귀납법에 관하여

임현승

Draft of September 1, 2018

1 귀납적 정의

귀납적 정의inductive definition란 뭘까? 프로그래밍 언어론 수업에서 왜 귀납법이라는 것을 배워야할까? 귀납법은 집합을 정의하는 중요한 방법이다. 프로그래밍 언어론은 프로그래밍 언어에 대한 이론이다. 우리는 프로그래밍 언어를 이용하여 프로그램을 작성한다. 하나의 프로그래밍 언어를 이용하여 작성할 수 있는 프로그램은 무수히 많다. 하나의 프로그래밍 언어를 이용하여 작성할 수 있는 모든 프로그램들의 집합 P를 생각해보자. 우리는 귀납법을 이용하여 그 집합 P를 "엄밀하게" 정의할 수 있다. 즉, 우리는 프로그래밍 언어로 작성할 수 있는 모든 집합을 표현하기 위해 귀납법을 공부한다.

어... 그런데 이상하다. 집합을 정의하는 방법은 우리가 중고등학교 때 배운 원소나열법, 조건제시법도 있는데 굳이 귀납법을 이용하는 이유는 뭘까? 결론부터 이야기하면 귀납법은 집합을 정의할 뿐 아니라 집합의 원소를 만드는 방법(알고리즘)도 제시해주기 때문이다. 일반적으로 원소나열법과 조건제시법은 우리가 어떤 값이 있을 때 그 값이 집합에 포함될 수 있는지 없는지를 판단하는 기준은 제시하지만 그 집합의 원소를 생성하는 방법을 알려주지는 않는다.

원소나열법 기억이 가물가물한 사람들을 위해서 자연수 집합을 정의해보자. 원소나열법을 이용하면

$$\text{자연수 집합} = \{0, 1, 2, 3, 4, ...\}$$

이 될 것이다.[1] 우리는 무수히 많은, 즉 무한한 수를 나열할 수 없다. 오직 유한한 수만을 나열할 수 있다. 따라서 원소나열법을 이용하면 어쩔 수 없이 찝찝하지만 무한히 많은 원소를 가진 집합을 표현하기 위해 ...을 사용할 수 밖에 없다. 만약 C나 OCaml로 작성 가능한 모든 프로그램들의 집합을 고려한다면, 그리고 그 집합의 원소(프로그램)의 일반적인 성질에 대해서 이야기 하고 싶다면?

조건제시법 집합을 정의하는 두 번째 방법은 조건제시법을 이용하는 것이다. 집합의 원소가 만족해야 되는 조건을 제시하는 것이다. 예를 들어,

$$\text{자연수 집합} = \{n \mid n \geq 0\}$$

를 고려해보자. 위 식은 엄밀히 말하면 자연수 집합을 정의하지 않는다. 0과 양의 실수들을 모두 포함하는 집합일까? 그렇다면 자연수 집합을 표현하기 위해서는 어떤 조건을 추가해야할까? 아래와 같이 하면 될까?

$$\text{자연수 집합} = \{n \mid n \geq 0,\ n\text{은 자연수}\}$$

위 정의는 뭔가 이상하다. 자연수 집합을 정의하기도 전에 n은 자연수라는 조건을 사용하다니... 결론부터 이야기하면 조건제시법을 이용해도 자연수 집합을 정의하기는 쉽지 않아 보인다... OTL

1.1 집합의 귀납적 정의

자, 이제 자연수 집합을 귀납법을 이용하여 정의해보자. 그런데 당췌 귀납법이 무슨 말인가? 생소하다. 한자를 풀어보면 귀(歸)는 "돌아가다", "돌아오다" 정도의 뜻이고 납(納)은 "거두어 들이다", "바치다" 정도의 뜻인데 그럼 귀납은 "돌아서 거두어 들이다", "돌아서 바치다" 정도의 뜻이 될 수 있겠다. 역시 무슨 말인지 잘 모르겠다. 귀는 귀로, 귀경길과 같은 단어에서 사용하는 한자말이고 납은 헌납, 수납과 같은 단어에서 사용하는 한자말이다. 그래도 귀납을 해석하기는 어렵다. 아마도 induction을 귀납법으로 번역하면 다소 어려운 것 같다.

[1] 0을 자연수 집합에 포함시키는 것은 여러모로 장점이 있으며 현대 수학 및 컴퓨터과학 분야에서는 주로 0을 자연수 집합의 기초 원소로 다룬다.

자, 더 쉽게 설명해보자. 귀납적 정의는 집합을 정의하는 방법이라고 했다. 그리고 집합의 원소를 만드는 방법을 알려주는 알고리즘이라고도 했다. 귀납적 정의는 집합의 원소를 그 집합의 또 다른 원소를 이용하여 정의하는 방법이다.[2] 예를 들어 자연수 집합을 귀납적 정의를 이용하여 정의해보자. 자연수 집합을 nat이라고 하자. 우리는 nat을 다음 두 가지 자연수 생성 규칙을 이용하여 정의할 수 있다.

1. $\mathsf{O} \in \mathsf{nat}$ base case (1단계)
2. $\mathsf{S}\ n \in \mathsf{nat}$ if $n \in \mathsf{nat}$ inductive case (귀납 단계)

우리는 일단 O은 자연수 집합 nat에 포함된다는 것을 안다. 아무런 선행지식도 필요없는 당연한 사실이자 정의이다 (또는 공리). 이를 1단계(base case)라 하자.

다음은 귀납 단계(inductive case)이다. 자연수는 무한한데 1단계에서는 유한한 경우 밖에 다루지 못 한다. 따라서 무한한 원소를 표현할 수 있는 방법이 필요하다. 이게 바로 귀납 단계이다. 귀납 단계는 우리가 이미 알고 있는 사실을 이용하여 새로운 사실을 만드는 단계이다. 즉, 이미 알고 있는 자연수 집합의 원소를 이용하여 새로운 자연수를 만드는 것이다. 달리말해, 자연수 집합 nat은 다음과 같은 성질을 만족한다고 정의하는 것이다:

n이 nat의 원소라면 S n도 nat의 원소.

여기에서 S n은 "successor of n"이라는 의미이며, 이는 n 다음 수, 즉 $n+1$을 의미한다.

귀납적 정의를 이용하여 표현한 자연수 집합이 우리가 원하는 자연수 집합임을 우리는 직관으로 알 수 있다. 또한 귀납적 정의를 이용하여 자연수를 생성하는 방법(알고리즘)도 안다. O부터 시작하여 계속 S를 앞에 붙여나가면 우리가 원하는 자연수를 만들 수 있다.

귀납적 정의를 좀 더 이론적으로 표현하면 아래와 같다:

$$\mathsf{nat} \quad n ::= \mathsf{O} \mid \mathsf{S}\ n'$$

nat은 syntactic category라고 말하는데 쉽게 말하면 집합이다. 단지 syntax(문법, 생김새)가 중요한 역할을 하는 집합이라는 의미이다. 무슨 말이냐고? 즉 집합의 원소가 생김새가 다르면 다르고 생김새가 같으면 같은 원소인 집합을 의미한다. 예를 들어, O과 S O은 서로 생긴게 다르니까 다른 원소가 된다.[3]

메타 변수 위의 정의에서 n은 메타 변수(metavariable)이라고 불린다. 변수는 변수인데 앞에 메타가 붙었다. 메타 변수는 임의의 집합 원소를 가리키는 변수 정도로 해석하면 되겠다. 여기에서 임의의 원소를 가리킨다는 게 중요하다. 메타 변수의 이름은 중요하지 않다. 우리가 C 코딩을 하거나 OCaml 코딩을 할 때 변수의 이름이 프로그램의 의미를 결정하는 데 중요하지 않은 것과 같다. 예를 들어,

```
let rec sum n = if n = 0 then 0 else n + sum (n - 1)
```

과

```
let rec sum x = if x = 0 then 0 else x + sum (x - 1)
```

는 본질적으로 같은 OCaml 함수 `sum`을 정의한다.

::= 는 "is defined as"의 수학적 표현식이다. 즉, $n ::= \ldots$ 은 집합의 원소 n은 ... 과 같이 정의된다 정도로 이해하면 되겠다. 다음으로 | 는 "or"의 수학적 표현이다. 따라서 위 귀납적 정의는 "자연수 집합 nat의 원소 n은 O이거나 또는 S n'으로 정의된다"는 의미이다. 이 때, n'은 또 다른 자연수 집합의 원소를 의미한다. 오해의 소지가 없다면 위 정의를 아래와 같이 정의할 수도 있다.

$$\mathsf{nat} \quad n ::= \mathsf{O} \mid \mathsf{S}\ n$$

여기서는 S n'을 쓰지 않고 S n을 썼다. 메타 변수 n은 "어떤 n" 정도로 해석할 수 있으므로 위 정의는 올바르다. 즉, "자연수 집합 nat의 어떤 원소 n은 O이거나 또는 또 다른 어떤 원소 n이 있어서 S n으로 정의된다" 정도로 생각하면 되겠다. 즉 같은 메타 변수 n을 여러번 사용하더라고 메타 변수는 사용할 때마다 서로 다른 임의의 원소를 가르킨다.[4]

Exercise 1.1. *이진수는 0 과 1로만 이루어진 수이다. 1로 시작하지 않는 양의 이진수 집합*[*syntactic category*] bin*을 귀납법을 이용하여 정의하라. 예를 들어, 01은 bin의 원소지만 10은 아니다.*

$$\mathsf{bin} \quad b ::= ???$$

[2] 그래서 귀납적 정의는 자기호출함수(recursive function)와 닮았다.

[3] 또 다른 집합 정의에서는 생김새가 달라도 의미가 같으면 같은 원소라고 할 수도 있다. 우리가 공부하는 syntactic category(집합)에서는 생김새가 다르면 다른 원소로 본다.

[4] 경우에 따라서는 같은 원소를 가리킬 수도 한다.