: The 9th National Congress and 16th Annual Meeting of the Chinese Society of Mineralogical and Petrographic Geochemistry. Xian,China. pp. 1005-1006.
[4] ISO/IEC 14882:2020, Programming languages — C++.
[5] Wu, D. (2016) An Empirical Study of the Key C++ Language Features Based on Open-Source Software. Nanjing University.
[6] Microsoft. (2023) NET API Browser. https://learn.microsoft.com/en-us/dotnet/api/.
[7] Oracle and/or its affiliates. (2023) Java® Platform, Standard Edition & Java Development Kit Version 20 API Specification. https://docs.oracle.com/en/java/javase/20/docs/api/.
[8] Microsoft. (2023) C# reference. https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/.
[9] Microsoft. (2021) Advanced NET programming documentation. https://learn.microsoft.com/en-us/dotnet/navigate/advanced-programming/.
[10] Purwanto. (2020) Construction of multiplicative groups in modular arithmetic. Journal of Physics: Conference Series. In: 1st International Conference on Mathematics and its Applications (ICoMathApp) 2020. Malang, Indonesia. pp: 1872:012009.