재귀 함수에 관한 고찰

사서림

2025.04.13.

[재귀 함수에 관한 고찰]

재귀 함수는 어떤 함수에 대해서 조건이 부합할 때까지만 자기자신, 즉 지금 실행하고 있는 함수를 실행시키는 함수이다. 우리는 이 함수에서 두 가지 요소를 쉽게 발견 할 수 있다. 바로 실행할 함수와 계속 반복 실행할 조건이다.

$$f = \text{실행할 함수}$$
$$g = \text{조건}$$

이렇게 표기하고 재귀 함수를 자세히 들여다보면 우리는 함수적 표현으로 재귀 함수를 R 라고 표현했을 때 다음과 같이 써볼 수도 있다는 생각이 들 것이다.

$$R(f, g)(x)$$

그렇다면 재귀 함수 $R(f, g)(x)$ 의 공역은 무엇일까? 한번 자세히 생각해보자.
일단 우리는 재귀 함수는 조건에 부합할 때까지만 실행 중인 f 에서 f 안에 f 를 넣는다는 걸 알 수 있다. 이때 함수에 넣는 값도 보기위하여 함수의 표현을 써서 $f(x)$ 라고 하면 우리는 다음과 같음을 알 수 있다.

$$\text{조건에 부합하면 } f(x) \text{ 실행 중에 } f(x) \text{ 에 } f \text{ 를 넣는다. 즉}$$
$$g = \text{true} \Rightarrow f(f)$$

즉 f 의 공역은 f 의 정의역과 같아야 함을 알 수 있다. 우리는 이것을 이렇게 쓸 수 있다.

$$f : T \to T$$

이때 우리는 g 또한 함수로 다룰 수 없을까 생각할 수 있다. 그럼 g 의 정의역은 무엇일까? 우리는 보통 조건을 f 의 입력값 x 에 대해서 생각한다. 하지만 조건이 $f(x)$ 에 영향을 받는 특이한 경우도 있으므로 g 를 다음과 같이 정의할 수 있다.

$$g(x, f)$$
$$g : T \times (T \to T) \to \text{bool}$$

그럼 $R(f, g)(x)$ 는 다음과 같이 표기 가능하다.

$$R : T \to T$$
$$R(f, g)(x) = \begin{cases} x & \text{if } g(x, f(x)) = \text{true} \\ R(f, g)(f(x)) & \text{otherwise} \end{cases}$$

하지만 우리 한번 굳이 조건이 끝났을 때 무조건 입력값 x 를 반환해야 하는지에 대해서 생각해보자. 사실 잘 생각해보면 조건이 끝났을 때 어떻게 연산 (동작) 하고 끝낼지도 결정할 수 있다. 이를 h 라고 해보자. h 는 x, f, g 의 영향을 받을 수 있는데 이는 입력값 x 와 두 개의 함수 f, g 가 매개변수로서 동작해야 한다는 말로도 이해할 수 있다. 그러므로 h 를 다음과 같이 정의하자.

$$h(x, f, g)$$

그런대 여기에서 잘 생각해보면 꼭 h 의 공역이 f 의 정의역이나 공역과 같을 필요는 없다. 왜냐하면 $h(x, f, g)$ 는 $f(x)$ 를 매개변수로 받을 뿐 순수하게 독립된 함수이기 때문이다. 그러므로 우리는 R 의 공역을 f 의 정의역이나 공역인 T 와 다르게 다른 집합 U 로 잡을 수도 있다. 그러므로 h 를 정의하고 R 을 다시 정의할 수 있다.

$$f(x) = \text{함수}$$
$$g(x, f(x)) = \text{함수} f(x) \text{의 재귀가 종료될 조건}$$
$$h(x, f(x), g(x, f(x))) = \text{재귀가 끝난 후 연산할 함수 (즉 해석기)}$$

여기에서 각각 함수의 정의는 다음과 같다.

$$f : T \to T$$
$$g : T \times (T \to T) \to \text{bool}$$
$$h : T \times (T \to T) \times (T \times (T \to T) \to \text{bool}) \to U$$

그리고 최종적인 재귀 함수 R 은 다음과 같이 정의한다.

$$R : f \times g \times h \times T \to U$$
$$R : (T \to T) \times (T \times (T \to T) \to \text{bool}) \times (T \times (T \to T) \times (T \times (T \to T) \to \text{bool}) \to U) \times T \to U$$
$$\text{재귀의 핵심 입력과 출력간의 관계만 따지면 } R : T \to U$$
$$R(f, g, h)(x) = \begin{cases} h(x, f, g) & \text{if } g(x, f(x)) = \text{true} \\ R(f, g, h)(f(x)) & \text{otherwise} \end{cases}$$

재귀 함수 $R(f,g,h)(x)$ 는 기존에 통합되어 있던 실제 동작하는 함수와 반복 조건 그리고 결과에 대한 연산부를 독립적인 함수로 분리함으로써 이전보다 편하게 유지관리할 수 있으며 재귀 함수 자체를 면밀히 분석하여 기존보다 더 잘 관리할 수 있는 가능성을 제시한다. 그리고 재귀 함수 R 을 변수로도 볼 수 있을것이라는 생각 또한 적어본다.

-[재귀 함수에 대한 고찰] 끝-

[재귀 함수 R 을 C# 으로 구현 시도해보기]

이번 챕터에서는 C# 를 이용하여 $R(f,g,h)(x)$ 을 구현하는 것을 시도 해보고자 한다. 그럼 바로 클래스를 만들어서 필요한 변수부터 정의하도록 하자.

```
namespace RecursionFuntion
{
    public partial class RecursionFuntion<T,U>
    {
        public Func<T, T> f { get; private set; }
        public Func<T,Func<T,T>, bool> g { get; private set; }
        public Func<T,Func<T,T>, Func<T, Func<T, T>, bool>,U> h { get; private set; }
    }
}
```

Listing 1: 재귀 함수 R 의 기본 매개변수 f,g,h 에 대한 정의부

위의 내용을 보면 RecursionFuntion 클래스를 템플릿 T, U 를 사용하는 클래스로 만들어서 $R : T \to U$ 개념을 구현하려고 하였음을 알 수 있다. 그리고 각 변수 f,g,h 를 정의한 것을 보면 충실하게 아래의 구조를 잘 따르게 만들어져 있음을 알 수 있다.

$f : T \to T$ Func< 매개변수, 출력 > 임으로
Func<T,T> 로 정의됨

$g : T \times (T \to T) \to \text{bool}$ Func< 매개변수 1, 매개변수 2, 출력 > 임으로
Func<T,Func<T,T>, bool> 로 정의됨

$h : T \times (T \to T) \times (T \times (T \to T) \to \text{bool}) \to U$ Func< 매개변수 1, 매개변수 2, 매개변수 3, 출력 > 임으로
Func<T,Func<T,T>, Func<T, Func<T, T>, bool>,U> 로 정의됨

여기에서 보면 f 에서 g, h 로 갈수록 이전 함수에 의존적이라는것을 알 수 있다. 이는 $f(x)$, $g(x,f)$, $h(x,f,g)$ 로서 조건 g 가 f 의 영향을 받고, 해석기 h 는 f 와 g 의 영향을 받을 수도 있기 때문이다.
이제 생성자와 f,g,h 를 할당할 수 있는 메소드를 만들어보자.

```
namespace RecursionFuntion
{
    public partial class RecursionFuntion<T,U>
    {
        public RecursionFuntion() { }
        public RecursionFuntion(
            Func<T, T> f,
            Func<T, Func<T, T>, bool> g,
            Func<T, Func<T, T>, Func<T, Func<T, T>, bool>, U> h)
        {
            this.f = f;
            this.g = g;
            this.h = h;
        }
        public void SetF(Func<T, T> f)
        {
            this.f = f;
        }
        public void SetG(Func<T, Func<T, T>, bool> g)
        {
```

```
            this.g = g;
        }
        public void SetH(Func<T, Func<T, T>, Func<T, Func<T, T>, bool>, U> h)
        {
            this.h = h;
        }
    }
}
```

Listing 2: 재귀 함수 R 의 생성자 및 할당 메소드

위의 내용을 보면 말 그대로 f, g, h 를 할당하는 간단한 생성자와 메소드들이 있다. 앞으로 이것을 어떻게 사용할 수 있는지는 실제로 팩토리얼 재귀 함수를 만들어서 확인해보겠다. 이제 R 재귀 함수의 구동부 메소드를 보자. 쉬운 이해를 위하여 선형적인 재귀 함수일 때를 중점으로 두고 만든 클래스임을 명심하고 보는 것이 중요하다.

```
namespace RecursionFuntion
{
    public partial class RecursionFuntion<T,U>
    {
        public U Run(T x)
        {
            while (!g(x, f))
                x = f(x);
            return h(x, f, g);
        }
    }
}
```

Listing 3: 재귀 함수 R 의 구동 메소드 Run

위의 내용을 보면 너무 간단해서 어이가 없을 수도 있을 것이다. 하지만 위의 내용은 선형적인 재귀 (분기 구조가 없는 재귀) 에서 나올 수 있는 가장 간단한 식이며, 기존의 재귀 함수에서도 선형적인 재귀 함수는 위와 비슷하게 $f(x)$ 의 결과를 x 에 할당하고 다시 f 에 넣음으로서 구동이 가능하다. 이제 재귀 함수 R 을 이용하여 간단한 양수 팩토리얼 재귀 함수를 만들어 보자.

```
namespace RecursionFuntion
{
    internal class Program
    {
        public struct A
        {
            public int control { get; set; }
            public long value { get; set; }
            public A(int control, long value)
            {
                this.control = control;
                this.value = value;
            }

            public static A F(A x)
            {
                A a = new(x.control, x.value);
                a.value = a.value * a.control;
                a.control--;
                return a;
            }
            public static bool G(A x, Func<A,A> F)
            {
                if (x.control == 0 && F(x).value == 0)
                    return true;
                else
                    return false;
            }
            public static long H(A x, Func<A,A> F, Func<A,Func<A,A>,bool> G)
            {
                return x.value;
            }
```

```
        }
        static void Main(string[] args)
        {
            RecursionFuntion<A, long> recursionFuntion = new RecursionFuntion<A, long>(A.F, A.G, A.H);
            Console.WriteLine( recursionFuntion.Run(new(10,1)) );
        }
    }
}
```

Listing 4: 재귀 함수 R 을 실제로 사용하는 예제

위의 내용을 보면 A.F 에서 연산을 하고 G 가 A 와 F 를 받아서 조건을 판단하고 최종적으로 H 가 해석해서 내보내는 구조임을 알 수 있다. 이제 다음으로 Stack 기반 R 을 보러가자
다만 Stack 은 초보여서 구현이 이상하게 되어 있을 수 있음으로 코드 전체와 함수의 정의에 대해서 말씀드리겠다.

```
namespace RecursionFuntion
{
    public class RecursionFuntionStack<T,U>
    {
        public Func<T, IEnumerable<T>> f { get; private set; }
        public Func<T, Func<T, IEnumerable<T>>, bool> g { get; private set; }
        public Func<T, Func<T, IEnumerable<T>>, Func<T, Func<T, IEnumerable<T>>, bool>, U> h { get; private set; }

        public RecursionFuntionStack() { }
        public RecursionFuntionStack(
            Func<T, IEnumerable<T>> f,
            Func<T, Func<T, IEnumerable<T>>, bool> g,
            Func<T, Func<T, IEnumerable<T>>, Func<T, Func<T, IEnumerable<T>>, bool>, U> h)
        {
            this.f = f;
            this.g = g;
            this.h = h;
        }

        public void SetF(Func<T, IEnumerable<T>> f)
        {
            this.f = f;
        }
        public void SetG(Func<T, Func<T, IEnumerable<T>>, bool> g)
        {
            this.g = g;
        }
        public void SetH(Func<T, Func<T, IEnumerable<T>>, Func<T, Func<T, IEnumerable<T>>, bool>, U> h)
        {
            this.h = h;
        }

        public List<U> Run(params T[] StartStates)
        {
            Stack<T> stack = new(StartStates);
            List<U> results = new();

            while (stack.Count > 0)
            {
                var current = stack.Pop();

                if (g(current,f))
                {
                    results.Add(h(current, f, g));
                }
                else
                {
                    foreach (var next in f(current))
                    {
```

```
                        stack.Push(next);
                    }
                }
            }
            return results;
        }

    }
}
```

Listing 5: 재귀 함수 R 의 Stack 버전

여기에서 f, g, h 는 다음과 같이 정의 되어있다.

$\mathcal{T}$ 는 T 집합의 요소인 T_n 들 중에 쓰는 T_n 의 집합 (배열이라고 생각해도 무방함) 이다.
$\mathcal{U}$ 는 U 집합의 요소인 U_n 들 중에 쓰는 U_n 의 집합 (배열이라고 생각해도 무방함) 이다.
$f : T \to \mathcal{T}$
$g : T \times (T \to \mathcal{T}) \to \mathrm{bool}$
$h : T \times (T \to \mathcal{T}) \times (T \times (T \to \mathcal{T}) \to \mathrm{bool}) \to \mathcal{U}$

그러므로 Stack 버전 재귀 함수 R 은 트리를 다룰 때에 유용하게 쓸 수 있을 것으로 추측하고 실험삼아 만든 것이다.

그래서 이 구조는 $R : T- > \mathcal{U}$ 인 것이다. 그리고 h 가 집합 $\mathcal{U}$ 를 내보내지 않고 최종적으로 하나의 U 를 내보내는 경우도 생각해볼 수 있다. 다만 그렇게 되도 $h : T \times f \times g \to \mathcal{U}$ 임으로 $\mathcal{U}$ 를 U 로 축약하는 해석기 $h' : \mathcal{U}- > U$ 를 정의하면 $R : T- > U$ 이면서 비선형 재귀인 구조도 가능할 것이라고 생각한다. 이때 R 은 $R(f, g, h, h')(x)$ 로 다시 정의된다.

$$R(f, g, h, h')(x) = \begin{cases} h'(h(x, f, g)) & \text{if } g(x, f(x)) = \text{true} \\ R(f, g, h, h')(f(x)) & \text{otherwise} \end{cases}$$

하지만 $R : T- > U$ 인 비선형 재귀 함수는 아직 구현하지 않았다.

이상으로 문서를 마치겠다. 읽어주셔서 고맙습니다!

-끝-