1 장

기본기

프로그래밍 언어를 이야기 하는데 기본적으로 사용하는 어휘들과 말하는 방식이 있다. 그 어휘와 방식의 기본은 수학이다. 이 장에서는 앞으로 우리가 사용할 어휘와 말하는 방식을 알아보고, 그 사용 예들을 살펴보자. 워밍업이 되겠다.

1.1 귀납법

1.1.1 집합의 정의

집합을 정의하는 방법에는 조건제시법과 원소나열법, 그리고 하나가 더 있다. 귀납법이다. 귀납법은 대게 증명의 한 방법이라고 배우지만, 사실 그 증명이 수긍이 가는 이유는 집합을 정의하는 귀납법의 얼개를 따라서 증명해 나가는 것이기 때문이다. 이에 대해서는 다음장(1.1.2)에서 살펴보기로 하고 귀납이 집합을 정의하는 데 사용되는 것에 대해서 우선 살펴보자. 귀납법이 즐거운 이유는, 유한한 갯수의 유한한 규칙들로 무한히 많은 원소를 가지는 집합을 정의할 수 있기 때문이다.

귀납법으로 집합을 정의하는 방법은, 그 이름이 의미하는데로, 되돌아서 바치는 것이다(되돌아서 return 歸, 바치다 dedicate 納). 되돌아서 만들어 바친다는 것은, 그 집합의 원소를 가지고 그 집합의 원소를 만든다는 것이다. 귀납법

으로 정의되는 집합은 몇개의 규칙들로 구성되고, 그 규칙들이 귀납적이다: 무엇무엇이 이미 집합의 원소들이라면 그것들로 어떻게어떻게 하면 다시 집합의 원소가 된다, 는 식이다.

1.1.1.1 그 집합은 이것이다

조금 일반적인 톤으로 이야기해보면 이렇다. 하나의 규칙은 짝으로 구성된다: 가정들 X와 결론 x. 그 규칙이 말하는 바는 "X에 있는 것들이 정의하려는 집합에 모두 있으면, x도 있어야 한다" 는 것이다. 이러한 규칙들이 모여서 하나의 집합을 정의한다. 그 집합은, 규칙들이 요구하는 원소들은 모조리 포함하고 있는 집합 가운데 가장 작은 집합으로 한다. 최소의 집합이라고 고집하는 이유는, 규칙들이 요구하는 원소들은 모두 포함하지만 그 밖의 것들은 제외하고 싶어서이다.

좀 더 명확하게 가보자. 가정들 X와 결론 x의 짝 (X, x)로 표현되는 규칙들의 모임을 Φ라고 하자.

그 규칙을 만족하는 원소들을 모두 품고 있는 집합을 "Φ에 대해서 닫혀있다" 혹은, "Φ-닫힘"이라고 한다. 즉, 집합 A가 Φ-닫힘, 은

$$(X, x) \in \Phi \text{ 이고 } X \subseteq A \text{ 이면 } x \in A$$

인 경우를 말한다. 이제, Φ가 정의하는 집합은 모든 Φ-닫힌 집합들의 교집합으로 정의된다:

$$\bigcap \{A | A\text{는 } \Phi\text{-닫힘}\}.$$

그러한 집합은 Φ-닫힌 집합이면서 가장 작다. (왜?)

Example 1 자연수의 집합은 다음 두개의 규칙들로 정의된다:

$$(\emptyset, 0) \quad (\{n\}, n+1)$$

즉, 0은 자연수이고, n이 자연수라면 $n+1$도 자연수이다. 이 규칙들에 대해서

닫혀있는 집합들은 우리가 알고 있는 자연수의 집합 $\mathbb{N}$ 뿐 아니라 유리수의 집합 $\mathbb{Q}$도 있다. 하지만 가장 작은 그러한 집합은 $\mathbb{N}$ 이다.□

Example 2 영문 소문자 알파벳으로 만들어 지는 스트링의 집합은 다음의 규칙들로 정의된다:

$$(\emptyset, \epsilon) \quad (\{\alpha\}, x\alpha \text{ for } x \in \{\mathsf{a}, \cdots, \mathsf{z}\})$$

즉, ϵ은 스트링이고, s가 스트링 이라면 그 앞에 영문 소문자를 하나 붙여도 스트링이다. 이 규칙들에 대해서 닫혀있는 가장 작은 집합이 이 규칙들이 정의하는 "스트링"이라는 집합이다. □

귀납법의 규칙을 나타내는 편리한 표기로 프로그래밍 언어에서는 다음 두 가지 것을 주로 쓴다. 위의 자연수 집합은:

$$n \ \rightarrow \ 0 \mid n+1$$

혹은

$$\frac{}{0} \qquad \frac{n}{n+1}$$

이고, 위의 스트링 집합은:

$$\begin{array}{rcl} \alpha & \rightarrow & \epsilon \\ & \mid & x\alpha \quad (x \in \{\mathsf{a}, \cdots, \mathsf{z}\}) \end{array}$$

혹은

$$\frac{}{\epsilon} \qquad \frac{\alpha}{x\alpha} \ x \in \{\mathsf{a}, \cdots, \mathsf{z}\}$$

으로 표현한다.

그리고, 귀납 규칙들은 반드시 되돌아서 표현될 필요는 없다. 집합이 유한하다면 재귀가 필요없이 원소나열법과 같이 주욱 쓰면 된다. 예를 들어, 집합 $\{1, 2, 3\}$을 규칙들로 표현하면

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

혹은

$$x \to 1 \mid 2 \mid 3$$

혹은

$$\frac{}{1} \quad \frac{}{2} \quad \frac{}{3}$$

가 된다.

Example 3 리스트의 집합도 다음 두개의 규칙들로 정의된다:

$$\frac{}{\texttt{nil}} \qquad \frac{\ell}{\circ\!-\!\ell}$$

혹은

$$\ell \;\to\; \texttt{nil} \mid \circ\!-\!\ell$$

즉, `nil`은 리스트이고, ℓ이 리스트라면 그 앞에 하나의 노드를 붙인 $\circ\!-\!\ell$도 리스트이다. 이 규칙들에 대해서 닫혀있는 가장 작은 집합이 이 규칙들이 정의하는 "리스트"라는 집합이다. □

Example 4 말단에 정수를 가지는 두갈래 나무(*binary tree*)들의 집합은 다음 네개의 규칙들로 정의된다:

$$\frac{}{n}\ n \in \mathbb{Z} \qquad \frac{t}{\texttt{N}(t, \texttt{nil})}$$

$$\frac{t}{\texttt{N}(\texttt{nil}, t)} \qquad \frac{t_1 \quad t_2}{\texttt{N}(t_1, t_2)}$$

혹은

$$\begin{aligned} t \;&\to\; n \quad (n \in \mathbb{Z}) \\ &\mid\; \texttt{N}(t, \texttt{nil}) \\ &\mid\; \texttt{N}(\texttt{nil}, t) \\ &\mid\; \texttt{N}(t, t) \end{aligned}$$

즉, 임의의 정수 n은 두갈래 나무이고, t가 두갈래 나무라면 그것을 왼편이나

오른편에 매단 것도 두갈래 나무이고, 두개의 두갈래 나무를 각각 왼편과 오른편에 매단 것도 두갈래 나무이다. 이 규칙들에 대해서 닫혀있는 가장 작은 집합이 이 규칙들이 정의하는 "두갈래 나무"라는 집합이다. □

Example 5 정수식들의 집합은 다음의 규칙들로 정의된다:

$$\frac{}{n}\ n \in \mathbb{N} \qquad \frac{e}{-e}$$

$$\frac{e_1 \quad e_2}{e_1 + e_2} \qquad \frac{e_1 \quad e_2}{e_1 * e_2}$$

혹은

$$\begin{array}{rcl} e & \rightarrow & n \quad (n \in \mathbb{N}) \\ & | & -e \\ & | & e + e \\ & | & e * e \end{array}$$

즉, 자연수 n은 정수식이고, e가 정수식 이라면 그 앞에 음의 부호를 붙여도 정수식이고, 두개의 정수식 사이에 덧셈 부호나 곱셈 부호를 끼워넣어도 정수식이다. 이 규칙들에 대해서 닫혀있는 가장 작은 집합이 이 규칙들이 정의하는 "정수식"이라는 집합이다. □

Example 6 다음의 규칙들이 만들어내는 $\langle A, a\rangle$쌍들의 집합은 무엇일까?

$$\frac{}{\langle A, a\rangle}\ a \in A$$

$$\frac{\langle A, a\rangle \quad \langle A, b\rangle}{\langle A, a \cdot b\rangle} \qquad \frac{\langle A, a \cdot b\rangle}{\langle A, a\rangle}$$

$$\frac{\langle A \cup \{a\}, b\rangle}{\langle A, a^b\rangle} \qquad \frac{\langle A, a^b\rangle \quad \langle A, a\rangle}{\langle A, b\rangle}$$

□

1.1.1.2 그 집합은 이렇게 만든다

그런데, 위의 정의는 명확하지만 한가지가 의아하다. 규칙들이 정의하는 집합이 무엇인지 정의하고 있지만, 그 집합을 만드는 방법을 알려주고 있지 않다. (The definition is non-constructive.) 짜장면은 무엇이다라고 정의해 놓았지만, 짜장면을 만드는 방법은 가르치고 있지 않다! 만드는 방법을 알려주는 정의는 다음과 같다.

규칙들의 집합 Φ는 함수 ϕ를 정의한다. ϕ는 집합을 받아서 그 집합을 가지고 Φ 규칙들을 이용해서 만들어 지는 모든 원소들을 내 놓는다:

$$\phi(Y) = \{x \mid \frac{X}{x} \in \Phi,\ X \subseteq Y\}$$

Φ 규칙들이 정의하는 집합은 함수 ϕ에 의해서 닫혀있는 집합(ϕ가 더 이상 새로운 것을 만들어 내지 못하는 집합, 즉 $\phi(X) \subseteq X$인 집합 X)중에서 최소의 집합

$$\bigcap\{X|\phi(X) \subseteq X\} \tag{1.1}$$

이고, 이것은 함수 ϕ의 최소의 고정점(*least fixed point*)가 된다. (왜?)

함수 ϕ의 최소의 고정점인 이 집합은 빈 집합에서 부터 출발해서(ϕ^0라고 하자) ϕ를 한번 적용하고 나온 집합 ϕ^1, 두번 적용하고 나온 집합 ϕ^2등을 모두 모으면 만들어 진다. 즉,

$$\begin{aligned} \phi^0 &= \emptyset \\ \phi^n &= \phi(\phi^{n-1}) \quad n \in \mathbb{N} \end{aligned}$$

라고하고

$$\phi^0 \cup \phi^1 \cup \phi^2 \cup \cdots = \bigcup_{i \in \mathbb{N}} \phi^i$$

런 식으로 만들어 지는 것이 Φ 규칙들이 정의하는 집합이 되는 것이다. 이렇게 만들어진 집합은 식 1.1과 일치한다. (왜?)

Example 7 자연수의 집합은 다음 두개의 규칙들로 정의된다:

$$n \ \rightarrow \ 0 \mid n+1$$

이 규칙들이 지정하는 집합 $\mathbb{N}$ 은 다음과 같이 만들어 진다:

$$\begin{aligned} \phi^0 &= \emptyset \\ \phi^1 &= \{0\} \\ \phi^2 &= \{0,1\} \\ \phi^3 &= \{0,1,2\} \\ &\cdots \end{aligned}$$

들의 합집합.□

Example 8 리스트의 집합도 다음 두개의 규칙들로 정의된다:

$$\ell \ \rightarrow \ \texttt{nil} \mid \circ\!\!-\!\ell$$

이 규칙들에 정의하는 "리스트"라는 집합은 다음과 같이 만들어 진다:

$$\begin{aligned} \phi^0 &= \emptyset \\ \phi^1 &= \{\texttt{nil}\} \\ \phi^2 &= \{\circ\!-\!\texttt{nil}\} \\ \phi^3 &= \{\circ\!-\!\texttt{nil}, \circ\!-\!\circ\!-\!\texttt{nil}\} \\ &\cdots \end{aligned}$$

들의 합집합.□

Example 9 두갈래 나무(*binary tree*)의 집합은 다음 네개의 규칙들로 정의된

다:

$$\begin{array}{rcl} t & \rightarrow & \circ \\ & | & \mathtt{N}(t, \mathtt{nil}) \\ & | & \mathtt{N}(\mathtt{nil}, t) \\ & | & \mathtt{N}(t, t) \end{array}$$

이 규칙들이 정의하는 "두갈래 나무"라는 집합은 다음과 같이 만들어 진다:

$$\begin{array}{rcl} \phi^0 & = & \emptyset \\ \phi^1 & = & \{\circ\} \\ \phi^2 & = & \{\circ, \mathtt{N}(\circ, \mathtt{nil}), \mathtt{N}(\mathtt{nil}, \circ), \mathtt{N}(\circ, \circ)\} \\ \cdots & & \end{array}$$

들의 합집합.□

Example 10 영문 소문자 알파벳으로 만들어 지는 스트링의 집합은 다음의 규칙들로 정의된다:

$$\begin{array}{rcl} \alpha & \rightarrow & \epsilon \\ & | & x\alpha \quad (x \in \{\mathtt{a}, \cdots, \mathtt{z}\}) \end{array}$$

이 규칙들이 정의하는 집합은

$$\begin{array}{rcl} \phi^0 & = & \emptyset \\ \phi^1 & = & \{\epsilon\} \\ \phi^2 & = & \{\epsilon, \mathtt{a}\epsilon, \cdots, \mathtt{z}\epsilon\} \\ \cdots & & \end{array}$$

들의 합집합이다. □

Example 11 귀납규칙 Φ에 있는 모든 규칙 $\frac{X}{x}$가 X를 공집합으로 가진 게 없다면, Φ가 정의하는 집합은 공집합이 된다. 만들어 지는 시작이 제공되지 않았기 때문이다: $\phi^0 = \phi^1 = \cdots = \emptyset$.□

1.1.1.3 원소들의 순서

Φ 규칙들로 귀납적으로 만들어지는 집합

$$\bigcup_{i\in\mathbb{N}} \phi^i$$

의 모든 원소들은 ϕ를 i번 적용해서 만들어 진다. 원소들의 순서를 정의하기를, ϕ^i번째에 새롭게 만들어 지는 원소들의 순서를 i라고 하자. 원소들 마다 자기가 몇째번에 만들어 진 것이냐에 따라 순서가 있는 것이다.

i째 번에 만들어지는 원소들 x는 $i-1$째 번에 만들어졌던 원소들 X 덕택에 귀납 규칙 (X,x)에 의해서 만들어 진 것이다. $i-1$째 번의 원소들은 또 $i-2$째 번의 원소들 덕택에 만들어 진 것이고. 이 연쇄반응은 결국에는 바닥에 닿는다: 궁극에는 항상 0째 번에 만들어진 원소들이 씨가 된다.

이렇게 언젠가는 바닥을 드러내는 순서를 "기초가 튼튼한 순서(*well-founded order*)"라고 한다. 귀납법으로 정의한 집합은 항상 기초가 튼튼한 순서를 가지고 있다. 이 성질이 귀납법 증명의 기술을 가능하게 해 준다.(장 1.1.2)

1.1.1.4 우리가 다루는 귀납규칙

규칙들이 만드는 집합의 원소들은 규칙들을 유한번 적용해서 만들어지는 원소들로 생각한다. 또, 애매하지 않는 규칙들만 사용한다. 일반적으로, 한 집합을 정의하는 귀납 규칙들은 무한히 많을 수 도 있고, 애매한 경우도 있을 수 있다(두개의 다른 규칙 (X,x)와 (X',x)이 같은 원소 x에 대해서 말하는 경우).

▶

1.1.1.5 무한히 큰 원소도 포함시키기

귀납 규칙들 Φ에 의해서 만들어 지는 원소들은 항상 ϕ를 유한번 적용해서 만

들어 지는 원소일 필요는 없다. 다시 말해서, 만들어지는 집합

$$\bigcup_{i\in\mathbb{N}} \phi^i$$

에서 인덱스 i가 자연수들이라고 했지만, 완전히 일반화 시켜서 i가 유한째 번 이상, 무한째 번 것들(*transfinite ordinals*) 까지로도 확장해서 정의 할 수 있다[Bar77]. 즉, i는 임의의 번째가(*oridinal*) 될 수 있다. 이렇게 확장하려면, 집합론에서 무한째 번으로 레벨을 높이는 방법을 사용하면 된다.

귀납 규칙 Φ가 정의하는 집합은 이제 그 함수 ϕ를 유한째 번 적용해서 생긴 것들 뿐 아니라 임의의 무한째 번을 적용해서 생긴 것들을 모두 포함한다. 이런 모든 "째 번"들을 포함하는 집합(the set of ordinals)을 $\mathcal{O}$라고 하면, Φ가 정의하는 집합은 이제

$$\bigcup_{i\in\mathcal{O}} \phi^i$$

인데, 이 때 ϕ^i는

$$\begin{aligned} \phi^0 &= \emptyset \\ \phi^i &= \phi^{<i} \cup \phi(\phi^{<i}), \quad i \in \mathcal{O} \end{aligned}$$

이며, 여기서

$$\phi^{<i} = \bigcup_{k<i} \phi^k, \quad i \in \mathcal{O}$$

로 정의한다. (이렇게 정의하면 i번 째 만들어 지는 원소는 단순히 $i-1$번째 만들어졌던 원소를 가지고 만들어 지지 않고 $i-1$이하의 모든 번째 원소들을 가지고 만들어 진다.)

이렇게 무한한 원소들이 포함되어 있는 집합인 경우에도, 그 원소들의 순서는 예전 방식대로 존재하고(0째번, 1째번, $\cdots$, 무한째번, 무한+1째번, $\cdots$) 항상 0째번 순서가 그 바닥이 된다(*well-founded order*). 이러한 집합에 대한 증명도 귀납법을 그대로 적용할 수 있고, 이런 경우까지를 포함해서 일반적으로 귀납법 증명을 "유한을 넘어서는 귀납법 증명"(*transfinite induction*)이라고도 한다.

◀

1.1.2 증명의 방법

집합 S의 모든 원소들이 어떤 성질 P를 만족하는 지를 증명하려고 한다고 하자. 즉,

$$\forall x \in S.P(x)$$

를 증명하려고 한다. $P(x)$는 x가 P라는 성질을 만족한다는 뜻이다. 그리고 그 집합 S가 귀납적인 규칙들 Φ에 의해서 정의되었다고 하자.

따라서, S의 원소들 사이에는 순서가 있고(0째번 원소, 1째번 원소, $\cdots$), 그 순서는 항상 0째번에 기초하고 있다. i째번 원소들이란, Φ 규칙들을 i번 적용해서 새롭게 만들어 지는 원소들이다.

집합 S는 모든 i째번 원소들을 모두 모은 집합이므로, S의 모든 원소에 대한 증명은 모든 i째 번에 만들어 지는 원소들에 대한 증명을 하면 된다. 0째번에 만들어 지는 원소는 없다. 따라서 증명할 것도 없다.

- 시작은, 1째번에 만들어 지는 원소들에 대해서 P가 사실임을 증명해야 한다.
- 그리고, 임의의 i 이전 번에 만들어진 원소들에 대해서 P가 사실이라는 가정하에(귀납가정(*induction hypothesis*)라고 함), i째번에 만들어지는 원소들이 P를 만족하는 지를 증명하자.

그러면 집합 S 의 모든 원소들에 대해서 P를 만족하는 지를 증명한 것이 된다. 왜냐면, 집합 S는 모든 i번째 만들어지는 원소들만으로 구성되므로.

1.1.2.1 일반적으로

두개로 쪼개지는 위와같은 증명을 하나로 이야기 하면, 귀납법 증명은 다음을

증명하면 된다: 모든 $i \in \mathcal{O}$에 대해서,

$$(\forall j < i.P(j\text{째번 원소})) \Rightarrow P(i\text{째번 원소}).$$

한꺼번에 두개가 포섭되는 이유를 따져 보면,

- i가 0일 때, "$j < i$"를 만족하는 j도 없고 0째번 원소도 없으므로($\phi^0 = \emptyset$) 위의 식은 그냥 성립한다. 공집합의 원소에 대해서는 모든 성질이 성립하므로.

  i가 1일 때, "$j < i$"를 만족하는 j는 0이고 0째번 원소는 없으므로 위의 식은 P(1째번 원소)과 같다. 왜냐하면, 공집합의 모든 원소에 대해서는 모든 성질이 성립하므로, 위의 식은 $\text{true} \Rightarrow P$(1째번 원소) 즉 P(1째번 원소)가 된다.

- i가 2 이상인 임의의 "째번"에 대해서는 특별한 것 없이, 위의 식 그대로를 증명하면, 귀납가정을 안고($\forall j < i.P(j\text{째번 원소})$) 증명하는 것이 된다.

1.1.2.2 귀납규칙들에 대한 것으로

이 증명 기술을 귀납규칙들에 대한 것으로 다시 이야기 하면,

- 우선, 첫째번에 만들어 지는 원소들은 Φ에 있는 규칙들 중에는 공집합을 전제로 가지고있는 규칙들 $(\emptyset, x)$가 만드는 x들이다. 고로, 그러한 x들에 대해서 증명하는 것이 첫째번의 원소들에 대한 증명이 된다.

- 그 다음, 그 이상 째번에 만들어 지는 원소들은 Φ에 있는 규칙들 중에서 공집합이 아닌 전제($X \neq \emptyset$)를 가지고 있는 규칙들 (X, x)이 만드는 x들이다. x가 i째번에 만들어 지는 원소라면 X에 있는 원소들은 그 이전 번에 만들어 진 원소들이다. 따라서 X에 나타나는 원소들에 대해서 사실이라는 가정 하에 x에 대해서 사실인지를 증명한다.

이렇게 위의 두가지를 증명하면, 모든 i째 번에 만들어지는 원소들(Φ가 정의하는 모든 원소들)에 대해서 증명한 것이 된다.

Example 12 모든 자연수 n에 대해서, $0+1+2+\cdots+n = n(n+1)/2$인지를 증명하자.

모든 자연수는 귀납규칙 $n \to 0 \,|\, n+1$를 유한번 적용해서 만들어지는 원소들의 모임이다.

첫째번에 만들어 지는 원소는 0이다. n이 0일 때 위의 등호가 성립하는 지 증명한다. 그 다음은 i 이전 번에 만들어지는 자연수에 대해서 위의 등호가 성립한다고 가정하고, i째 번에 만들어 지는 자연수에 대해서 사실인지 증명한다. i째 번에 만들어지는 자연수를 n+1이라고 하자. 그 이전 번에 만들어진 자연수들 중에는 n이 있다. 그러면, $0+\cdots+n+(n+1) = n(n+1)/2+(n+1) = (n+1)(n+2)/2$ 이므로 위의 등식이 성립한다.

귀납규칙들을 따라 증명하면, 0일 때 사실인지 증명하고, n일 때 사실이라는 가정하에 $n+1$일 때 사실인지 증명하면 된다. □

Example 13 두갈래가 꽉찬 나무(*complete binary tree*)에 대해서, 말단 노드의 갯수는 내부 노드의 갯수와 같거나 하나 많다는 것을 증명하자. 두갈래가 꽉찬 나무 t를 만드는 귀납규칙은

$$t \to \circ \,|\, \mathtt{N}(t,t)$$

이다. 임의의 t에서 $\circ$(말단 노드)의 갯수 l이 $\mathtt{N}$(내부 노드)의 갯수 n 더하기 1이다.

$\circ$인 경우는, 그렇다. t_1과 t_2의 성질이 그렇다고 하자: $l_1 = n_1 + 1$ 이고 $l_2 = n_2 + 1$. 그러면, $\mathtt{N}(t_1, t_2)$의 말단 노드의 수는 $l_1 + l_2$이고, 내부 노드의 수는 $n_1 + n_2 + 1$이다. $l_1 + l_2 = (n_1 + n_2 + 1) + 1$이 성립한다. □

1.2 형식 논리

형식 논리는 프로그래밍 언어이다. 그 언어로 짜여진 프로그램은 참이거나 거

짓을 계산한다. 형식 논리에 대한 이야기는 프로그래밍 언어에 대해서 이야기 할 때 만나게 될 여러 개념들을 익숙하게 해 준다.

선언논리(*propositional logic*)를 중심으로 알아보자.

1.2.1 모양과 뜻

선언논리에서 생각하는 논리식 f는 다음의 귀납법칙으로 만들어 지는 집합의 원소이다. 간단히, 논리식 f는 다음의 방법으로 만들어 진다.

$$\begin{array}{rcl} f & \rightarrow & T \mid F \\ & \mid & \neg f \\ & \mid & f \wedge f \\ & \mid & f \vee f \\ & \mid & f \Rightarrow f \end{array}$$

위의 귀납법칙이 정의하는 집합의 원소들이 선언논리라는 언어로 짜여진 프로그램들이다. 선언논리식은 위의 방법대로 만들어 지는 것만이 제대로 생긴 선언논리식인 것이다. 무한히 많은 선언논리식들은 모두 위의 일곱가지 방법을 반복 적용해서 만들어 진다.

그럼, 선언논리식의 의미는 무엇인가? 무한히 많은 선언논리식들 하나하나마다 그 뜻이 무었인지 유한하게 정의할 방법은 있는가? 이 경우에도 귀납법을 이용해서 의미하는 바를 정의할 수 있다. 임의의 논리식은 i째 번에 만들어 진다. i째 번에 만들어 지는 논리식은 그 이전 번에 만들어진 논리식을 가지고 만들어 진다. 따라서, i째 번 이전에 만들어진 논리식의 의미를 가지고, i째 번에 만들어 진 논리식의 의미를 정의하는 방법이면 되겠다.(다른 더 좋은 방법이 있을까?)

논리식 f의 의미를 $[\![f]\!]$라고 하자.

- 첫째 번에 만들어 지는 논리식의 의미를 정의하자. T는 참을 뜻하고 F는

거짓을 뜻한다.

$$\begin{aligned}[\![T]\!] &= \text{true} \\ [\![F]\!] &= \text{false}\end{aligned}$$

- i째번 논리식 $\neg f$의 의미는 그 이전에 만들어진 f의 의미를 가지고 정의된다. 그 의미의 부정이 된다.

$$[\![\neg f]\!] = \text{not}[\![f]\!]$$

- 마찬가지로, 이전에 만들어진 f_1과 f_2의 의미를 가지고, 다른 식들의 의미가 결정된다. $f_1 \wedge f_2$의 의미는 그 둘의 논리곱이고, $f_1 \vee f_2$의 의미는 그 둘의 논리합이고, $f_1 \Rightarrow f_2$의 의미는 그 둘의 논리승이다.

$$\begin{aligned}[\![f_1 \wedge f_2]\!] &= [\![f_1]\!] \text{ andalso } [\![f_2]\!] \\ [\![f_1 \vee f_2]\!] &= [\![f_1]\!] \text{ orelse } [\![f_2]\!] \\ [\![f_1 \Rightarrow f_2]\!] &= [\![f_1]\!] \text{ implies } [\![f_2]\!]\end{aligned}$$

이렇게 어느 논리식의 의미가 그 논리식을 구성하는 하부 논리식의 의미로 정의되면 임의의 논리식의 의미가 정의되는 셈이다. 임의의 논리식은 기초가 되는(1째번) 논리식에서 부터 차곡 차곡 만들어진 것이기 때문이다.

Example 14 논리식 $(T \wedge (T \vee F)) \Rightarrow F$의 의미는, 정의에 따라 차곡 차곡 써보면

$$\begin{aligned}[\![(T \wedge (T \vee F)) \Rightarrow F]\!] &= [\![T \wedge (T \vee F)]\!] \text{ implies } [\![F]\!] \\ &= ([\![T]\!] \text{ andalso } [\![T \vee F]\!]) \text{ implies false} \\ &= (\text{true andalso } ([\![T]\!] \text{ orelse } [\![F]\!])) \text{ implies false} \\ &= (\text{true andalso } (\text{true orelse false})) \text{ implies false} \\ &= \text{false}\end{aligned}$$

이다. □

1.2.2 추론규칙

선언 논리식의 의미가 정의되었다. 그 의미는 참이거나 거짓이다. 참인 논리식을 판별하는 방법은 무엇일까?

물론, 논리식의 의미를 정의한대로 따라가다 보면 결과를 알 수 있다: 참 혹은 거짓. 의미의 결과를 보면 논리식이 참인지 아닌지를 판별하면 된다. 이것은 프로그램을 돌려보고 그 프로그램이 참을 결과로 내는 지를 판별하는 것과 같다.

혹시, 프로그램을 돌리지 않고 알아내는 방법은 없을까? 논리식의 의미를 계산하지 않고, 참인지 거짓인지를 알 수 있는 방법은 없을까? 논리식의 모양만을 보면서 참인지를 판별할 수는 없을까? 이러한 판별규칙이 가능하지 않을까? 그런 규칙을 증명규칙(*proof rule*) 혹은 추론규칙(*inference rule*)이라고 한다. 이 규칙들도 어떤 집합을 만들어내는 귀납적인 규칙들로 정의된다. 그 규칙들이 만들어 내는 집합은 우리가 원하는 논리식들(참인 논리식들)로 구성되는 집합이다.

예를 들어 다음과 같은 증명규칙을 생각하자. 귀납규칙들을 분수꼴로 표현한 것이다.

$$\frac{}{(\Gamma, T)} \qquad \frac{}{(\Gamma, f)}\ f \in \Gamma \qquad \frac{(\Gamma, F)}{(\Gamma, f)} \qquad \frac{(\Gamma, \neg\neg f)}{(\Gamma, f)}$$

$$\frac{(\Gamma, f_1) \quad (\Gamma, f_2)}{(\Gamma, f_1 \wedge f_2)} \qquad \frac{(\Gamma, f_1 \wedge f_2)}{(\Gamma, f_1)}$$

$$\frac{(\Gamma, f_1)}{(\Gamma, f_1 \vee f_2)} \qquad \frac{(\Gamma, f_1 \vee f_2) \quad (\Gamma \cup \{f_1\}, f_3) \quad (\Gamma \cup \{f_2\}, f_3)}{(\Gamma, f_3)}$$

$$\frac{(\Gamma \cup \{f_1\}, f_2)}{(\Gamma, f_1 \Rightarrow f_2)} \qquad \frac{(\Gamma, f_1 \Rightarrow f_2) \quad (\Gamma, f_1)}{(\Gamma, f_2)}$$

$$\frac{(\Gamma \cup \{f\}, F)}{(\Gamma, \neg f)} \qquad \frac{(\Gamma, f) \quad (\Gamma, \neg f)}{(\Gamma, F)}$$

(Γ, f)쌍들의 집합을 만드는 규칙들인데, "Γ에 있는 모든 논리식들이 참이면 f는 참"인 경우의 (Γ, f)를 만들어 내는 규칙들이다. 혹은, 이 규칙들이 만들어 내는 원소들 중에서 $(\emptyset, f)$에 나타나는 f들이 참인 논리식들이다, 를 노리고 만들어진 규칙들이다.

대게 형식논리에서는, "(Γ, f)"를 "$\Gamma \vdash f$"로 표기한다. 규칙마다 번호를 붙이고 다시 쓰면:

$$\frac{}{\Gamma \vdash T}\ (r1) \qquad \frac{}{\Gamma \vdash f}\ f \in \Gamma\ \ (r2) \qquad \frac{\Gamma \vdash F}{\Gamma \vdash f}\ (r3) \qquad \frac{\Gamma \vdash \neg\neg f}{\Gamma \vdash f}\ (r4)$$

$$\frac{\Gamma \vdash f_1 \quad \Gamma \vdash f_2}{\Gamma \vdash f_1 \wedge f_2}\ (r5) \qquad \frac{\Gamma \vdash f_1 \wedge f_2}{\Gamma \vdash f_1}\ (r6)$$

$$\frac{\Gamma \vdash f_1}{\Gamma \vdash f_1 \vee f_2}\ (r7) \qquad \frac{\Gamma \vdash f_1 \vee f_2 \quad \Gamma \cup \{f_1\} \vdash f_3 \quad \Gamma \cup \{f_2\} \vdash f_3}{\Gamma \vdash f_3}\ (r8)$$

$$\frac{\Gamma \cup \{f_1\} \vdash f_2}{\Gamma \vdash f_1 \Rightarrow f_2}\ (r9) \qquad \frac{\Gamma \vdash f_1 \Rightarrow f_2 \quad \Gamma \vdash f_1}{\Gamma \vdash f_2}\ (r10)$$

$$\frac{\Gamma \cup \{f\} \vdash F}{\Gamma \vdash \neg f}\ (r11) \qquad \frac{\Gamma \vdash f \quad \Gamma \vdash \neg f}{\Gamma \vdash F}\ (r12)$$

이 증명규칙들을 $\Gamma \vdash f$들의 집합을 만드는 귀납규칙으로 볼 뿐 아니라, $\Gamma \vdash f$의 증명을 만드는 귀납규칙으로도 본다. (그래서 "증명규칙"이라고 부르는 것이다.) 예를 들어, 증명규칙

$$\frac{\Gamma \vdash f_1 \quad \Gamma \vdash f_2}{\Gamma \vdash f_1 \wedge f_2}$$

이 귀납적으로 이야기하는 것이 두가지다. $\Gamma \vdash f_1$와 $\Gamma \vdash f_2$가 집합에 있다면, $\Gamma \vdash f_1 \wedge f_2$도 집합의 원소여야 한다는 것이 하나. 그리고, $\Gamma \vdash f_1 \wedge f_2$의 증명을 만드는 (귀납적인) 방법으로, $\Gamma \vdash f_1$와 $\Gamma \vdash f_2$의 증명이 있다면 그 증명들을 분자에 매달고 분모에는 $\Gamma \vdash f_1 \wedge f_2$를 놓은 것이 $\Gamma \vdash f_1 \wedge f_2$의 증명을 만드는 방법이라는 것이다.

이렇게 만들어 지는 증명은 하나의 나무 구조가 된다. 나무의 갈래구조는 사용한 증명규칙들이 만들어 낸다. 규칙의 식들이 각각 노드가 되는 데, 분모식 노드에서 분자식 노드들로 가지치는 구조가 된다. 그 분자식들은 다시 다른 규칙의 분모가 되서 그 규칙의 분자식들로 다시 가지치고. 그 가지들의 최종 잎새는 증명규칙중에서 분자가 없는 규칙들의 분모식으로 끝맺게 된다. 이 나무 구조의 뿌리 노드는 증명의 최종 결론에 해당한다.

예를 들어, $\{p \to \neg p\} \vdash \neg p$의 증명이 위의 증명규칙에 의해 만들어 지면 아래와 같은 나무 구조가 되겠다:

$$\dfrac{\dfrac{}{\{p \to \neg p, p\} \vdash p}\,(r2) \qquad \dfrac{\dfrac{\dfrac{}{\{p \to \neg p, p\} \vdash p \to \neg p}\,(r2) \qquad \dfrac{}{\{p \to \neg p, p\} \vdash p}\,(r2)}{\{p \to \neg p, p\} \vdash \neg p}\,(r10)}{\{p \to \neg p, p\} \vdash F}\,(r12)}{\{p \to \neg p\} \vdash \neg p}\,(r11)$$

1.2.3 안전한 혹은 완전한

위의 규칙들이 만들어 내는 $(\emptyset, f)$의 f는 모두 참인가? 반대로, 참인 식들은 모두 위의 증명규칙들을 통해서 $(\emptyset, f)$꼴로 만들어 지는가? 첫 질문에 예 라고 답할 수 있으면 우리의 규칙은 안전하다(*sound*)고, 믿을 만 하다고 한다. 둘째 질문에 예 라고 답할 수 있으면 우리의 규칙은 완전하다(*complete*)고, 빠뜨림이 없다고 한다. 증명규칙들이 안전하기도 하고 완전하기도 하다면, 참인 식들만 빠짐없이 만들어 내는 규칙이 되는 것이다.

안전하냐 완전하냐는 것은 기계적인 증명규칙들의 성질에 대한 것이다. ("기계적인" 이란 "맹목적인, 의미를 생각치 않는" 이라는 뜻이다.) 그 기계적인 규칙들의 성질은 오직 우리가 생각하는 의미에 준해서 결정될 수 있다. 기계적인 증명규칙의 성질은 그 증명규칙이 만들어 내는 식들의 의미를 참조해야만 안전한지 완전한지, 혹은 그렇지 않은지를 결정할 수 있는 것이다. 신라면을 찍어내는 기계는 맹목적일 뿐이다. 그 기계가 좋은 이유는 항상 인기 있는 맛좋은 신라면을 만들어 낸다는 그 기계 바깥의 의미를 참조할 때에 비로소 결정된다.

프로그램에서도 그 겉모양(문법)과 속내용(의미)은 대게 다른 두개의 세계로 정의되고, 프로그램을 관찰하고 분석하는 모든 과정은 기계적으로 정의된다. (왜냐하면 컴퓨터로 돌리고 싶어서.) 그 기계적인 과정들의 성질들(과연 맞는지, 무엇을 하는 것인지) 등에 대한 논의는 항상 그 과정의 의미나 결과물들의 의미를 참조하면서 확인된다.