Proofmood, A Computer Logic System

정 주 희

2011 년 12 월 28 일

차 례

1 소개

`Proofmood`는 논리학을 위한 컴퓨터 소프트웨어 시스템이다. 여기서 '시스템'이라는 단어를 사용한 이유는 `Proofmood`가 하나의 소프트웨어로 이루어진 것이 아니라 관련있는 여러 개의 소프트웨어들로 이루어져 있기 때문이다. 이 소프트웨어들은 다음의 웹사이트에서 만나 볼 수 있으며, 웹프로그램이므로 사용자의 컴퓨터에 설치하지 않고 웹브라우저에서 즉시 사용할 수 있다.

http://www.proofmood.com

다만, 현재는 웹브라우저 중에서 Internet Explorer 만 지원하며 크롬, 파이어폭스, 사파리 등 타 웹브라우저의 지원은 개발 중이다.

현재 다음과 같은 4 개의 프로그램이 구현되어 있다.

1. Truth Table (진리표)
2. Logic Tableau (논리도해, 명제논리)
3. Fitch, Propositional (명제논리 피치 증명 시스템)
4. Fitch, 1st-order (1계논리 피치 증명 시스템)

이들 프로그램을 사용하기 위하여 어느 정도의 논리학 지식이 필요한 것이 당연한데, 이 매뉴얼은 독자들이 아래의 용어들만 알고 있으면 큰 어려움 없이 읽을 수 있도록 작성되었다.

명제, 부정, 역, 대우, 드모르강의 법칙

2 진리표

2.1 명제, 논리식, 진리값배정, 진리값 함수

진리표를 사용하기 위하여는 명제와 논리식이라는 두 용어를 잘 이해하고 있어야 한다.

명제는 참인지 거짓인지 분명히 판별할 수 있는 식이나 문장이며, 논리식은 명제를 기호로 나타낸 것이다. 아래의 표에 명제와 논리식의 간단한 예들을 보였다.

표 1: 명제와 논리식의 예

명제	논리식
소풍을 간다.	p
비가 온다.	r
비가 안 온다.	$\neg r$
비가 안 오면 소풍을 간다.	$\neg r \to p$
아픈 사람이 있다.	s
아픈 사람이 없다면, 비가 안 오면 소풍을 간다.	$\neg s \to (\neg r \to p)$
아픈 사람이 있다면, 비가 안 와도 소풍을 안 간다.	$s \to \neg(\neg r \to p)$

위의 논리식에 사용된 기호 중에 $\to$와 $\neg$는 각각 함의(*implication*)와 부정(*negation*)을 뜻하는 결합자(*connective*)이다. 즉, $A \to B$는 "A이면 B이다.", $\neg A$는 "A가 아니다."를 의미한다.

1 Exercise 아래의 명제들을 나타내는 논리식을 쓰시오. 단, "소풍을 간다"는 p, "비가 온다"는 r, "아픈 사람이 있다"는 s로 나타낸다.

논리식을 작성할 때 결합자 기호로 "또한"을 나타내는 $\wedge$와 "혹은"을 나타내는 $\vee$을 사용할 수 있다. 즉, $A \wedge B$는 "A이고 또한 B이다.", $A \vee B$는 "A이거나 혹은 B이다."를 뜻한다.

(i) 소풍을 간다면 비가 안 오고 아픈 사람도 없다.

(ii) 비가 오거나 아픈 사람이 있으면 소풍을 가지 않는다.

(iii) 비가 와도 소풍을 안 가고, 아픈 사람이 있어도 소풍을 안 간다. ⊣

결합자들의 명칭과 읽는 법을 아래의 표 2에 나타내었다. 결합자들의 의미는 명칭으로부터 대략 짐작할 수 있을 것이며, 정확한 정의는 표 4에 나와 있다.

함의문 $A \to B$의 앞인자와 뒤인자의 위치를 교환하여 얻은 함의문 $B \to A$는 $A \to B$의 역(*converse*)이라고 한다. 그리고 $\neg B \to \neg A$를 $A \to B$의 대우(*contrapositive*)라고 한다.

표 2: 결합자

기호	명칭	읽기
$\neg$	부정(*negation*)	부정, not
$\wedge$	논리곱(*conjunction*)	곱, and
$\vee$	논리합(*disjunction*)	합, or
$\rightarrow$	함의(*implication*)	이면, implies
$\leftrightarrow$	양방향함의(*biconditional*)	이면이[†], 동등, iff
$\perp$	모순(*contradiction*)	오, bottom

[†] $A \rightarrow B$는 'A 이면 B'로 읽으므로 $A \leftarrow B$는 'A 면이 B'로 읽고 $A \leftrightarrow B$는 'A 이면이 B'로 읽기로 한다.

논리식을 만들 때 괄호가 너무 많아져서 쓰기와 보기가 불편할 수 있다. 이를 피하기 위하여 (1) 가장 바깥의 괄호쌍을 제거하고, (2) 결합자 간의 우선순위(*priority*)를 정하여 사용하며, (3) $\wedge$와 $\vee$의 결합법칙을 이용하여 괄호를 생략하기로 한다.

논리식에서의 연산기호인 결합자들 간의 우선순위는 $\neg$이 $\wedge$과 $\vee$에 앞서며, $\wedge$과 $\vee$은 다시 $\rightarrow$, $\leftrightarrow$에 앞서는 것으로 정한다.

예를 들어 논리식

$$(((p \vee q) \wedge (\neg r)) \rightarrow q)$$

는 괄호를 여러 개 생략하여

$$(p \vee q) \wedge \neg r \rightarrow q$$

로 쓸 수 있다. 그러나 $((p \wedge q) \vee r)$은 $(p \wedge q) \vee r$로 쓸 수는 있어도 $p \wedge q \vee r$로 쓸 수는 없다. 왜냐하면 $\wedge$과 $\vee$의 우선순위가 같으므로 $p \wedge q \vee r$은 $(p \wedge q) \vee r$와 $p \wedge (q \vee r)$의 두 논리식 중 어느 것을 의미하는지 불분명하기 때문이다.

결합자 $\wedge$의 결합법칙이란 논리식 $(A \wedge B) \wedge C$는 $A \wedge (B \wedge C)$와 의미가 같다는 것이며 따라서 이 논리식들에 괄호를 넣지 않고 그냥 $A \wedge B \wedge C$로 써도 된다. 같은 이유로 $A \vee B \vee C \vee D$ 같은 문자열도 합법적인 논리식이다.

두 논리식의 "의미가 같다"는 것은 곧 두 논리식이 동등(*equivalent*)하다는 것인데 논리식의 동등이라는 개념에 대한 설명은 5쪽에 나와 있다. 흔히 '동등'을 써야 할 자리에 '동치'라는 용어를 쓰는데 이것은 일본식 표현임을 알아야 할 것이다.

논리식의 수학적 의미는 그것을 구성하는 명제문자들의 진리값 함수(*truth function*)에 의하여 규정된다. 예를 들어 논리식 $p \rightarrow q$의 본래 의도된 의미는 "p이면 q이다." 이지만, 수학적 의미는 다음과 같은 함수 f일 뿐이다.

$$f(1,1) = 1, \quad f(1,0) = 0, \quad f(0,1) = 1, \quad f(0,0) = 1 \tag{1}$$

여기서 1은 참, 0은 거짓을 나타낸다.

$(1,1), (1,0)...$ 등은 명제문자 p, q에 주어진 진리값배정(*truth value assignment*)이라고 부른다. 진리값배정은 흔히 변수 $\bar{x}$, $\bar{y}$ 등을 사용해서 나타낸다. 예를 들어

진리값배정 $\bar{x} = (1,1)$에서 논리식 $p \rightarrow q$의 진리값은 1로 계산된다.

와 같이 말할 수 있다. 혹은 더 간단히 "$\bar{x} = 11$에서 ..." 와 같이 써도 된다.

다시 말해서 논리식 A의 의미는 그것을 구성하는 명제문자들에 다양한 진리값배정이 주어졌을 때 A의 진리값이 어떻게 나오느냐에 의하여 규정된다.

진리값 함수는 진리표(*truth table*)로 나타내는 것이 일반적이다. 예를 들어 (1)에 나왔던 $p \to q$의 진리값 함수 f를 진리표로 나타내면 표 3과 같다.

표 3: 진리값 함수를 나타내는 두 가지 방법

p	q	$p \to q$
1	1	1
1	0	0
0	1	1
0	0	1

$\bar{x}$	$f(\bar{x})$
11	1
10	0
01	1
00	1

이 진리값 함수의 정의역은 4개의 원소로 구성된 집합 $\{(1,1),(1,0),(0,1),(0,0)\}$, 혹은 $\{11,10,01,00\}$이며 공역은 $\{1,0\}$이다. 일반적으로 n 개의 명제문자로 구성된 논리식에 대응되는 진리값 함수[†]의 정의역은 2^n 개의 원소로 구성되며 공역은 항상 $\{1,0\}$이다. 이 진리값 함수를 나타내는 또 하나의 방법이 있는데 그것은 1011과 같이 함수의 값을 차례로 나열하는 것이다. 이 3번째 방법이 가장 간편하므로 우리는 진리값 함수를 나타낼 때 주로 이 방법을 사용할 것이다.

앞서 명제논리에서 사용되는 결합자들의 의미를 표 2에서 설명하였는데 이를 진리표를 사용하여 수학적으로 명확하게 정의하여 표 4에 보였다.

표 4: 결합자의 의미

p	q	$\neg p$	$p \wedge q$	$p \vee q$	$p \to q$	$p \leftrightarrow q$	$\bot$
1	1	0	1	1	1	1	0
1	0	0	0	1	0	0	0
0	1	1	0	1	1	0	0
0	0	1	0	0	1	1	0

앞서 나왔던 논리식 $(p \vee q) \wedge \neg r \to q$의 의미를 나타내는 진리값함수를 얻기 위하여 진리표를 그리면 그림 1과 같다. 이 진리표를 작성하는 방법을 아래에 설명하였다.

(1) 왼쪽에 있는 3 개의 열에 진리값배정대로 진리값을 써 넣는다. 명제문자가 3개이므로 진리값배정은 $2^3 = 8$ 개가 존재할 것이다.

(2) 맨 아래 행의 각 쎌(cell)에는 대응되는 부분논리식의 계산단계를 써 넣는다. 부분논리식이 원자논리식일 때는 그 쎌이 속한 열의 맨 위 행에 그 명제문자가 나타나 있고, 부분논리식이 합성논리식일 때는 맨 위 행에 그 부분논리식의 주결합자가 나타나 있다. 계산단계 중 전체 논리식의 주 결합자의 열에 대응하는 것은 그림에서 보듯이 굵은 글씨로 나타내어 눈에 잘 뜨이도록 하는 것이 좋다.

(3) 1 단계 열에 속한 쎌들의 진리값은 처음 3 개 열에 나와 있는 진리값배정에서 그대로 베껴 쓴다. 이는 각 1 단계 열들의 맨 위 행에 명제문자가 나타나 있기 때문에 아주 쉽다.

[†]정확히 말하자면 진리값 함수는 논리식으로부터 결정되는 것이 아니라 논리식과 그 논리식에 속하는 모든 명제문자들을 포함하는 명제문자들의 열이 주어지면 비로소 결정된다.

그림 1: 논리식 $(p \vee q) \wedge \neg r \rightarrow q$의 진리표

p	q	r	(	p	∨	q	)	∧	¬	r	→	q
1	1	1		1	1	1		0	0	1	1	1
1	1	0		1	1	1		1	1	0	1	1
1	0	1		1	1	0		0	0	1	1	0
1	0	0		1	1	0		1	1	0	0	0
0	1	1		0	1	1		0	0	1	1	1
0	1	0		0	1	1		1	1	0	1	1
0	0	1		0	0	0		0	0	1	1	0
0	0	0		0	0	0		0	1	0	1	0
계산단계				1	2	1		3	2	1	**4**	1

(4) 2 이상 단계의 열에서의 진리값은 좌우에 있는 열들의 진리값(이미 계산되어 있을 것임)과 그 열의 맨 위 행에 있는 결합자로부터 표 4에 의하여 계산하여 써 넣는다.

(5) 마지막 단계의 열까지 진리값을 구해 넣으면 진리표가 완성된다.

주어진 논리식의 의미를 나타내는 진리값함수는 이 표에서 마지막 단계의 열, 즉 주결합자의 열에 나타나게 된다. 위의 예에서는 이 진리값함수가 11101111으로 계산되었다.

아래의 그림에서 두 개의 논리식 $p \vee q$와 $\neg p \rightarrow q$의 진리표를 비교하였다.

그림 2: 동등한 두 논리식

p	q	p	∨	q	¬	p	→	q
1	1	1	1	1	0	1	1	1
1	0	1	1	0	0	1	1	0
0	1	0	1	1	1	0	1	1
0	0	0	0	0	1	0	0	0
계산단계		1	**2**	1	2	1	**3**	1

이 진리표의 모든 행에서, 즉 모든 진리값배정에서 두 논리식의 진리값이 일치함을 볼 수 있다. (두 논리식의 진리값 함수는 1110으로 동일하다.) 이런 경우 두 논리식은 논리적으로 동등(*equivalent*)하다고 말한다.

그림 3에 동등한 두 논리식의 예를 또 하나 보였다. 진리표에서 보듯이 이 두 논리식은 모든 행에서 진리값이 1 이라는 특성을 가진다. 이런 논리식들을 항진(*logically valid formula*), 혹은 토톨로지(*tautology*)라고 부른다. 모든 항진 논리식들이 서로 동등함은 당연하다.

그림 3: 항진 논리식의 예

p	q	p	∨	¬	p	(	p	∨	q	)	↔	(	¬	p	→	q	)
1	1	1	1	0	1		1	1	1		1		0	1	1	1	
1	0	1	1	0	1		1	1	0		1		0	1	1	0	
0	1	0	1	1	0		0	1	1		1		1	0	1	1	
0	0	0	1	1	0		0	0	0		1		1	0	0	0	
계산단계		1	**3**	2	1		1	2	1		**4**		2	1	3	1	

항진에 대칭되는 개념으로 모순(*contradiction*)이 있는데, 이는 $p \wedge \neg p$와 같이 모든 진리값배

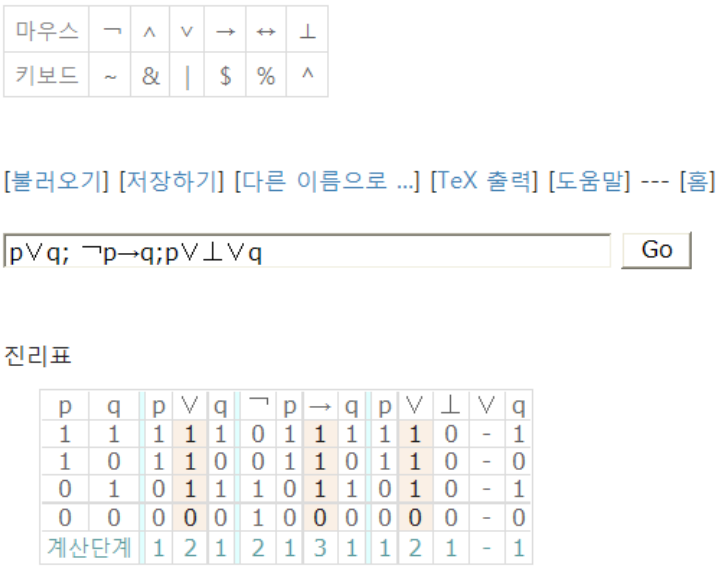
정에서 진리값이 0으로 계산되는 논리식을 말한다. 일반적으로 항진의 부정은 모순이고 모순의 부정은 항진임은 당연하다.

2.2 프로그램 사용법

제일 먼저 알아야 할 것은 결합자 기호를 입력하는 방법인데, 이는 프로그램 화면을 보면 곧 알 수 있을 것이다. 원하는 기호를 마우스로 클릭하면 된다. 그림 4에 프로그램 화면의 예를 보였다.

키보드를 사용하면 마우스를 사용하는 것보다 더 빨리 입력할 수 있다. 각 결합자에 대응하는 키는 화면에 나타나 있다. 예를 들어 부정결합자 ¬는 ~을 치면 입력되고 함의결합자 →는 $를 치면 입력된다.

그림 4: 진리표 프로그램 화면



2개 이상의 논리식의 진리표를 동시에 작성하려면 논리식들의 사이에 세미콜론(;)을 넣으면 된다.

그림 4의 3번째 논리식 $p \vee \bot \vee q$에서 보듯이 결합법칙이 성립하는 결합자 ∨, 혹은 ∧를 3개 이상 연달아 사용하는 경우에는 맨 왼쪽 결합자 밑의 열에 진리값을 계산하여 보여 주고, 두번째 결합자부터는 비워 놓는다. (비어 있다는 의미로 – 를 보여 준다.)

이 진리표 프로그램에서 사용한 논리식은 저장해 두었다가 다시 읽을 수 있다. 화면에 보이는 [불러오기], [저장하기], [다른 이름으로...] 등의 버튼을 클릭하여 사용법을 쉽게 익힐 수 있을 것이다. 다만 이 데이터베이스는 모든 사람이 공유하므로 내가 저장한 논리식을 다른 사람이 지울 수 있음에 유의해야 한다.†

명제문자는 영문 대문자, 소문자 모두 사용이 가능하며 길이가 2이상인 문자열도 허용된다. 상세한 것은 [도움말]을 클릭하면 알 수 있다. [TeX 출력]에 대해서도 역시 [도움말]에서 알아볼 수 있다.

†사용자가 로그인하여 자신만의 공간에 저장할 수 있도록 하는 기능은 현재 개발중임.

3 논리도해

3.1 모델, 만족관계, 논리적 귀결관계

진리값배정을 나타내는 변수로 흔히 변수 $\bar{x}$ 를 사용한다. 예를 들어

$\bar{x} = 10$ 에서 $p \wedge \neg q$ 는 참이다.

라고 말한다. 위의 문장을 기호로는

$$\bar{x} \vDash p \wedge \neg q$$

로 나타내고 "$\bar{x}$ models $p \wedge \neg q$", 혹은 "$\bar{x}$ 야 $p \wedge \neg q$" 라고 읽는다. 그리고

$\bar{x}$ 는 $p \wedge \neg q$ 를 만족(*satisfy*) 한다.

고 말하기도 한다.

$\vDash$ 의 우변에는 논리식의 집합을 둘 수도 있다. 예를 들어

$$\bar{x}_1 = 10,\ \bar{x}_2 = 11,\ \Phi = \{p \wedge \neg q,\ q \to p\}$$

로 두었을 때

$$\bar{x}_1 \vDash \Phi, \quad \bar{x}_2 \nvDash \Phi$$

가 성립한다. 즉, Φ 의 모든 원소가 $\bar{x}$ 에서 참일 때 $\bar{x} \vDash \Phi$ 인 것으로 정의한다.

Φ 의 모델집합, 즉 Φ 의 모든 모델들의 집합을 $\mathrm{Mod}(\Phi)$ 로 나타낸다. 식으로 쓰면 다음과 같다.

$$\{\bar{x} \in \{1,0\}^n \mid \bar{x} \vDash \Phi\}$$

$\mathrm{Mod}(\Phi) = \varnothing$ 일 때 Φ 를 만족불가능(*unsatisfiable*) 이라고 한다.

A 가 논리식일 때 $\mathrm{Mod}(\{A\})$ 대신 $\mathrm{Mod}(A)$ 로 써도 되는 것으로 한다. 그러면

$$A\text{는 모순} \Leftrightarrow \mathrm{Mod}(A) = \varnothing,$$
$$A\text{는 항진} \Leftrightarrow \mathrm{Mod}(A) = \text{전체집합}$$

이 된다. 논리식에 있어 항진과 모순은 특수한 경우이고 일반적인 논리식은 이 둘 중 어느 하나도 아닌 경우가 많다.

$\mathrm{Mod}(\Phi) \subseteq \mathrm{Mod}(A)$ 일 때, 그리고 이때만

$$\Phi \vDash A$$

라고 쓰고 "Φ 야 A", 혹은 "Φ entails A" 로 읽는다. 그러나 "Φ models A" 로 읽지는 않는다. 그리고 이럴 때 A 를 Φ 의 논리적 귀결(*logical consequence*) 라고 말한다.

그러니까 기호 $\vDash$ 는 때로는 만족관계를 나타내고 때로는 논리적 귀결관계를 나타낸다. 즉 문맥에 따라 두 가지 서로 다른 의미로 쓰일 수 있으므로 주의해야 한다.

2 Exercise 다음의 논리적 귀결관계들을 증명하시오.

$p,\ p \to q \vDash q,$

$p \lor q,\ p \to r \vDash (q \to r) \to r,$

$A,\ A \lor B \nvDash B,$

$(A \to B) \to C \nvDash B \lor C.$ ⊣

3 Lemma $A_1, \ldots, A_n$ 과 B 가 논리식일 때,

(1) $A_1, \ldots, A_n \vDash B \Leftrightarrow A_1 \land \cdots \land A_n \to B$ 는 항진

(2) $A_1, \ldots, A_n \vDash B \Leftrightarrow A_1, \cdots, A_{n-1} \vDash A_n \to B$
(A_n 을 $A_i (i = 1, \ldots, n)$ 로 바꿔 넣어도 됨.)

(3) $A_1, \ldots, A_n \vDash B \Leftrightarrow \{A_1, \ldots, A_n, \neg B\}$가 만족불가능[†]

(4) $A_1, \ldots, A_n$ 과 $B_1, \ldots, B_m$ 이 논리식일 때,

$$A_1, \ldots, A_n \vDash B_1 \lor \cdots \lor B_m \Leftrightarrow$$
$$A_1, \ldots, A_n, \neg B_1 \vDash B_2 \lor \cdots \lor B_m \Leftrightarrow$$
$$A_2, \ldots, A_n \vDash \neg A_1 \lor B_1 \lor B_2 \lor \cdots \lor B_m$$

이 성립한다. A_1 대신 A_i $(i \le n)$, B_1 대신 B_j $(j \le m)$ 을 써도 된다. ⊣

3.2 논리도해 수형도

진리표를 이용하여 주어진 논리식 집합의 모델집합을 구할 때 문제가 되는 것이 있으니 그것은, 명제문자의 개수가 늘어남에 따라 진리표의 크기가 기하급수적으로 커진다는 것이다. 명제문자의 개수가 5 정도만 되어도 컴퓨터 없이 손으로 진리표를 그리기가 현실적으로 힘들 것이다.

진리표를 사용하지 않고 주어진 논리식들을 만족하는 진리값배정들을 찾아내는 방법 중 하나가 바로 논리도해이다.

하나의 예로 논리식 4 개로 이루어진 집합

$$\Phi \stackrel{\text{def}}{=} \{\neg a, b \lor d, \neg c \to a, b \to \neg c\} \tag{2}$$

의 모델집합 $\mathrm{Mod}(\Phi)$ 를 찾아 보자.

$\bar{x} \vDash \Phi$ 를 가정하자. 그러면 $\bar{x}$ 에서 $b \lor d$ 가 참이므로 $\bar{x}$ 에서 b 가 참이거나 혹은 d 가 참일 것이다. 그러므로 $\bar{x}$ 에서 $\Phi \cup \{b\}$ 가 만족되거나 혹은 $\Phi \cup \{d\}$ 가 만족된다. 그러므로 아래의 수형도에서 왼쪽 가지 $\Phi \cup \{b\} = \{\neg a, b \lor d, \neg c \to a, b \to \neg c, b\}$ 가 만족되거나 혹은 오른쪽 가지 $\Phi \cup \{c\} = \{\neg a, b \lor d, \neg c \to a, b \to \neg c, d\}$ 가 만족된다.

$$\begin{array}{c} \neg a,\ \ b \lor d \\ \neg c \to a,\ \ b \to \neg c \\ \hline b \qquad d \end{array}$$

역으로 $\bar{x} \vDash \Phi \cup \{b\}$ 이거나 $\bar{x} \vDash \Phi \cup \{c\}$ 이면 $\bar{x} \vDash \Phi$ 가 성립함은 당연하다.

$\bar{x} \vDash \neg c \to a \in \Phi$ 이면 $\bar{x} \vDash c \lor a$ 이다. 왜냐하면 $\neg c \to a$ 와 $c \lor a$ 가 동등하기 때문이다. 그러므로 앞서와 같은 논리로 Φ 의 모델은 아래의 수형도의 3 가지 중 어느 하나의 모델이고 역으로 3 가지 중 어느 하나의 모델은 Φ 의 모델이 된다.

[†](3)의 $\Leftrightarrow$ 중에서 $\Leftarrow$를 이용한 증명 방법을 귀류법이라 한다.

$$\dfrac{\dfrac{\neg a,\ \ b \vee d \qquad \neg c \to a,\ \ b \to \neg c}{\dfrac{b}{\dfrac{c}{\ }} \qquad d}}{\ } \qquad \begin{matrix} a \\ (\times) \end{matrix}$$

그런데 가운데 가지는 $\neg a$ 와 a 를 동시에 포함하므로 절대로 만족될 수 없다. 그래서 이 가지는 죽은 것으로 결론 내리고 $(\times)$ 로 표시하였다.

이제 $(b \to \neg c) \in \Phi$ 가 $\neg b \vee \neg c$ 와 동등함을 이용하여 첫번째 가지를 다시 가지치기 하면 아래의 수형도를 얻는다.

$$\dfrac{\dfrac{\dfrac{\neg a,\ \ b \vee d \qquad \neg c \to a,\ \ b \to \neg c}{\dfrac{b}{\dfrac{c}{\ }} \qquad d}}{\ } \qquad \begin{matrix} a \\ (\times) \end{matrix}}{\ } \qquad \begin{matrix} \neg b \\ (\times) \end{matrix} \qquad \begin{matrix} \neg c \\ (\times) \end{matrix}$$

첫번째 가지에는 $\neg b$ 와 b 가 동시에 포함되어 있으므로 $(\times)$ 표 하고, 두번째 가지에는 $\neg c$ 와 c 가 동시에 포함되어 있으므로 $(\times)$ 표 한다.

이제 4 개의 가지 중 유일하게 살아 남아 있는 4 번째 가지에서 지금까지와 같은 방법으로 다시 가지치기를 (두 번) 하면 다음과 같은 결과를 얻는다.

$$\dfrac{\dfrac{\dfrac{\neg a,\ \ b \vee d \qquad \neg c \to a,\ \ b \to \neg c}{\dfrac{b}{\dfrac{c}{\ }} \qquad \begin{matrix} a \\ (\times) \end{matrix}} \qquad \dfrac{d}{\dfrac{c}{\ }} \qquad \begin{matrix} a \\ (\times) \end{matrix}}{\ } \qquad \begin{matrix} \neg b \\ (\times) \end{matrix} \qquad \begin{matrix} \neg c \\ (\times) \end{matrix} \qquad \begin{matrix} \neg b \\ \ \end{matrix} \qquad \begin{matrix} \neg c \\ (\times) \end{matrix}$$

총 6 개의 가지 중에 4 번째 가지만 살아 남았다. 이 가지에 속한 논리식들의 집합을 Θ 라 하면 다음과 같이 쓸 수 있다.

$$\{\neg a, b \vee d, \neg c \to a, b \to \neg c, d, c, \neg b\} \stackrel{\text{def}}{=} \Theta \supseteq \Phi$$

$\mathrm{Mod}(\Phi) = \mathrm{Mod}(\Theta)$ 임은 지금까지 설명한 바와 같다. 그런데 Θ 에는 리터럴 $\neg a$, $\neg b$, c, d 가 들어 있으므로 $\bar{x} \vDash \Theta$ 인 $\bar{x}$ 는 0011 일 수밖에 없다. 그러므로 $\mathrm{Mod}(\Phi) = \{0011\}$ 이다.

위에서 얻은 Φ 의 논리도해는 분명 16 개의 행을 가진 Φ 의 진리표보다 간단하다. 이 예를 통하여 논리도해는 주어진 논리식들의 집합의 모델집합을 찾는 도구로 쓰일 수 있음을 알 수 있을 것이다.

논리식의 집합의 모델집합을 찾는 도구로 논리도해가 반드시 진리표보다 더 유용하다는 보장은 없다. 일반적으로 Φ 가 만족불가능이거나 이에 가까울 때 (즉 만족하는 진리값배정이 1,2 개 이내일 때)는 논리도해가 유용하다. 왜냐하면 일단 어떤 하나의 가지를 $(\times)$ 표 하게 되면 그 가지는 더 이상 고려할 필요가 없게 되기 때문이다.

위의 논리도해를 작성하는 데 사용된 규칙은 아래에 보인 것 하나 뿐이다.

$$\vee \text{ rule } \frac{\begin{array}{c} A \vee B \\ \vdots \end{array}}{A \qquad B}$$

$b \to \neg c$ 같은 함의문은 일단 이것과 동등한 논리식 $\neg b \vee \neg c$로 바꾼 다음 위의 규칙을 적용하였다. 그리고 $\neg c \to a$는 이것과 동등한 논리식 $c \vee a$로 바꾸어 사용했는데 이는 실은 $\neg c \to a$를 일단 $\neg\neg c \vee a$로 바꾼 후 다시 $\neg\neg c$를 c로 바꾸는 $\neg\neg$ 규칙을 적용한 것이다. 그러므로 아래의 두 규칙도 이미 사용한 셈이다.

$$\to \text{ rule } \frac{\begin{array}{c} A \to B \\ \vdots \end{array}}{\neg A \qquad B} \qquad\qquad \neg\neg \text{ rule } \frac{\begin{array}{c} \neg\neg A \\ \vdots \end{array}}{A}$$

이제 다음과 같은 6개의 규칙을 더하면 논리도해 작성의 규칙이 완비된다.

$$\neg\vee \text{ rule } \frac{\begin{array}{c} \neg(A \vee B) \\ \vdots \end{array}}{\neg A,\ \neg B} \qquad\qquad \neg\to \text{ rule } \frac{\begin{array}{c} \neg(A \to B) \\ \vdots \end{array}}{A,\ \neg B}$$

$$\leftrightarrow \text{ rule } \frac{\begin{array}{c} A \leftrightarrow B \\ \vdots \end{array}}{A,\ B \qquad \neg A,\ \neg B} \qquad\qquad \wedge \text{ rule } \frac{\begin{array}{c} A \wedge B \\ \vdots \end{array}}{A,\ B}$$

$$\neg\leftrightarrow \text{ rule } \frac{\begin{array}{c} \neg(A \leftrightarrow B) \\ \vdots \end{array}}{\neg A,\ B \qquad A,\ \neg B} \qquad\qquad \neg\wedge \text{ rule } \frac{\begin{array}{c} \neg(A \wedge B) \\ \vdots \end{array}}{\neg A \qquad \neg B}$$

논리도해 생성규칙을 적용하여 수형도를 확장하는 작업을 터뜨리기(*pop*)라고 부르기로 하자.

터뜨리기 중에 $\vee$, $\to$, $\leftrightarrow$, $\neg\leftrightarrow$, $\neg\wedge$ rule을 적용한 것을 가지치기(*branching*)라고 부르고, $\neg\neg$, $\neg\vee$, $\neg\to$, $\wedge$ rule을 적용한 것을 쌓아놓기(*piling up*)라고 부른다. 전자는 2개의 가지를 생성하고 후자는 그렇지 않는다는 차이가 있다.

지금까지 공부한 것을 정리하면 다음과 같다. Φ를 뿌리 노드에 둔 논리도해 수형도가 주어졌다 하자. 이 수형도의 각 단말점마다 하나의 가지가 대응된다. n개의 단말점이 있는 경우 가지들을 $\Theta_1, \ldots, \Theta_n$라 한다면 다음의 등식이 성립한다.

$$\mathrm{Mod}(\Phi) = \mathrm{Mod}(\Theta_1) \cup \cdots \cup \mathrm{Mod}(\Theta_n).$$

Θ_i의 단말점에 $(\times)$표가 되어 있다면 $\mathrm{Mod}(\Theta_i) = \varnothing$이다.

$(\times)$표 되어 있지 않은 경우만 생각하자. Φ에 사용된 모든 명제문자들이 Θ_i에 리터럴 형태로 나타나 있다면 $\mathrm{Mod}(\Theta_i)$을 구하기는 아주 쉽다. 그렇지 않다면 진리표 등 다른 수단을 동원하여 $\mathrm{Mod}(\Theta_i)$를 구한다.

위에 설명한 내용은 논리도해가 완성되지 않은 상태에서도, 즉 터뜨리기를 할 것이 남아 있는 가지가 있는 상태에서도 유효하다.

4 EXAMPLE $\Phi = \{p,\ p \vee q,\ r \to \neg q\}$ 의 모델집합을 구해 보자. 먼저 논리도해를 다음과 같이 얻는다.

$$\dfrac{\dfrac{p,\ p \vee q,\ r \to \neg q}{p} \qquad \dfrac{}{q}}{\neg r \qquad \neg q \qquad \neg r \qquad \underset{(\times)}{\neg q}}$$

가지는 다음과 같이 4 개 있다. Θ_1 과 Θ_2 에서는 p 가 중복되어 나타나므로 하나만 보였다. 그리고 Θ_4 에서는 서로 부정인 리터럴 q 와 $\neg q$ 가 들어 있어 $(\times)$ 표 하였다.

$$\begin{aligned}
\Theta_1 &= \{p,\ p \vee q,\ r \to \neg q,\ \neg r\},\\
\Theta_2 &= \{p,\ p \vee q,\ r \to \neg q,\ \neg q\},\\
\Theta_3 &= \{p,\ p \vee q,\ r \to \neg q,\ q,\ \neg r\},\\
\Theta_4 &= \{p,\ p \vee q,\ r \to \neg q,\ q,\ \neg q\}.
\end{aligned}$$

Θ_1 에는 명제문자 p 와 r 에 대한 리터럴은 각각 p 와 $\neg r$ 로 나와 있으며 q 에 대한 리터럴은 나와 있지 않으므로 $\mathrm{Mod}(\Theta_1) \subseteq \{100, 110\}$ 임을 알 수 있다. $\supseteq$ 도 성립함은 쉽게 확인할 수 있을 것이다. Θ_2 와 Θ_3 에 대해서도 이런 식으로 모델집합을 구하고 이들의 합집합을 취하면

$$\mathrm{Mod}(\Phi) = \{100, 110\} \cup \{100, 101\} \cup \{110\} = \{100, 110, 101\}$$

를 얻게 된다. 논리식 $p \wedge (p \vee q) \wedge (r \to \neg q)$ 의 진리표를 그려 보아도 같은 결과를 얻게 될 것이다. ⊣

3.3 프로그램 사용법, 추론

$\{A \vee B, \neg A \wedge \neg B\}$ 가 만족불가능임을 `proofmood.com` 의 논리도해 프로그램을 이용하여 보이는 과정을 그림 5 와 그림 6 에 나타내었다.

그림 5: 논리도해 1

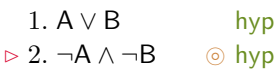


그림 5 에서 2 번 라인 맨 왼쪽에 보이는 붉은 삼각형 기호는 현재 이 라인에 포커스 (focus) 가 놓여 있다는 뜻이다.

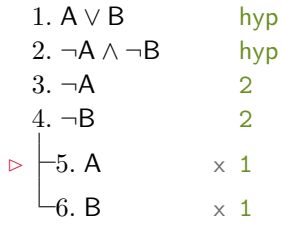
그림 오른쪽의 `hyp` 는 가설 (*hypothesis*) 을 뜻하는 주석 (*annotation*) 이다. 이 두 라인은 가설, 즉 뿌리노드에 넣는 논리식으로 주어졌다는 의미이다. 2 번 라인 `hyp` 의 바로 왼편에 동그라미 모양의 기호가 보이는데 이것은 수형도에서 살아 있는 가지, 즉 $(\times)$ 표 하지 않은 가지의 단말점을 나타내는 것이다.

라인 번호 바로 왼편의 영력에 마우스를 올려 놓으면 마우스가 손 모양으로 바뀐다. 이 영역을 라인의 클릭영역이라고 부르기로 하자.

라인의 논리식을 터뜨리려면 그 라인의 클릭영역을 `Shift-Click` 하면 된다. 1 번과 2 번을 터뜨리면 그림 6 과 같은 수형도가 얻어진다. 이 수형도는 1 개의 내부점 (4 개의 라인 1,2,3,4 로

이루어짐)과 2개의 단말점(각각 라인5 및 라인 6로 이루어짐)으로 구성되어 있다. 키보드의 위/아래 화살표, 혹은 마우스를 이용하여 포커스를 이리저리 이동시켜 본다. 마우스를 화면 오른편 주석영역의 라인 번호 및 × 표 위에 올려 놓아 본다.

그림 6: 논리도해 2



3번과 4번 라인의 주석에는 숫자 2가 보이는데 이것은 이들 라인이 2번 라인을 전제(*premise*)로 삼아 터뜨려서 나온 결과라는 뜻이다. 그리고 포커스가 3번 혹은 4번 라인에 놓여 있을 때는 2번 라인의 논리식이 파란색으로 나타나는데 이것은 2번 라인이 현재 포커스가 있는 라인의 전제(의 일부)라는 것을 의미한다. 또한 이때 3,4 번 라인은 초록색이며 이것은 이 두 라인들이 같은 전제로부터 가지치기 하여 얻어진 형제(*sibling*)라는 사실을 나타내고 있다.

3번 라인의 주석의 전제 번호 2에 마우스를 올리면 3번 라인을 얻는 데 사용된 터뜨리기 규칙 [쌓아놓기 ∧]가 말풍선으로 나타난다.

5번, 6번 라인의 주석을 보면 이 라인들을 얻을 때 사용한 전제가 1번 라인임을 알 수 있다. 전제 번호 바로 왼편의 × 표는 이 라인을 포함하는 가지는 죽었다는, 즉 만족불가능이라는 뜻이다. 이 논리도해에 존재하는 2개의 가지가 모두 만족불가능이므로 처음에 주어진 논리식들, 즉 주석이 hyp인 1번, 2번 가설 논리식들의 집합은 만족불가능이라고 결론지을 수 있다.

6번 라인 주석의 × 표에 마우스를 올리면 이 가지가 모순인 이유인, 서로 모순인 4번과 6번 논리식이 붉은색으로 나타남을 관찰할 수 있을 것이다. 그리고 이 가지에 속한 모든 5개의 논리식들의 번호는 파란색으로 나타날 것이다.

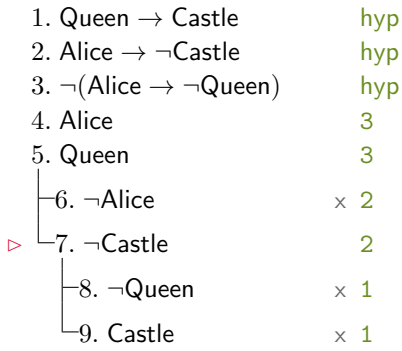
그림 6의 논리도해는 1번 라인을 터뜨린 다음에 2번 라인을 터뜨려서 얻은 것인데 순서를 바꾸어 2번 다음에 1번을 터뜨려도 동일한 논리도해를 얻게 될 것이다. 이를 확인하기 위하여는 화면 상단의 [새 논리도해]를 클릭하여 새 논리도해를 열어 처음부터 새로 작성해도 되나 1번, 2번 라인은 남겨놓고 3,4,5,6번 라인들만 삭제하고 다시 작성해도 된다.

라인의 삭제는 그 라인에 포커스가 놓인 상태에서 [줄 삭제]를 클릭하면 된다. 그러나 5번, 6번 라인은 이렇게 해서는 삭제가 되지 않는다. 둘 중에 어느 하나만 삭제하면 이 논리도해의 무결성(integrity)이 손상되기 때문이다. 4번 라인에 포커스를 두고 [부분나무 삭제]를 클릭하면 바로 그 아래의 5번, 6번으로 이루어진 부분나무가 삭제된다. 실은 이때 4번 라인과 같은 노드에 속하는 1번, 2번, 3번 라인 중 어느 것에 포커스를 두고 클릭해도 같은 결과가 나온다.

위의 예에서 보듯이 논리도해에서 터뜨리기의 순서는 중요하지 않을 수 있다. 그러나 터뜨리는 순간 포커스의 위치는 때로 매우 중요하다. 예를 들어 그림 7의 논리도해는 3번, 2번, 1번의 순서로 터뜨려서 얻었는데 맨 마지막에 1번을 터뜨릴 때 반드시 포커스를 7번 라인에 두어야 한다. 포커스가 7번 아닌 다른 라인에 있을 때 1번 라인의 터뜨리기를 시도하면 어떤 현상이 일어나는지 관찰해 보라.

어떤 라인을 [가지치기] 하면 번호가 빨간색으로 나타나 있는 라인 바로 밑에 새로운 부분나무가 생성·삽입된다. 만일 글번호가 빨간색으로 나타나 있는 라인이 존재하지 않는다면 포커스를

그림 7: 논리도해 3



옮기기 전에는 가지치기가 불가능하다. 그림 7에서 이 사실을 확인해 보기 바란다. 번호가 빨간색으로 나타나는 라인은 포커스가 놓인 라인을 포함한 노드에 속한 가장 아래쪽에 있는 라인이다. 단, 이때 이 노드는 단말점이어야만 한다. 그러므로 만일 포커스가 내부점에 속한 라인에 놓여 있다면 번호가 빨간색으로 나타나는 라인은 존재하지 않게 된다.

어떤 라인을 [쌓아놓기] 하면 그 라인이 속한 수형도 노드의 맨 아래에 새 논리식들이 삽입된다. 그러므로 쌓아놓기의 경우 포커스의 위치는 (가지치기의 경우와는 달리) 어디 있든지 상관없다.

논리도해에서 작업 결과를 저장하고 다시 읽어 들이는 기능은 [출력] 및 [입력] 버튼에 구현되어 있다. 출력 버튼을 클릭하면 현재 화면에 보이는 논리도해를 인코딩한 문자열이 팝업창에 나타난다. 팝업창 화면을 `Ctrl-A`, `Ctrl-C` 하여 버퍼에 복사한 후 원하는 문서에 `Ctrl-V` 하여 붙여 넣으면 출력·저장된 것이다.

출력한 것을 읽어 들이려면 저장된 파일의 내용을 버퍼에 복사한 후 [입력] 버튼을 클릭하여 팝업된 창에 붙여 넣으면 된다.

논리도해의 도구로 '터뜨리기' 외에 추론(*inference*)을 더한다. 이 방법의 이론적 근거는

$$A_1, \ldots, A_n \vDash B$$

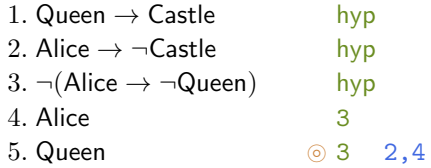
일 때

$$\mathrm{Mod}(A_1, \ldots, A_n, B) = \mathrm{Mod}(A_1, \ldots, A_n)$$

이라는 것이다.

이 간단한 사실은 논리도해에서 대단히 강력한 도구로 사용될 수 있다. 앨리스 추론을 예로 들어 설명하겠다.

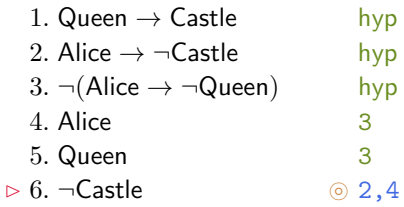
그림 8: 앨리스 추론 1



현재 화면은 가설 라인 1,2,3 중 3번 가설을 터뜨려(쌓아놓기 하여) 4번, 5번 라인이 추가로 얻어진 상태다. 이제 2번과 4번 라인의 논리적 귀결인 ¬Castle을 5번 라인 아래에 넣어 보자.

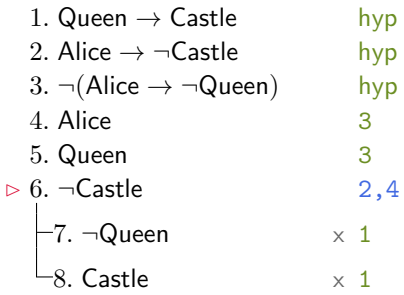
5번 라인에 포커스를 놓은 상태에서 추론에 사용될 전제인 2번과 4번 라인의 클릭영역을 각각 Right- Click하면 5번 라인의 주석에 2,4 라는 숫자가 파란색으로 나타나고, 2번, 4번 논리식은 연보라색으로 바뀐다. 이제 5번 라인(의 클릭영역)을 더블클릭하면 이 추론의 전제인 2번, 4번 라인의 논리적 귀결인 ¬Castle이 5번 라인 아래에 6번 라인으로 추가되어 그림 9를 얻게 된다. 그러면 이제 이 논리도해를 완성하는 3가지 방법을 보이겠다.

그림 9: 앨리스 추론 2



먼저 그림 10은 그림 9의 1번 라인을 가지치기 하여 두 개의 죽은 가지를 얻은 것이다. 이 논리도해 수형도는 3개의 노드로 구성된다.

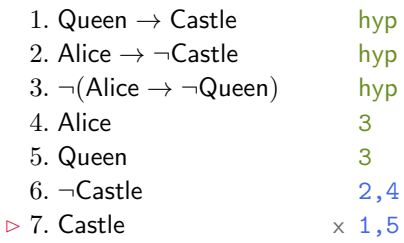
그림 10: 앨리스 추론 3.1



다음, 그림 11은 그림 9에서 1번과 5번을 전제로 하는 추론을 실행하여 Castle이라는 결론을 얻은 것이다. 이 논리도해 수형도는 단 1개의 노드만을 가진다. 주석영역의 전제 번호 1,5에 마우스를 올리면 이 추론에서 사용된 규칙 [예예]를 볼 수 있다.

마지막으로 그림 12는 그림 9에 1번과 6번을 전제로 하는 추론을 적용하여 ¬Queen이라는 결론을 얻은 것이다. 이 논리도해 수형도도 역시 단 1개의 노드만을 가진다. 이 추론에서 사용된 규칙은 [아니아니]이다.

그림 11: 앨리스 추론 3.2



그러면 이제 논리도해에서 사용가능한 추론규칙(*inference rule*)들에 대하여 알아 보자.

우리가 논리도해에서 사용할 수 있는 추론규칙의 개수는 이론적으로는 제한이 없으나 현실적으로 인간이 편리하게 사용하기 위하여는 개수가 너무 작거나 크면 안 될 것이다. 너무 작으

그림 12: 앨리스 추론 3.3

1.	Queen → Castle	hyp
2.	Alice → ¬Castle	hyp
3.	¬(Alice → ¬Queen)	hyp
4.	Alice	3
5.	Queen	3
6.	¬Castle	2,4
▷ 7.	¬Queen	× 1,6

면 규칙들이 충분히 강력하지 못할 것이고, 너무 크면 이들을 다 암기하여 능숙하게 사용하기가 어려울 것이다. Proofmood의 논리도해에서는 추론규칙 8개를 사용하고 있으며 그것들은 다음과 같다.

먼저 고대 그리스 시대부터 논리학자들에게 잘 알려져 있었던 4개의 추론규칙이 있는데 그것은 각각 *Modus ponendo ponens, Modus tollendo tollens, Modus tollendo ponens* 및 *Modus ponendo tollens*이다. 이들 추론규칙의 이름은 라틴어이며 사용된 단어 ponens는 긍정, tollens는 부정을 의미한다. 우리는 이 규칙들의 명칭을 기억하기 쉬운 한글로 바꾸어 각각 순서대로 [예예], [아니아니], [아니예], [예아니]로 부르기로 한다. 그림 13에 이 추론규칙들을 보였다.

그림 13: 4M-규칙

[예예]

1.	A → B	hyp
2.	A	hyp
▷ 3.	B	◎ 1,2

[아니아니]

1.	A → B	hyp
2.	¬B	hyp
▷ 3.	¬A	◎ 1,2

[아니예]

1.	A ∨ B	hyp
2.	¬A	hyp
▷ 3.	B	◎ 1,2

[예아니]

1.	¬(A ∧ B)	hyp
2.	A	hyp
▷ 3.	¬B	◎ 1,2

이들 4개의 규칙은 모두 M으로 시작하므로 *4M-규칙*이라 부르기로 하자. 4M-추론규칙을 적용할 때는 이중부정은 원래의 논리식과 동등하다는 사실을 이용할 수 있다. 예를 들어 그림 14의 추론은 [예아니]에 의하여 타당성이 입증된다.

그림 14: [예아니] 규칙 적용

1.	¬(¬A ∧ ¬B)	hyp
2.	¬B	hyp
▷ 3.	A	◎ 1,2

위의 추론에서 2번 라인의 논리식 ¬B는 1번 라인의 논리식의 한 부분논리식에 ¬를 덧붙이지 않고 그대로 가져와서 사용하였으므로 예이고, 3번 라인의 논리식 A는 1번 라인의 논리식의 한 부분논리식을 가져온 후 ¬를 붙여 사용했으므로(즉시 이중부정을 소거했음) 아니인 것이다.

[예예]와 [아니아니] 규칙은 실제로는 좀 더 강력한 버전으로 구현되어 있으며 그것은 아래와 같다.

1. $A1 \vee A2 \vee A3 \to B$	hyp		1. $A \to B1 \wedge B2 \wedge B3$	hyp	
2. A1	hyp		2. $\neg$B1	hyp	
▷ 3. B	◎ 1,2		▷ 3. $\neg$A	◎ 1,2	

이 규칙은 3개 이상의 합인자나 곱인자를 가진 경우, 그리고 2번 라인에서 첫번째 인자가 아닌 임의의 인자, 예를 들어 $A2, A3, B2, B3$ 등을 사용하는 경우에도 적용된다.

`Proofmood`의 논리도해에서는 4M-추론규칙에 그림 15와 같은 4개의 규칙을 더 추가하여 도합 8개의 추론규칙을 사용한다. 새로 더한 4개의 규칙 중에서 동등소거는, 이 그림에서는 두 개의 규칙만을 보였지만 실은 이 두 규칙 각각에서 2번 라인과 3번 라인의 위치를 서로 교환한 규칙들을 더하여 도합 4개의 규칙으로 이루어진다.

그림 15: 추가 추론 규칙

[논리곱 앞인자소거]

1. $A \wedge B \to C$	hyp
2. A	hyp
3. B	hyp
▷ 4. C	◎ 1,2,3

[논리합 소거]

1. $A \vee B$	hyp
2. $A \to C$	hyp
3. $B \to C$	hyp
▷ 4. C	◎ 1,2,3

[논리합 뒤인자소거]

1. $A \to B \vee C$	hyp
2. $\neg$B	hyp
3. $\neg$C	hyp
▷ 4. $\neg$A	◎ 1,2,3

[동등 소거]

1. $A \leftrightarrow B$	hyp		1. $A \leftrightarrow B$	hyp	
2. A	hyp	,	2. $\neg$A	hyp	
▷ 3. B	◎ 1,2		▷ 3. $\neg$B	◎ 1,2	

이상 설명한 8개의 추론규칙들은 그 선택에 특별한 이론적 근거가 있는 것이 아니고, 다만 경험적으로 유용한 규칙들을 모은 것이다. 추론규칙을 많이 넣을수록 우리의 도해는 강력해지겠지만 대신 암기해야 할 규칙이 많아져서 사용하기에 불편해 진다.

우리가 사용하는 논리도해의 추론규칙의 특징은 전제들로부터 결론이 유일하게 결정된다는 것이다. 그렇지 않은 경우에는 사용자가 직접 결론을 찾아서 입력해야 하므로 효율성이 많이 떨어질 것이다.

5 Remark 논리학에서 가설(hypothesis)과 전제(premise)는 비슷한 의미로 사용된다. 이 책의 논리도해에서는 이 둘을 다음과 같이 구별하여 사용한다.

가설은 사용자가 직접 입력하는 논리식이며 논리도해의 뿌리노드에 위치한다. 주석은 hyp이다. 전제는 터뜨리기, 혹은 추론을 통하여 새로운 논리식을 얻을 때 사용하는 논리식을 뜻한다. 전제는 가설일 수도 있고 그렇지 않을 수도 있다. ⊣

논리도해 수형도에서 모든 단말점(의 마지막 라인)은 ×, ✓, ◎ 중 하나로 표시된다. ×는 이 노드를 말단으로 하는 가지는 만족불가능이라는 뜻이다. ✓는 이 노드를 말단으로 하는 가지는 만족가능이며 더 이상 가지치기를 해 보았자 ×를 얻을 수 없다는 뜻이다. 그 이외의 (진행중인) 단말점은 ◎로 표시된다.

6 Example 지금까지 공부한 지식을 이용하여 아래의 퍼즐을 풀어 보자.

"온도와 기압이 일정하면 비가 오지 않는다. 온도는 일정했다." 라는 가설로부터 "비가 왔다면 [...]." 라는 결론이 도출된다고 할 때 [...]에 들어갈 수 있는 것을 있는 대로 고르시오.

① 온도가 일정했다.

② 온도가 일정하지 않았다.

③ 기압이 일정했다.

④ 기압이 일정하지 않았다.

⑤ 비가 오지 않았다.

이 문제를 기호화 하는 것이 문제풀이의 첫 단계이므로 다음과 같이 놓아 보자.

T : 온도(temperature)가 일정.
P : 기압(pressure)이 일정.
R : 비가(Rain) 온다.
X : "비가 왔다면 [...]." 에서 [...]에 들어갈 것.

이제 이 문제는 아래의 귀결관계

$$T \wedge P \to \neg R,\ T \vDash R \to X \tag{3}$$

가 성립되는 X를 주어진 5개의 예에서 찾는 것이므로, 예에 주어진 5 개의 명제를 나타내는 논리식 T, $\neg T$, P, $\neg P$, $\neg R$를 차례로 X에 대입하면서

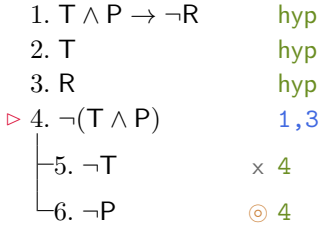
$$T \wedge P \to \neg R,\ T,\ \neg(R \to X)\text{이 만족불가능}$$

이 되는 경우를 진리표나 논리도해를 써서 찾으면 된다.† 이 방법으로 답을 구할 수 있는 것은 확실하나 이 방법은 그리 효율적이지 못하다. 이보다는 (3)과 동등한

$$T \wedge P \to \neg R,\ T,\ R \vDash X \tag{4}$$

가 성립하는 X를 ①~⑤ 중에서 찾는 것이 훨씬 빠르다.

주어진 가설, 즉 (4)의 좌변의 모델집합을 찾기 위하여 다음과 같이 논리도해를 구성하였다. 우리가 찾는 모델집합은 6번 라인을 단말점으로 하는 가지의 모델집합과 일치하며, 이 가지에 속하는 리터럴이 T, R, $\neg P$이므로 결국 모델집합은 한 개의 진리값배정 $TPR = 101$로 이루어짐을 알 수 있다. 그러므로 ①과 ④가 답이 된다.



나머지 3개 ②, ③, ⑤는 (4)의 X가 될 수 없음도 이 모델집합으로부터 쉽게 확인된다. ⊣

† 보조정리 3.(3)을 이용한 것임.

4 피치 증명시스템, 명제논리

피치 증명 시스템에 의한 간단한 증명 하나를 보면서 이 시스템의 사용법과 대체적인 구조를 파악해 보자.

그림 16은 논리적 귀결 관계

$$A, A \to B, A \to C, B \wedge C \to D \vDash D \tag{5}$$

의 가설과 결론을 피치 시스템에 입력한 상태의 화면을 보여 준다. 가설 1,2,3,4 라인들이 주석에 hyp를 사용하는 것은 논리도해와 같다. 가설부와 결론부는 짧은 수평선분으로 구분된다. 라인 5의 타당성 입증(validity verification)을 위한 주석은 아직 입력되지 않았으므로 `Rule ?`로 나타내었다.

그림 16: 피치 증명 시작

	1. A	hyp
	2. A → B	hyp
	3. A → C	hyp
	4. B ∧ C → D	hyp
▷	5. D	Rule ?

식 (5)는 간단한 논리적 귀결이므로 다음과 같은 단계를 거쳐 결론 D를 얻는 과정을 쉽게 이해할 수 있을 것이다.

> 라인 1과 2로부터 B를 얻고, 라인 1과 3으로부터 C를 얻고, 이 두 논리식 B와 C로부터 $B \wedge C$를 얻고, 마지막으로 라인 4와 $B \wedge C$로부터 D를 얻는다.

현재 포커스가 라인 5에 있는데 이 라인 바로 위에 논리식 B, C 등을 넣어 그림 17을 얻어야 하므로 [줄 끼워넣기]를 클릭한다. 이어서 논리식 C와 $B \wedge C$를 차례로 각각 하나의 라인에 넣는다. 포커스가 있는 라인 바로 아래에 새로운 라인을 넣을 때는 [줄 더하기]를 클릭해도 되고 그냥 `Enter` 키를 쳐도 된다.

그림 17: 피치 증명 2

	1. A	hyp
	2. A → B	hyp
	3. A → C	hyp
	4. B ∧ C → D	hyp
▷	5. B	Rule ? 1,2
	6. C	Rule ?
	7. B ∧ C	Rule ?
	8. D	Rule ?

논리도해에서는 가설만 사용자가 직접 입력하고 가설 이외의 라인들은 모두 터뜨리기나 추론에 의하여 자동으로 생성되지만, 피치 시스템에서는 모든 라인을 사용자가 입력해야 한다는 점에서 이 두 시스템은 크게 다르다.

이제 논리식들은 다 넣었으니 각 라인의 입증을 위한 주석을 넣어야 한다. 피치 시스템에서 주석은 전제와 추론규칙으로 이루어져 있다.

전제의 지정은 오른쪽 마우스 클릭으로 한다. 그림 17은 라인 5의 전제인 라인 1과 라인2의 클릭영역을 오른쪽 마우스 클릭한 직후의 화면이다.

그림 18은 라인5의 추론규칙을 넣기 위해서 Rule ?를 클릭하고 마우스를 움직여 [소거 - →]를 선택하고 있는 장면이다. 이 추론규칙의 이름이 [소거 - →]인 이유는 전제에 들어있던 기호 →가 결론에서는 소거되었기 때문이다. (앞으로 몇 개의 추론규칙이 더 나올 것이고 이들에 대한 설명은 나중에 주어질 것이다.) 이 상태에서 클릭을 하면 라인 5의 주석의 입력이 완료된다.

그림 18: 피치 증명 3

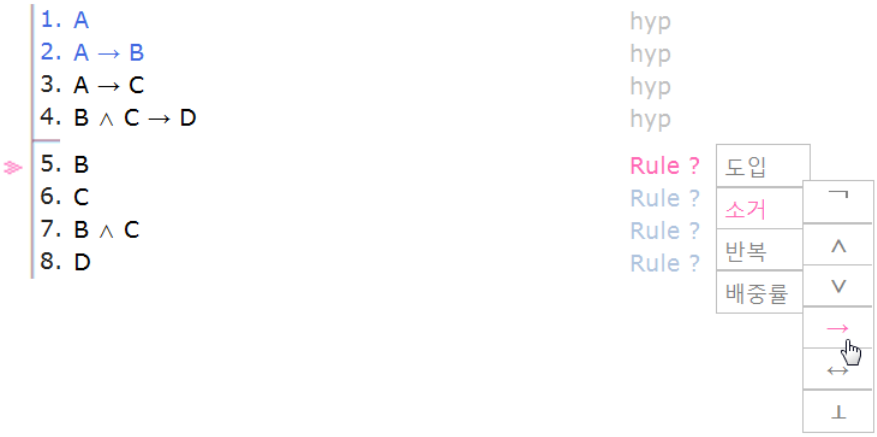
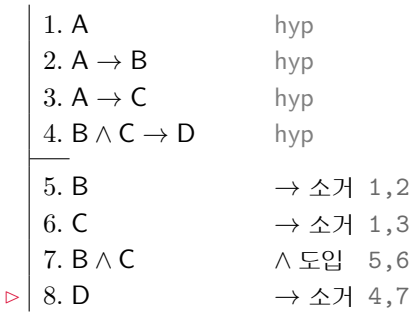


그림 19는 모든 라인에 주석을 넣어 증명이 완료된 상태를 보여 주고 있다. [소거 - →]외에 [도입 - ∧] 규칙이 사용된 것이 보인다.

그림 19: 피치 증명 4, 완료



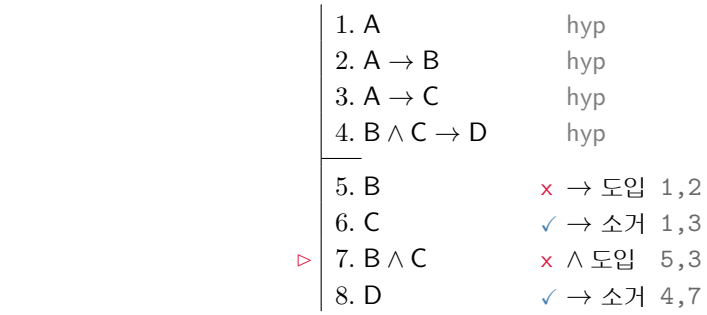
논리도해에서는 주석을 컴퓨터가 알아서 넣어 주었지만, 피치 시스템에서는 주석을 사용자가 직접 손으로 넣으므로 오류가 있을 수 있다. 오류 여부를 확인하기 위하여 [전체 검사]를 클릭한다. 그러면 오류가 있는 주석은 붉은 x 표, 오류가 없는 주석은 파란 ✓ 로 표시된다. 특정한 하나의 줄만 검사하려면 [전체 검사] 대신 그 왼편에 있는 [줄 검사]를, 그 라인에 포커스가 놓인 상태에서 클릭하면 된다.

그림 19에서는 원래 주석에 오류가 없었는데, 그림 20은 일부러, 라인 5에서 추론규칙을 틀리게 바꾸어 넣고, 라인 7에서는 전제를 틀리게 바꾸어 넣어서 [전체 검사]한 결과를 보여준 것이다.

확인 결과 화면에 나타난 ✓ 와 x 표를 지우려면 [전체 검사]를 한 번 더 클릭하여 토글(toggle)하면 된다.

주석을 삭제하려면 추론규칙 부분을 더블클릭하면 된다. 전제만 삭제하려면 그 라인에 포커

그림 20: 피치 증명, 검사



스가 있는 상태에서 전제 번호(들)를 클릭하면 된다.

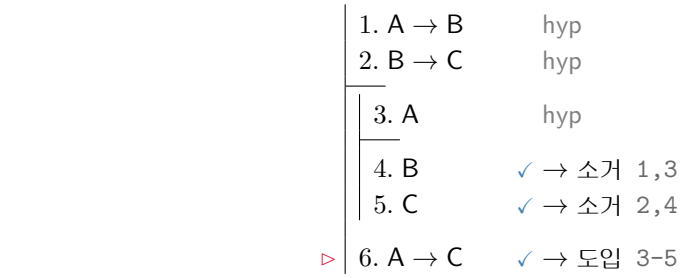
피치 시스템의 가장 큰 특징은 증명 도중에 가설을 자유로이 도입하여 부분증명(*subproof*)을 만들 수 있고, 또 이 부분증명을 후에 얻게 되는 다른 논리식의 전제로 사용할 수 있다는 것이다.

이제 우리가 흔히 사용하는 '→의 추이율'을 피치로 증명해 보자. 그림 21은

$$A \to B, B \to C \vDash A \to C$$

의 피치 증명이다.

그림 21: →의 추이율 증명



결론 $A \to C$를 증명하는 가장 자연스런 방법은 A를 가정하고 이 가정으로부터 C를 증명하는 것이다. 그래서 부분증명을 열고 이 부분증명의 가설인 라인 3에 A를 두었다. 라인 4와 라인 5를 어떻게 얻었는지는 주석을 보면 자명하다.

마지막으로 라인 6는 라인 3에서 라인 5에 이르는 부분증명을 전제로 취한다. 부분증명을 전제로 지정하려면 부분증명에 속하는 아무 라인이나 선택하여 그것의 클릭영역을 Shift-Right-Click 하면 된다. 이 전제를 주석에서 나타낼 때 3,5가 아닌 3–5를 사용하였음에 주목하라. 전자는 라인 3과 라인 5라는 두 논리식을 전제로 취한다는 의미이고 후자는 라인 3(부분증명의 가설)에서 라인 5(부분증명의 결론)에 이르는 부분증명을 전제로 취한다는 의미이다.

이 추론단계에서 사용한 추론규칙은 →-도입이라고 부른다. 이 추론규칙은 일반적으로 아래의 함의가 성립하기 때문에 타당하다. (실은 ⇔가 성립한다.)

$$\Phi, H \vDash C \;\Rightarrow\; \Phi \vDash H \to C \tag{6}$$

여기서 Φ는 논리식의 집합이고 H는 가설, C는 결론이다. 그림 21의 라인 6는

$$\Phi = \{A \to B, B \to C\},\; H \equiv A,\; C \equiv C$$

로 놓고 (6)을 적용하여 얻은 것이다.

부분증명의 가설 및 부분증명의 결론부에 있는 어떤 논리식이든 부분증명의 외부에서는 사용할 수 없다는 것이 중요한데 이 이슈는 피치 증명 전체를 수형도로 보고 노드들의 집합에 적당한 순서구조를 주는 방식으로 처리한다.

수형도의 노드 위치 간의 순서 관계는 다음과 같이 정한다. 논리도해도 수형도로 볼 수 있었는데 피치증명 수형도와는 다음과 같은 중요한 차이가 있다.

- 논리도해 수형도에서는 하나 혹은 그 이상의 라인이 모여 노드를 이루었는데 피치증명에서 하나의 라인은 반드시 하나의 노드를 이룬다.
- 피치증명에서는 라인뿐만 아니라 부분증명도 하나의 노드로 취급한다.

7 Definition 피치 증명의 수형도에서 "노드1은 노드2 이전에 나온다", 혹은 "노드1 < 노드2"의 순서관계는 다음과 같이 정의한다.

(*i*) 같은 부모 노드를 가지는 형제 노드 간에는 라인 번호가 작을수록 이전에 나온 것으로 정한다.

(*ii*) 부모는 자녀보다 이전에 나온 것으로 정한다.

(*iii*) 이 관계는 추이적이다. 즉

$$(\text{노드}1 < \text{노드}2) \text{ and } (\text{노드}2 < \text{노드}3) \;\Rightarrow\; (\text{노드}1 < \text{노드}3)$$

의 함의가 성립한다. ⊣

피치 증명에서 노드 간의 순서관계가 명확하게 정의되었으므로 이제 형식증명시스템에서 가장 중요한 기호인 $\vdash$를 정의할 수 있게 되었다.

8 Definition $\mathcal{F}$를 하나의 피치 증명이라 하자. $\mathcal{F}$의 가설(*hypothesis*)이란 이 증명에 들어있는 어떤 부분 증명의 가설부에 놓인 논리식, 즉 주석이 `hyp`인 논리식을 말한다. $\mathcal{F}$ 내의 한 라인의 가설이란 $\mathcal{F}$의 가설로서 그 라인의 이전에 위치한 것을 말한다. 한 라인의 전제(*premise*)는 그 라인의 주석에 번호가 나타나 있는 라인, 혹은 부분증명을 말한다.

한 라인은 그것의 전제(들)로부터 조금 후에 소개할 12개의 추론규칙 중 어느 하나를 적용하여 그 라인이 얻어질 때 입증(*verify*)되었다고 정의한다.

$\mathcal{F}$ 내에서 입증된 라인은 그것의 전제가 모두 $\mathcal{F}$ 내에서 증명되어 있을 때면이 $\mathcal{F}$ 내에서 증명되었다(*proved in* $\mathcal{F}$)고 재귀적으로 정의한다. 증명된 라인에 논리식 A가 있고, 그 라인의 모든 가설들의 집합을 Φ라 했을 때 A는 Φ로부터 증명된다(A *is proved from* Φ), 혹은 Φ는 A를 증명한다(Φ *proves* A)고 말하고 기호로는

$$\Phi \vdash A$$

로 나타내고 "Φ 아 A", 혹은 "Φ proves A"로 읽는다. $\vdash$는 증명관계(*proof relation*)라고 한다.

피치 증명(Fitch Proof)은 피치 도출과정(*Fitch derivation*)이라고도 한다. ⊣

9 Definition 어떤 형식증명시스템에서

$$\Phi \vdash A \;\Rightarrow\; \Phi \vDash A \tag{7}$$

가 성립할 때 이 증명시스템을 건전(*sound*)하다고 말하고, 역으로

$$\Phi \vDash A \;\Rightarrow\; \Phi \vdash A \tag{8}$$

가 성립할 때 이 증명시스템을 완전(*complete*)하다고 말한다. ⊣

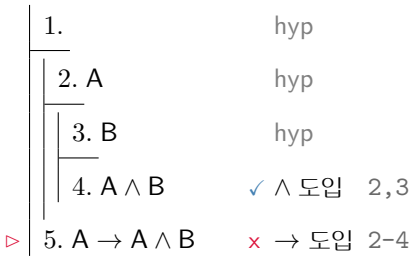
피치 시스템은 건전하고 또한 완전하다.
위의 (7)과 (8)에 의하여 이제 20쪽의 (6)은 다음과 같이 쓸 수 있다.

$$\Phi, H \vdash C \;\Rightarrow\; \Phi \vdash H \to C \tag{9}$$

(9)는 연역정리(*deduction theorem*)라고 하는, 간단하지만 대단히 중요한 논리학의 정리이다.

10 Exercise 아래의 증명에서 5번 라인의 주석을 보라.

그림 22: → 도입의 잘못된 적용



5번 라인의 전제는 부분증명 2–4이며, 이 부분증명의 가설은 A이고 마지막 라인은 $A \wedge B$이다. 그래서 → 도입에 의하여 $A \to A \wedge B$를 얻은 것이다. 입증에 실패한 이유가 무엇일까? ⊣

지금까지 우리가 사용한 추론규칙은 → 소거와 ∧ 도입 및 → 도입 등 3개 뿐이었는데, 피치 시스템에서는 모든 6 개의 결합자 $\neg, \wedge, \vee, \to, \leftrightarrow, \bot$ 각각에 대하여 도입과 소거 규칙이 있어 총 12개의 추론규칙을 사용한다. 피치에서 사용하는 12개의 추론규칙은 표 5와 같다. 이 표에서 $A \vdash B$는 A를 가설로 하고 B를 결론으로 하는 부분증명을 뜻한다.

이 표의 각 추론규칙에서 수평선 위에 나타나는 논리식과 부분증명은 전제(premise), 수평선 아래에 나타나는 논리식은 결론(conclusion)이며 전제 노드들은 모두 결론 노드 이전에 나와야 한다. 단, 전제 노드 간의 순서관계는 어떻게 되든 상관없다.

$$\frac{\begin{array}{c}A\\ \neg A\end{array}}{\bot}\ \bot \text{ 도입} \qquad \frac{\bot}{A}\ \bot \text{ 소거}$$

이상의 12개의 추론규칙들이, 예를 들어 ∧ 도입 규칙 $\{A, B\} \vDash A \wedge B$이 건전함(타당함)은 진리표에 의해서 확인된다. 여기서 문제가 될 수 있는 것은 전제에 부분증명이 사용된 경우들인데, 이 경우에는

$$A \vdash B \;\Rightarrow\; A \vDash B \;\Leftrightarrow\; \vDash A \to B$$

로부터 타당성을 보장받을 수 있다. 위에서 오른쪽의 ⇔는 이미 몇 번 언급한 보조정리 3.(2)에 의하여 성립하며, 왼쪽의 ⇒는 귀납가설(induction hypothesis)이다. 귀납법을 사용할 수 있음은 피치 증명의 재귀적 구조에 의한다.

표 5: Fitch 증명시스템의 추론 규칙

$$\frac{A \vdash \bot}{\neg A}\ \neg \text{ 도입} \qquad \frac{\neg A \vdash \bot}{A}\ \neg \text{ 소거}$$

$$\frac{\begin{array}{c}A\\B\end{array}}{A \wedge B}\ \wedge \text{ 도입} \qquad \frac{A \wedge B}{A},\ \frac{A \wedge B}{B}\ \wedge \text{ 소거}$$

$$\frac{A}{A \vee B},\ \frac{B}{A \vee B}\ \vee \text{ 도입} \qquad \frac{\begin{array}{c}A \vee B\\A \vdash C\\B \vdash C\end{array}}{C}\ \vee \text{ 소거}$$

$$\frac{A \vdash B}{A \to B}\ \to \text{ 도입} \qquad \frac{\begin{array}{c}A \to B\\A\end{array}}{B}\ \to \text{ 소거}$$

$$\frac{\begin{array}{c}A \vdash B\\B \vdash A\end{array}}{A \leftrightarrow B}\ \leftrightarrow \text{ 도입} \qquad \frac{A \leftrightarrow B}{A},\ \frac{A \leftrightarrow B}{B}\ \leftrightarrow \text{ 소거}$$

`Proofmood`에는 증명의 편의와 효율을 위하여 아래와 같은 2개의 규칙을 더 추가 구현되어 있다.

$$\frac{A}{A}\ \text{반복} \qquad \frac{}{A \vee \neg A}\ \text{배중률}$$

이 두 규칙이 타당함은 자명하다.

아래의 귀결 관계를 피치를 사용하여 증명해 보자.[†]

$$A \to B \vDash A \wedge C \to B \wedge C \tag{10}$$

먼저 다음과 같이 가설과 결론을 넣는다.

```
  | 1. A → B          hyp
  |---
▷ | 2. A ∧ C → B ∧ C  Rule ?
```

결론 $A \wedge C \to B \wedge C$의 주결합자가 $\to$이므로 $\to$ 도입을 사용해야 할 것이다. 포커스가 2번 라인에 있는 상태에서 [부분증명 끼워넣기]를 클릭하여 다음과 같이 만든다.

```
  | 1. A → B          hyp
  |---
  |  | 2. A ∧ C       hyp
  |  |---
▷ |  | 3. B ∧ C       Rule ?
  |
  | 4. A ∧ C → B ∧ C  Rule ?
```

[†]피치를 사용하여 $\Phi \vDash A$를 증명한다는 것은 $\Phi \vdash A$의 도출과정을 만들어 낸다는 뜻이다.

3번 라인의 주결합자가 $\wedge$이므로 $\wedge$ 도입을 사용하는 것이 당연하다. 그러므로 3번 라인에 포커스가 놓인 상태에서 [줄 끼워넣기]를 클릭한 후 두 라인을 넣어 다음과 같이 만든다.

	1. A → B	hyp
	2. A ∧ C	hyp
	3. B	Rule ?
▷	4. C	Rule ?
	5. B ∧ C	Rule ?
	6. A ∧ C → B ∧ C	Rule ?

C(라인 4)는 $A \wedge C$(라인 2)에 $\wedge$ 소거를 적용하면 나올 것이다. B(라인 3)는 $A \to B$(라인 1)와 A에 $\to$ 소거를 적용하여 얻으면 될 것인데, A는 아직 도출과정 내에 없지만 라인 2에 $\wedge$ 소거를 적용하면 쉽게 얻어진다. 그러므로 결국 다음과 같은 증명을 완성하게 된다.

	1. A → B	hyp
	2. A ∧ C	hyp
	3. A	✓ ∧ 소거 2
	4. B	✓ → 소거 1,3
	5. C	✓ ∧ 소거 2
	6. B ∧ C	✓ ∧ 도입 4,5
▷	7. A ∧ C → B ∧ C	✓ → 도입 2-6

이 증명을 구성하는 과정에서 피치 커맨드들을 어느 정도 익숙하게 사용하게 되었을 것이다. 단, 전제 지정을 하다가 실수하여 다른 라인 번호가 전제로 주석에 지정되었을 때는 그 라인을 다시 한 번 Right Click 하면 그 번호가 주석에서 토글되어 사라진다는 것을 알고 있어야 할 것이다.

그런데 피치 커맨드들 중에 [부분증명 끝내기]와 [첫 결론 끼워넣기]는 어떤 상황에서 사용하는 것인지 알기 힘들 수 있다. 전자는 다음 그림의 왼쪽 상황에서 오른쪽 상황을 만들 때 쓰인다.

	1. A → B	hyp
	2. A ∧ C	hyp
▷	3. B ∧ C	Rule ?

	1. A → B	hyp
	2. A ∧ C	hyp
	3. B ∧ C	Rule ?
▷	4.	Rule ?

후자([첫 결론 끼워넣기])는 다음 그림의 왼쪽 상황에서 오른쪽 상황을 만들 때 쓰인다.

▷	1. A → B	hyp
	2. A ∧ C	hyp
	3. B ∧ C	Rule ?

	1. A → B	hyp
▷	2.	Rule ?
	3. A ∧ C	hyp
	4. B ∧ C	Rule ?

전자와 후자 모두 포커스의 위치에 각별히 주목해야 할 것이다.

이번에는 아래의 귀결 관계를 피치를 사용하여 증명해 보자.

$$A \to B \vDash A \vee C \to B \vee C \tag{11}$$

(11)은 (10)과 비슷하므로 피치 증명도 아까와 같이 시작하면 된다.

	1. A → B	hyp
	2. A ∨ C	hyp
	3. B ∨ C	Rule ?
▷	4. A ∨ C → B ∨ C	✓ → 도입 2-3

그런데 여기서 $B \vee C$(라인 3)을 증명하기 위하여 이 논리식의 주결합자인 $\vee$에 주목하여 $\vee$ 도입을 사용하려 한다면 잘 되지 않는다. 왜냐하면 $\vee$ 도입을 사용하여 $B \vee C$를 얻으려면 위하여는 그 전에 B, 혹은 C를 얻어야 하는데 이들은 $B \vee C$ 보다 더 '강한' 논리식이기 때문이다.

여기서 발상을 전환하여 $A \vee C$(라인 2)를 가설로 이용하여 $B \vee C$(라인 3)을 얻으려면 어떤 추론규칙을 써야 하겠는지를 생각해 보라. 라인 2의 주결합자가 $\vee$이므로 $\vee$ 소거를 사용해야 한다는 것을 쉽게 떠 올릴 수 있을 것이다. 그러므로 일단 다음과 같은 그림을 얻는다.

	1. A → B	hyp
	2. A ∨ C	hyp
	3. A	hyp
	4. B ∨ C	Rule ?
	5. C	hyp
	6. B ∨ C	Rule ?
▷	7. B ∨ C	✓ ∨ 소거 2,3-4,5-6
	8. A ∨ C → B ∨ C	✓ → 도입 2-7

(라인 6)을 (라인 5)에서 얻기는 쉽다. $\vee$ 도입을 쓰면 된다. (라인 4)를 (라인 3)으로부터 얻는 것은 약간 더 까다롭지만 (라인 1)과 (라인 3)에 $\rightarrow$ 소거를 적용하여 B를 얻은 다음 다시 $\vee$ 도입을 써서 (라인 4)를 얻는 것을 어렵지 않게 마음 속에 그릴 수 있을 것이다. 이하는 생략한다.

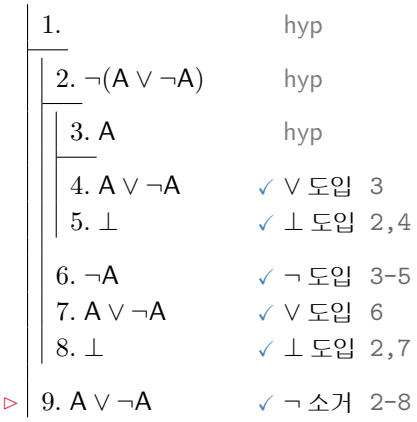
앞의 증명은 부분증명 내에 다시 부분증명들이 들어 있는 구조를 가진다. 이 증명의 높이는 3단이다. 복잡한 피치 증명은 높이가 4단, 혹은 5단까지 올라가기도 하나 그 이상으로 올라가는 경우는 거의 없다.

높이가 높은, 복잡한 피치 증명에서 현재 포커스 된 라인이 어떤 부분증명에 속해있는지를 파악하는 것은 그 부분증명의 왼편에 있는 ($\vdash$ 모양의) 기준선이 핑크색으로 강조되어 있으므로 아주 수월하다. 예를 들어 부분증명을 [부분증명 삭제] 커맨드로 삭제할 경우 기준선의 색깔만 주목하면 실수로 엉뚱한 다른 부분증명을 삭제해 버릴 위험은 크지 않을 것이다.

배중률(*law of excluded middle, LEM*), 즉 $\vdash A \vee \neg A$는 피치 시스템의 12개 기본 추론규칙들만 사용하여 증명하기가 상당히 까다롭다. 그림 23에 풀이를 주었으니 깊이 음미해 보기 바란다. 이 배중률 증명에서 사용된 기법은 드모르강의 법칙의 증명 등 여러 곳에서 유용하게 쓰인다.

배중률은 원래 피치의 추론규칙에는 들어 있지 않다. 그러나 이 규칙은 `Proofmood`에 구현해 두었다. 그러므로 일단 배중률 추론규칙을 써서 피치 증명을 얻은 후에 그 배중률 라인을 그림 23을 이용하여 다시 직접 증명하면 배중률 규칙을 쓰지 않은 증명을 얻게 된다.

그림 23: 배중률



11 Exercise 반복도 배중률과 같이 원래의 피치 시스템에는 들어 있지 않지만 피치 시스템에서 증명이 가능하다. 반복의 증명을 찾아 보라. ⊣

5 피치 증명시스템, 1 계논리

5.1 왜 1 계논리인가 ?

대단히 유명한 추론 하나를 아래에 보였다.

$$\begin{aligned} (H_1)&\quad \text{모든 인간은 죽는다.}\\ (H_2)&\quad \text{소크라테스는 인간이다.}\\ (C)&\quad \text{그러므로 소크라테스는 죽는다.} \end{aligned} \tag{12}$$

이 추론이 타당한 이유를 흔히 다음과 같이 설명한다.

> 인간을 h, 죽음을 m, 소크라테스를 s 로 나타내고 이 추론을 아래와 같이 기호화 한다.
>
> $$\begin{aligned} (H_1)&\quad h \to m\\ (H_2)&\quad s \to h\\ (C)&\quad s \to m \end{aligned}$$
>
> 그리고 논리적 귀결관계 $h \to m,\ s \to h \vDash s \to m$ 이 성립함을 진리표나 논리도해, 혹은 피치 시스템 등의 적당한 방법으로 보인다.

이 추론은 실상 첫 시작인 기호화 단계에서 이미 옳지 않다. 논리식은 명제를 기호화 한 것이고 명제는 진위를 분별할 수 있는 문장이라고 하였다. 진위를 분별할 수 있는 문장은 평서문이어야 할 것이고 이러한 평서문에서 주어와 술어는 생략할 수 없는, 반드시 존재해야 하는 구성요소이다. "모든 인간은 죽는다." 라는 명제에서 주어는 "인간" 이고 술어는 "죽는다" 가 되어야 할 것이며 "모든" 은 주어를 수식하는 한정사(*determiner*) 이다. 이 명제를 단순히 "인간 $\to$ 죽음" 으로 기호화 하면 다음과 같은 문제에 봉착한다.

결합자는 기존의 논리식(들)을 결합하여 새로운 논리식을 만들어 주는 것이므로 결합자의 앞인자와 뒤인자는 논리식이어야 한다. 따라서 앞인자 "인간"과 뒤인자 "죽음"이 각각 명제이어야 할 것인데, 앞서 말했듯이 명제는 주어와 술어를 가져야 한다. 그러나 "인간"이나 "죽음"이라는 단어들을 각각 주어와 술어로 분해할 수 없음은 명백하다.

그렇다면 추론 (12)를 기호화 하려면 어떻게 해야 할까?

"죽는다"는 술어(predicate)이고 "인간"은 주어이니 "인간은 죽는다"를 $m(h)$로 나타내는 것도 하나의 방법이 되겠다. 그런데 이러한 기호화에는 두 개의 문제점이 있다. 하나는 "모든"이라는 한정사는 어디로 갔느냐는 것이다. 다른 하나는 인간은 인류 전체를 나타내는 단어이지 개체(individual)가 아니므로 주어이기 보다는 술어에 가깝게 보인다는 것이다. 실제로 소전제 (H_2): "소크라테스는 인간이다."에서 "인간"은 술어로 기능하고 있다.

"죽는다"와 "인간임"을 각각 술어 m과 h로 보고 "소크라테스"를 주어 역할을 하는 개체 s로 본다면 소전제 (H_2)는 $h(s)$, 결론 (C)는 $m(s)$로 쓰면 될 것이다. 문제는 대전제 (H_1)인데 이것은 변수의 개념을 도입하여

$$(H_1)\text{: 모든 } x \text{에 대해서, } x \text{가 인간이면 } x \text{는 죽는다.}$$

로 쓰면 될 것이다. 모든 x에 대해서를 기호 $\forall x$로 나타내기로 하면 이제 (H_1)은

$$\forall x(h(x) \to m(x)) \tag{13}$$

로 기호화 하여 나타낼 수 있다.

(13)의 x는 임의의 개체일 수 있으므로 여기에 소크라테스 s를 대입하면

$$h(s) \to m(s) \tag{14}$$

을 얻는다.

우리는 (14)에 더하여 소전제 $h(s)$를 가지고 있으므로 논리적 귀결관계

$$h(s) \to m(s),\ h(s) \vDash m(s)$$

로부터 우리가 원하던 결론 $m(s)$를 얻는다. 이상의 논의를 피치 시스템에 구현하면 그림 24와 같다.

그림 24: 소크라테스의 죽음: 피치 증명 1

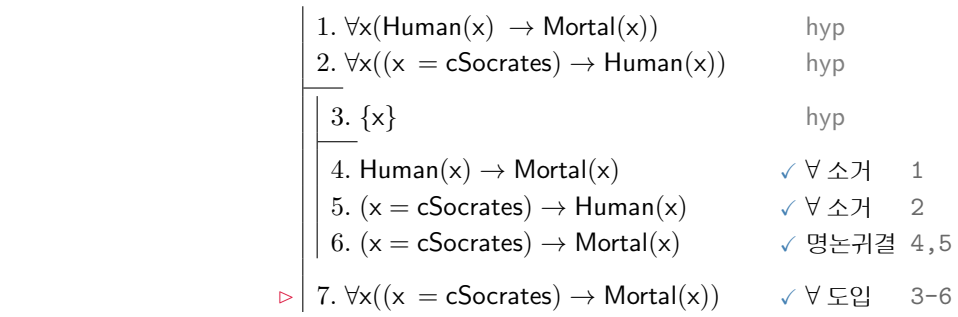
	1. ∀x(Human(x) → Mortal(x))	hyp
	2. Human(cSocrates)	hyp
	3. Human(cSocrates) → Mortal(cSocrates)	✓ ∀ 소거 1
▷	4. Mortal(cSocrates)	✓ 명논귀결 2,3

여기서 술어 "인간임"과 "죽는다"는 각각 1항-술어기호 `Human(_)`과 `Mortal(_)`로, 그리고 "소크라테스"라는 개체는 상수기호 `cScorates`로 나타내었다.

이제까지 추론 (12)를 1계논리로 기호화 하여 증명하는 과정을 보였는데 이것이 유일한 방법인 것은 아니다. 추론 (12)의 또 하나의 피치 증명을 그림 25에 보였다.

그림 24와 그림 25의 주석에 나타난 추론규칙에 대해서는 나중에 상세하게 설명할 것이다.

그림 25: 소크라테스의 죽음: 피치 증명 2



명제논리는 명제를 기호화 하여 엄격한 추론을 하는 데 있어 가장 기본이 되는 중요한 논리 체계이다. 그러나 방금 소크라테스 추론의 예에서 보았듯이 명제논리만 가지고는 명제를 충분히 세밀하게 기호화 하기에 부족하다.

1 계논리(first-order logic)는 명제의 기본적인 구성요소인 술어와 한정사를 비롯한 다양한 구문요소들을 추가하여 더욱 풍부한 형식언어 체계를 제공한다. 물론 1 계논리도 인간의 모든 사상과 감정을 나타내기에는 대단히 부족하지만 수학을 표현하는 데는 충분한 것으로 널리 인식되고 있다.

5.2 1 계논리식의 구문과 의미

명제논리에서는 명제문자와 결합자, 그리고 괄호만 가지고 모든 논리식을 만들었는데, 1 계논리에서는 더 다양한 종류의 기호들을 사용한다.

$x+3=5$ 와 같은 방정식을 위해서는 변수 x 와 등호(*equality*) $=$ 가 필요하다. x 는 그것의 값(value)으로 자연수의 집합 $\mathbb{N}$ 의 원소인 개체(*individual*)를 취하게 되므로 개체변수(*individual variable*)라고 부른다.†

$x+x<x\cdot x+1$ 와 같은 부등식을 위해서는 술어기호(*predicate symbol*) $<$ 가 필요하다.

그리고 "모든 x 에 대해서 $x+1=1+x$ 이다." 와 같은 항등식을 간단히

$$\forall x(x+1=1+x)$$

로 나타내기 위하여 전칭한정기호(*universal quantifier*) $\forall$ 이 필요하다.

"방정식 $x+6=x\cdot x$ 의 해가 존재한다." 라는 명제는 "어떤 x 가 존재하여 $x+6=x\cdot x$ 가 성립한다." 로 쓸 수 있을 것인데 이를 간단하게

$$\exists x(x+6=x\cdot x)$$

로 나타내기 위하여 존재한정기호(*existential quantifier*) $\exists$ 가 필요하다.

마지막으로 이미 얻은 논리식을 결합하여 새로운 논리식을 만들어 내는 데 사용되는 결합자(*connective*)가 필요함은 명제논리의 경우와 같다.

그리고 $f(x,y)=x\cdot x+y$ 같은 표현에서 괄호 $(,)$ 와 쉼표 $,$ 가 사용되었는데, 이들은 어떤 수학적·논리적인 의미를 가지고 있는 기호가 아니라 읽기의 편의나 다른 기호들의 위치관계 지정을 위한 보조기호(*auxiliary symbol*)이다.

† 개체변수는 흔히 변수기호(*variable symbol*)라고도 한다.

12 Definition 1계논리의 알파벳(*alphabet*), 즉 1계논리식에서 사용되는 기호 전체의 집합 Σ는 다음과 같이 정의된다.

$$\Sigma = \text{술어기호} \cup \text{함수기호} \cup \text{상수기호} \cup \text{개체변수} \cup \text{한정기호} \cup \text{등호} \cup \text{결합자} \cup \text{보조기호}$$

술어기호: $\{A, B, P, Q, <, \leq, \approx, \ldots\}$

함수기호: $\{f, g, +, -, \cdot, \div \ldots\}$

상수기호: $\{c, d, 0, 1, \ldots\}$

개체변수: $\{x, y, i, j, \ldots\}$

한정기호: $\{\forall, \exists\}$

등호: $\{=\}$

결합자: $\{\neg, \wedge, \vee, \rightarrow, \leftrightarrow, \perp\}$

보조기호: $\{(,), ,\}$

우리는 여기서 1계언어 알파벳의 클래스(class)를 정의한 것이다. 술어기호와 함수기호 및 상수기호는 언어에 따라 달라질 수 있으며 이들을 논리외적 기호(*extralogical symbol*)라고 부른다. 결합자, 개체변수, 한정기호와 등호는 모든 1계언어에 공통이며 이들을 논리적 기호(*logical symbol*)라고 부른다. ⊣

명제논리에서는 기호들을 결합하여 논리식을 만든다. 1계논리에서는 논리식 외에도 논리항이라는 개념이 있다. 예를 들어 $x + y$ 같은 표현은 논리항이다. 이것은 어떤 값을 의미하고자 하는데 그 값은 진리값이 아니다. $x + y = 2$와 같은 표현은 논리식이다. 이 논리식에서는 술어기호로 $=$을 사용하고 있다.

13 Definition 1계논리의 논리항(*term*)은 다음과 같이 재귀적으로 정의되는 문자열이다.

(*i*) 상수기호와 개체변수는 그 자체로 논리항이다. 이런 논리항을 원자논리항(*atomic term*)이라고 부른다.

(*ii*) f가 앞에쓰기 n항-함수기호이고 $t_1, \ldots, t_n$이 논리항이면 $f(t_1, \ldots, t_n)$도 논리항이다. f가 가운데쓰기 함수기호일 때는 $(t_1\ f\ t_2)$가 논리항이며, f가 뒤에쓰기 함수기호일 때는 $(t_1\ f)$가 논리항이다. 이렇게 얻은 논리항들을 합성논리항(*compound term*)이라고 부른다. 합성논리항 $t \stackrel{\text{def}}{\equiv} f(t_1, \ldots, t_n)$을 구성하는 t_i들을 t의 부분논리항(*subterm*)이라 한다.†

(*iii*) 모든 논리항들은 위의 두 규칙들을 반복적용하여 얻어진다. ⊣

14 Definition 1계논리의 논리식(*formula*)은 다음과 같이 재귀적으로 정의되는 문자열이다.

(*i*) P가 n항-술어기호이고 $t_1, \ldots, t_n$이 논리항이면 $P(t_1, \ldots, t_n)$는 논리식이다. 단, $n = 0$일 때는 P는 그 자체로 논리식이다.

이렇게 얻어지는 논리식을 원자논리식(*atomic formula*)부른다. P가 가운데쓰기, 혹은

†부분논리항 관계는 추이적이 되도록 확장시켜 적용한다.

뒤에쓰기 술어기호일 때의 원자논리식은 각각 $(t_1\,P\,t_2)$ 및 $(t_1\,P)$와 같이 정의된다. 등호 '='는 가운데쓰기 2항-기호이다.

(ii) 논리식들은 명제논리에서와 같은 방법으로 결합자로 결합되어 새로운 논리식을 이룬다.

(iii) x가 개체변수이고 α가 논리식이면 $\forall x\,\alpha$와 $\exists x\,\alpha$는 논리식이다.

(iv) 모든 논리식들은 위의 3 규칙들을 반복적용하여 얻어진다. 원자논리식이 아닌 논리식을 합성논리식(*compound formula*)이라고 부른다.

(v) 부분논리식(*subformula*)에 대한 당연한 정의는 생략한다. ⊣

명제논리에서 괄호를 생략하기 위하여 결합자 간의 우선순위를 사용하였듯이 1계논리에서도 결합자, 한정사, 함수기호, 술어기호 간의 우선순위(*priority*)를 사용한다.

- 함수기호와 술어기호가 앞에쓰기로 사용될 때는 괄호와 함께 사용되므로 가장 높은 우선순위를 가진다. 다만 '$-$'가 1항-함수기호로 쓰일 때는 뒤에쓰기 함수기호보다 낮은 우선순위를 가진다. 예를 들어 $f(x,y)'$은 $(f(x,y))'$을 뜻하며 $-(x+y)'$은 $-((x+y)')$을 뜻한다.
- 뒤에쓰기 함수기호는 가운데쓰기 함수기호보다 높은 우선순위를 가진다.
- 가운데쓰기 함수기호는 가운데쓰기 술어기호보다 높은 우선순위를 가진다. 예를 들어 $x+y<z$는 $(x+y)<z$를 뜻한다. (실은 $x+(y<z)$는 문법에 어긋나므로 우선순위 논의의 대상이 되지 못한다.)
- 가운데쓰기 술어기호는 결합자나 한정사보다 높은 우선순위를 가진다. 예를 들어 $x<y\to x\le y$는 $(x<y)\to(x\le y)$를 뜻한다. (실은 $x<(y\to x)\le y$ 등은 문법에 어긋나므로 우선순위 논의의 대상이 되지 못한다.)
- 한정사와 결합자들은 다음과 같은 우선순위를 가진다.

  (1) $\forall x, \exists y, \neg$: 높음

  (2) $\wedge, \vee$: 중간

  (3) $\to, \leftrightarrow$: 낮음

  예를 들어 $\forall x\,A(x)\wedge B(x)$는 $\forall x\,(A(x)\wedge B(x))$가 아니라 $(\forall x\,A(x))\wedge B(x)$를 뜻한다.

1계논리에서 자유변수(free variable)와 묶인변수(bound variable)의 개념은 대단히 중요하다. 아래의 논리식을 생각해 보자.

$$(x\ge 2)\wedge\forall y\forall z\big((y\cdot z=x)\to(y=1\ \vee\ z=1)\big) \tag{15}$$

(15)는 정수의 영역에서 "x는 소수이다." 라는 '명제'를 1계논리식으로 나타낸 것이다. 그런데 무릇 명제란 진위를 판별할 수 있는 것이라 하였다. 그렇다면 (15)는 참인가 거짓인가?

이 물음에 대한 대답은 간단하다. (15)는 x가 소수일 때는 참이고 소수가 아닐 때는 거짓이다. 즉 (15)의 진위는 x의 값이 어떻게 주어지느냐에 따라 달라진다.

(15)에 나타나는 변수에는 x 외에도 y와 z가 있다. 그런데 (15)의 진위는 y나 z의 값이 어떻게 주어지느냐에 따라 달라지지 않는다. 좀 더 정확히 말하자면 y나 z에 값을 준다는 것 자체가 의미가 없다.

x와 같이 한정기호에 묶여 있지 않아 값을 자유로이 배정(assign)할 수 있는 변수를 자유변수라 하고 y나 z와 같이 한정기호에 묶여 있는 변수를 묶인변수라 한다.

15 Definition 논리식 α의 자유변수(*free variable*)들의 집합 $\mathrm{FV}(\alpha)$는 다음과 같이 재귀적으로 정의한다.

(*i*) α가 원자논리식일 때는 $\mathrm{FV}(\alpha)$는 α에 나타나는 모든 변수들의 집합이다.

(*ii*) $\alpha \equiv \neg\beta$일 때는 $\mathrm{FV}(\alpha) = \mathrm{FV}(\beta)$이다.

(*iii*) $\alpha \equiv \beta \wedge \gamma$, $\beta \vee \gamma$, $\beta \to \gamma$, 혹은 $\beta \leftrightarrow \gamma$일 때는 $\mathrm{FV}(\alpha) = \mathrm{FV}(\beta) \cup \mathrm{FV}(\gamma)$이다.

(*iv*) $\mathrm{FV}(\bot) = \varnothing$이다.

(*v*) $\alpha \equiv \forall x\,\beta$, 혹은 $\alpha \equiv \exists x\,\beta$일 때는 $\mathrm{FV}(\alpha) = \mathrm{FV}(\beta) - \{x\}$이다.

변수 x가 $\mathrm{FV}(\alpha)$의 원소일 때면이 x는 α에서 자유변수로 나타난다고 하고, x가 α에 나타나되 자유로이 나타나지 않을 때 x는 α의 묶인변수(*bound variable*)이다, 혹은 α에서 묶인변수로 나타난다고 한다. 자유변수를 하나도 갖지 않은 논리식을 1계문장(*first-order sentence*)이라고 한다. ⊣

하나의 변수 x가 하나의 논리식 내에서 자유변수인 동시에 묶인변수일 수 있다. 예를 들어 논리식 $\forall x A(x) \to A(y) \wedge \forall z B(z,x)$에서 x는 자유변수인 동시에 묶인변수이고, y는 자유변수이며 z는 묶인변수이다.

피치 시스템에서 1계논리의 기호를 사용할 때는 그것이 술어기호, 함수기호, 결합자 등 여러 종류 중 어느 것인지를 지정해 주어야 하는 것이 원칙이다. 명제논리에서는 기호가 명제문자와 결합자의 두 종류밖에 없었으므로 특별히 언급하지 않았는데, 실은 명제문자는 영문 소문자 혹은 대문자로 시작하고 이어서 영문 소문자, 대문자, 숫자(0,1,...,9) 및 밑줄(underscore)을 0개 이상 덧붙여 만든 문자열이며, 결합자는 $\neg, \wedge, \vee, \to, \leftrightarrow$ 및 $\bot$ 중 어느 하나로 이루어진 길이 1의 문자열인 것으로 정하여 사용해왔다.

1계논리에는 사용하는 기호의 종류가 많으므로 명제논리 때보다는 조금 더 상세하게 기술하도록 하겠다.

- 술어기호: 대문자로 시작하고 이어서 소문자/대문자, 숫자 및 밑줄을 0개 이상 덧붙인다. 혹은 특수기호 $=$, $\neq$, $<$, $\leq$, $>$, $\geq$, $\equiv$, $\approx$, $\in$, $\notin$ 및 $\subset$을 가운데쓰기 2항기호로 사용한다. 가운데쓰기 술어기호들의 우선순위는 모두 동일하다.

  $x = y < z \in w$와 같은 표현은 허용되며 이는 $(x = y) \wedge (y < z) \wedge (z \in w)$를 뜻한다. 이런 표현을 연속 2항술어기호문이라고 한다. 연속 2항술어기호문 중 특별히 $t_1 = t_2 = t_3$ 형태의 논리식을 3등식이라고 한다.

- 함수기호: `f`, `g`, `h` 중 어느 하나로 시작하고 이어서 소문자/대문자, 숫자 및 밑줄을 0개 이상 덧붙인다. 혹은 영문 소문자만으로 이루어진 길이 2 이상의 문자열이다. 혹은 특수기호 $+$, $-$, $/$, $\cdot$, $\cap$, $\cup$을 가운데쓰기 2항기호로 사용한다. 또한 $'$는 뒤에쓰기 1항기호로

사용한다. $\cdot$와 $/$의 우선순위는 나머지 가운데쓰기 함수기호들의 우선순위보다 높다.

우선순위가 같은 가운데쓰기 2항-함수기호가 연달아 쓰이면 왼쪽결합(*left association*)이 사용된 것으로 간주한다. 예를 들어 $x \cdot y/z$는 $(x \cdot y)/z$를 뜻하고, $x \cdot y \cdot z$는 $(x \cdot y) \cdot z$를 뜻한다.

- 상수기호: (1) `a`, `b`, `c`, `d`, `e` 중 어느 하나로 시작하고 ① 대문자, 혹은 밑줄을 1개 사용 후 소문자/대문자, 숫자 및 밑줄을 0개 이상 덧붙이거나, 혹은 ② 숫자만 0개 이상 덧붙인다. 혹은 (2) 숫자만으로 이루어진 길이 1 이상의 문자열이다. 또한 특수기호 $\varnothing$도 상수기호로 사용할 수 있다.

- 개체변수: `i`~`m`, `u`~`z`로 시작하고 숫자를 0개 이상 덧붙인다.

5.3 명제논리적 귀결

명제논리의 추론규칙은 각 결합자마다 도입규칙과 소거규칙이 있어 모두 12개로 구성됨을 배웠다. 1계논리는 명제논리를 확장한 논리체계라고 볼 수 있으므로 명제논리의 추론규칙 12개 모두를 사용하고, 여기에 추가하여 새로운 요소인 한정기호(전칭과 존재의 2개)와 등호에 대하여 각각 도입규칙과 소거규칙이 필요할 것으로 짐작되며, 실제로 이 6개의 추론규칙을 추가하면 완전한(complete) 1계논리 증명시스템이 이루어지게 된다.

그렇다면 1계논리의 추론규칙의 개수는 12개 + 6개 = 18개가 되는데 이를 모두 그대로 사용하는 것은 썩 효율적이지 못하다. 우리는 증명의 효율성을 위하여 명제논리에서의 추론규칙 12개를 단 하나의 규칙, 명논귀결(명제논리적 귀결, *tautological consequence*)로 대체하여 사용한다. 아래에 명논귀결의 사용 예를 보였다.

```
⋮
2. ∀x A(x) → (∃x B(x) ↔ C)        Rule ?
3. ¬C                              Rule ?
4. ∃x B(x)                         Rule ?
⋮
▷ 6. ¬∀x A(x)                      ✓ 명논귀결 2,3,4
```

그러니까 $\forall x\, \mathsf{A}(x) \overset{\text{def}}{\equiv} p$, $\exists x\, \mathsf{B}(x) \overset{\text{def}}{\equiv} q$와 $C \overset{\text{def}}{\equiv} r$를 각각 하나의 명제문자로 보면 명제논리적 귀결관계

$$p \to (q \leftrightarrow r),\ \neg r,\ q \vDash_{PL} \neg p$$

가 성립하며, `Proofmood`의 명논귀결은 이를 추론규칙으로 구현하고 있다는 것이다. 또한 부분증명과 연속 2항술어기호문도 아래의 예에서 보듯이 제대로 처리한다.

```
⋮
2. a < b < c                       Rule ?
  3. b < c                         hyp
  ⋮
  5. φ                             Rule ?
▷ 6. φ                             ✓ 명논귀결 2,3-5
```

5.4 한정사의 도입과 소거

추론규칙 ∀ 소거를 그림 26에 보였다.

그림 26: ∀ 소거 추론규칙

	⋮	
	2. $\forall x\, \varphi(x)$	Rule ?
	⋮	
▷	4. $\varphi(t)$	✓ ∀ 소거 2

단, 여기서 t는 x를 치환가능해야 하며 $\varphi(t) \stackrel{\text{def}}{\equiv} \varphi(t/x)$ 이다. 그리고 t와 φ는 각각 논리항과 논리식을 나타내는 메타기호이다. 아래의 그림에 ∀ 소거의 적용이 불가능한 예와 가능한 예를 각각 하나씩 보였다.

	⋮			⋮	
	2. $\forall x \exists y\, A(x,y)$	Rule ?		2. $\forall x \exists z\, A(x,y)$	Rule ?
	⋮			⋮	
▷	4. $\exists y\, A(y,y)$	× ∀ 소거 2	▷	4. $\exists z\, A(y,y)$	✓ ∀ 소거 2

∀ 소거 규칙의 타당성에 대해서는 앞서 여러 번 언급되었으므로 더 이상의 설명은 생략한다.

그러면 이제 ∀ 소거 규칙을 형식 증명에서 사용한 예를 알아 보자. 아주 간단한 예는 27쪽에 나와 있는 그림 24의 라인 3에서 찾아볼 수 있으며 이것보다 조금 더 현실적인 예를 아래의 그림에 보였다.

그림 27: ∀ 소거 추론규칙의 적용 예

	1. $\forall x \forall y\, (x + y \geq 2 \cdot \mathrm{sqrt}(x \cdot y))$	hyp
	2. $\forall x\, (\mathrm{sqrt}(\mathrm{frac}(1,x) \cdot x) = 1)$	hyp
	3. $2 \cdot 1 = 2$	hyp
	4. $\forall y\, (\mathrm{frac}(1, z \cdot z + 1) + y \geq 2 \cdot \mathrm{sqrt}(\mathrm{frac}(1, z \cdot z + 1) \cdot y))$	✓ ∀ 소거 1
	5. $\mathrm{frac}(1, z \cdot z + 1) + (z \cdot z + 1) \geq 2 \cdot \mathrm{sqrt}(\mathrm{frac}(1, z \cdot z + 1) \cdot (z \cdot z + 1))$	✓ ∀ 소거 4
	6. $\mathrm{sqrt}(\mathrm{frac}(1, z \cdot z + 1) \cdot (z \cdot z + 1)) = 1$	✓ ∀ 소거 2
	7. $\mathrm{frac}(1, z \cdot z + 1) + (z \cdot z + 1) \geq 2 \cdot 1$	✓ = 소거 6,5
▷	8. $\mathrm{frac}(1, z \cdot z + 1) + (z \cdot z + 1) \geq 2$	✓ = 소거 3,7

라인 4, 5, 6에서 ∀ 소거를 사용한 것을 볼 수 있으며, 라인 7과 8에서 사용한 = 소거 추론규칙에 대해서는 조금 후에 설명할 것이다.

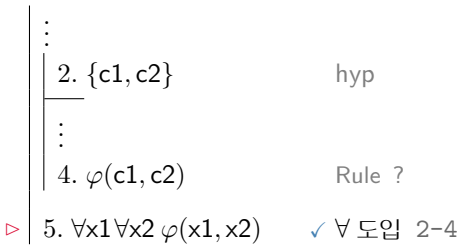
추론규칙 ∀ 도입은 그림 28과 29에 보인 규칙이다. 단, 여기서 c와 x는 새로운 기호이어야

그림 28: ∀ 도입 추론규칙 1

	⋮			⋮	
	2. {c}	hyp		2. {x}	hyp
	⋮			⋮	
	4. $\varphi(c)$	Rule ?		4. $\varphi(x)$	Rule ?
▷	5. $\forall x\, \varphi(x)$	✓ ∀ 도입 2–4	▷	5. $\forall x\, \varphi(x)$	✓ ∀ 도입 2–4

하며 $\varphi(\mathsf{c}) \overset{\text{def}}{\equiv} \varphi(\mathsf{c}/\mathsf{x})$ 이다. 상수기호 c 가 새롭다 함은 이 부분증명의 첫 라인의 이전 위치의 라인에 c 가 사용된 적이 없다는 뜻이며, 개체변수 x 가 새롭다 함은 이 부분증명의 첫 라인의 이전 위치의 라인에 x 가 자유변수로 사용된 적이 없다는 뜻이다. ∀ 도입에서 사용되는 새로운 기호를 신규기호(*fresh symbol*) 라고 부른다. 신규기호는 조금 후에 공부할 ∃ 소거 추론규칙에서도 쓰인다.

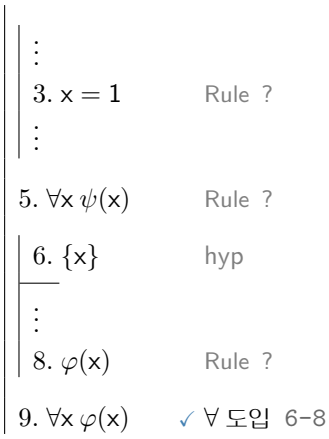
그림 29: ∀ 도입 추론규칙 2



신규기호는 2 개 이상 사용해도 된다. 예를 들어 그림 29 와 같은 증명도 허용된다. 라인 2 와 4 에서 c1, c2 를 각각 x1, x2 로 바꾸어도 됨은 물론이다.

그림 28 과 29 의 2 번 라인에서 보듯이 신규기호를 도입할 때는 중괄호를 사용한다. 5 번 라인의 묶인변수로는 아무 개체변수를 사용해도 된다. 예를 들면 그림 28 의 오른편 그림 5 번 라인에 $\forall \mathsf{y}\, \varphi(\mathsf{y})$ 를 써도 된다.

아래에 ∀ 도입의 적용이 가능한 예를 하나 보였다.



6 번 라인의 기호 x 는 새로운 기호라고 말할 수 있다. 그 이유는, 3 번 라인은 피치 증명 시스템의 라인 간 순서구조에서는 6 번 라인 이전에 있지 않은 것으로 간주되며, 5 번 라인의 x 는 자유변수로 사용된 것이 아니기 때문이다.

이제 ∀ 도입 규칙의 타당성에 대해서 알아 보자. c 는 새로운 기호이므로 우리는 c(의 해석)에 대한 어떠한 조건, 혹은 정보도 가지고 있지 않다. 그런데 $\varphi(\mathsf{c})$ 가 증명되었다면 이것은 c 가 대상영역 내의 임의의 값 u 를 취해도 $\varphi(\mathsf{c})$ 가 성립한다는 뜻이다. 다시 말해서 $\forall \mathsf{x}\, \varphi(\mathsf{x})$ 가 성립한다. 새로운 기호로 상수기호 c 아닌 개체변수 x 를 사용할 때도 마찬가지로 설명할 수 있다.

∀ 도입 규칙의 간단한 적용 예를 아래에 보였다.

```
    1. ∀x (x · x ≥ 0)          hyp
    ├──
    │ 2. {c}                   hyp
    │ ├──
    │ 3. c · c ≥ 0             ✓ ∀ 소거 1
▷   4. ∀y (y · y ≥ 0)          ✓ ∀ 도입 2-3
```

추론규칙 ∃ 도입은 아래와 같다.

그림 30: ∃ 도입 추론규칙

```
    ⋮
    2. φ(t)                    Rule ?
    ⋮
▷   4. ∃x φ(x)                 ✓ ∃ 도입 2
```

단, 여기서 t는 φ 안에서 x를 치환가능해야 하며 $\varphi(t) \overset{\text{def}}{\equiv} \varphi(t/x)$ 이다. 아래 그림에 ∃ 도입 규칙의 적용 예를 보였다.

```
    1. ∀y (0 + y = y)          hyp
    ├──
    2. ∃x∀y (x + y = y)        ✓ ∃ 도입 1
    3. 0 + 0 = 0               ✓ ∀ 소거 1
▷   4. ∃x (x + x = x)          ✓ ∃ 도입 3
```

∃ 도입 규칙은 치환 조건이 만족되지 않으면 적용할 수 없다. 예를 들어 $\varphi(x) \equiv \forall y(y = x)$, $t \equiv y$ 로 놓으면 $\varphi(t) \equiv \forall y(y = y)$ 가 되는데 이런 경우에는 $\varphi(t)$ 로부터 $\exists x \varphi(x)$ 를 얻을 수 없음이 당연하며 이를 오른편 그림에 보였다.

```
    ⋮
    2. ∀y (y = y)              Rule ?
▷   3. ∃x∀y (y = x)            x ∃ 도입 2
```

16 Exercise ∃ 도입 규칙의 타당성을 보이시오. ⊣

추론규칙 ∃ 소거를 그림 31과 32에 보였다. 단, 여기서 c와 x는 신규기호이고 c는 θ에 나타나지 않으며 x는 θ에 자유변수로 나타나지 않는다. 신규기호는 2개 이상 사용해도 된다. 예

그림 31: ∃ 소거 추론규칙 1

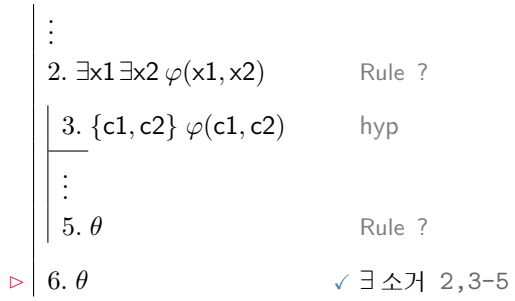
```
    ⋮
    2. ∃x φ(x)                 Rule ?
    │ 3. {c} φ(c)              hyp
    │ ├──
    │ ⋮
    │ 5. θ                     Rule ?
▷   6. θ                       ✓ ∃ 소거 2,3-5
```

```
    ⋮
    2. ∃x φ(x)                 Rule ?
    │ 3. {x} φ(x)              hyp
    │ ├──
    │ ⋮
    │ 5. θ                     Rule ?
▷   6. θ                       ✓ ∃ 소거 2,3-5
```

를 들어 그림 31과 같은 증명도 허용된다. 라인 3에서 c1, c2를 각각 x1, x2로 바꾸어도 됨은 물론이다. ∃ 소거를 사용할 때는 주석에서 전제의 순서가 지켜져야 한다. 즉, 그림 31의 라인 6에서 2,3-5를 3-5,2로 두면 안 된다.

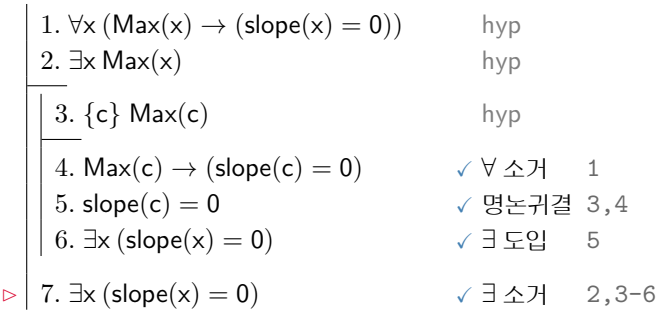
이 추론규칙의 적용 예로 미분학의 한 정리를 형식증명으로 나타내 보이겠다. 미분 가능한 함수 $f : \mathbb{R} \to \mathbb{R}$ 이 $x = c$ 에서 최대값을 가진다면 $(c, f(c))$ 에서 접선의 기울기 (slope) 는 0이다.

그림 32: ∃ 소거 추론규칙 2



그러므로 f 가 최대값을 가진다는 가설로부터 f 의 그래프에는 접선의 기울기가 0 인 점이 있다는 결론을 얻을 수 있다. 그림 33 은 이 논의의 형식증명이다.

그림 33: ∃ 소거 추론규칙 적용 예



17 Exercise ∃ 소거 규칙이 명제논리에서의 ∨ 소거와 비슷한 부분을 찾으시오. ⊣

18 Exercise ∃ 소거 규칙의 타당성을 보이시오. ⊣

19 Remark 부분증명은 가설 라인 1 개와 결론 라인 1 개 이상으로 이루어진다. 명제논리에서 사용하는 부분증명의 가설 라인에는 반드시 논리식이 들어 있어야 한다. 그러나 1 계논리에서 사용하는 부분증명의 가설라인에는 논리식이 들어 있을 수도 있고, 신규기호가 들어 있을 수도 있으며 (∀ 도입), 신규기호와 또한 그 기호를 사용한 논리식이 들어 있을 수도 있음을 (∃ 소거) 알아야 한다.

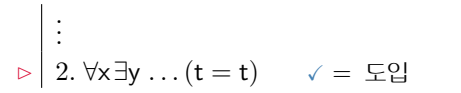
부분증명의 가설 라인에 논리식 없이 신규기호만 들어 있는 경우 주석에 `hyp` 를 두는 것은 엄격하게 말하자면 부정확한 표현이라고 불 수 있다. 이러한 부분증명은 새로운 가설의 적용범위를 지정해 주는 것이 아니라 새로 도입한 기호의 사용범위를 지정해 주는 역할을 한다. ⊣

5.5 등호의 도입과 소거

토톨로지는 피치 증명의 임의의 위치에 있는 라인에 자유로이 넣을 수 있다. $t = t$ 형태의 논리식도 그러하다. 이 규칙의 타당성은 모든 사람이 자명하게 받아들일 것이다. 이제 ∀ 도입 규칙과 ∃ 도입 규칙을 이용하면 등식 $t = t$ 의 앞에 임의의 한정사들을 붙인 논리식을 자유로이 사용할 수 있음을 알 수 있다.

이 규칙은 전제를 필요로 하지 않는 아주 간단하고 쉬운 추론규칙이며 그림 34 에 이를 보였다.

그림 34: = 도입 추론규칙



다음에 공부할 = 소거 추론규칙은 이것보다는 조금 더 복잡하다. 등호(=)의 기본 성질은 다음과 같다.

> $x = y$ 이면 어떤 x 에 대한 명제가 주어졌을 때 이 명제에서 x 를 y 로 바꾸어도 원래의 명제와 동등하다.

위 사실의 기호논리 버전은 다음과 같다.

$$\forall x \forall y \left(x = y \;\Rightarrow\; (\varphi(x) \leftrightarrow \varphi(y)) \right) \tag{16}$$

그런데 흔히 (16)을 다음과 같이 쓴다.

$$\forall x \forall y \left(x = y \;\Rightarrow\; (\varphi(x, x) \leftrightarrow \varphi(x, y)) \right) \tag{17}$$

이는 φ 에 x 가 여러 군데 나타나는 경우, 이 중에 일부의 x 만 y 로 바꾸어도 된다는 것을 의미하는 것이다. 즉, (17)에서

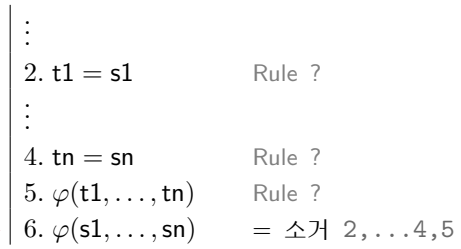
> $\varphi(x, y)$ 는 $\varphi(x, x)$ 에서 x 의 나타남의 일부, 혹은 전부를 y 로 대체해서 얻은 논리식

을 나타내고 있는 것이다.

20 Remark 치환(*substitution*)과 대체(*replacement*)는 의미상의 중요한 차이가 있다. 치환이라 함은 어떤 표현에 나타나는 변수를 다른 것(변수 혹은 표현)으로 바꾸는 것으로, 별도의 설명이 없으면 그 변수의 모든 나타남을 바꾸는 것이다. 대체라 함은 어떤 표현에 나타나는 부분표현(*subexpression*)을 다른 것으로 바꾸는 것으로, 별도의 설명이 없으면 그 부분표현의 나타남의 일부만 바꾸는 것이다. ⊣

= 소거 추론규칙은 `Proofmood`에서 그림 35와 같이 구현되어 있다.

그림 35: = 소거 추론규칙



여기서 6 번 라인의 논리식 $\varphi(\mathsf{s1}, \ldots, \mathsf{sn})$ 는 5 번 라인의 논리식 $\varphi(\mathsf{t1}, \ldots, \mathsf{tn})$ 에서, 각 $i = 1, \ldots, \mathsf{n}$ 에 대해서 $\mathsf{t}i$ 들을 $\mathsf{s}i$ 로 대체해서 얻은 논리식을 뜻한다.

21 Exercise = 도입 추론규칙과 = 소거 추론규칙의 타당성을 보이시오. ⊣

5.6 추론규칙의 확충

지금까지 설명한 7개의 추론규칙, 즉 명제논리적 귀결, 전칭한정사의 도입과 소거, 존재한정사의 도입과 소거 및 등호의 도입과 소거는 모든 1계항진 논리식을 증명하기에 충분하다.

그런데 실제로 수학의 증명을 작성할 때 이 7개의 추론규칙만 사용해서는 불편한 점이 많다. Proofmood에는 증명의 효율성을 위하여 더 풍부한 추론규칙들이 구현되어 있으며 이들을 이 절에서 공부할 것이다. 이 절부터 시작하여 앞으로 공부할 심화 추론규칙들은 항진 논리식을 증명하는 데 꼭 필요한 것이 아니며 다만 증명의 효율성을 위하여 도입된 것임을 다시 한 번 말해 둔다.

∀ 도입과 ∃ 소거에서 한정사 2개 이상을 한꺼번에 사용할 수 있음을 그림 29와 32에서 설명했는데 이는 ∀ 소거와 ∃ 도입에서도 마찬가지다. 예를 들어 설명하겠다. 수학자라면 누구나 알고 있는 사실로 전칭한정사의 교환법칙이 있다. 이 법칙을 기호를 이용하여 나타내면 다음과 같다.

$$\forall x \forall y\, \varphi(x,y) \;\Leftrightarrow\; \forall y \forall x\, \varphi(x,y) \tag{18}$$

위의 양방향함의의 $\Rightarrow$를 Proofmood에서 증명하여 그림 36에 보였다. ($\Leftarrow$는 똑같은 방법으로 증명되므로 생략한다.)

그림 36: ∀ 소거, 도입 추론규칙 2

	1. ∀x∀y φ(x, y)	hyp
	2. {x, y}	hyp
	3. φ(x, y)	✓ ∀ 소거 1
▷	4. ∀y∀x φ(x, y)	✓ ∀ 도입 2–3

3번 라인의 ∀ 소거에서 두 한정사를 한꺼번에 처리하였으며 4번 라인의 ∀ 도입에서도 역시 그러하였다. 그런데 실은 3번 라인의 ∀ 소거에서는 두 개의 한정사를 동시에 처리할 수 있다는 규칙이 꼭 필요한 것은 아니다. 그림 37을 보면 이 사실을 알 수 있을 것이다.

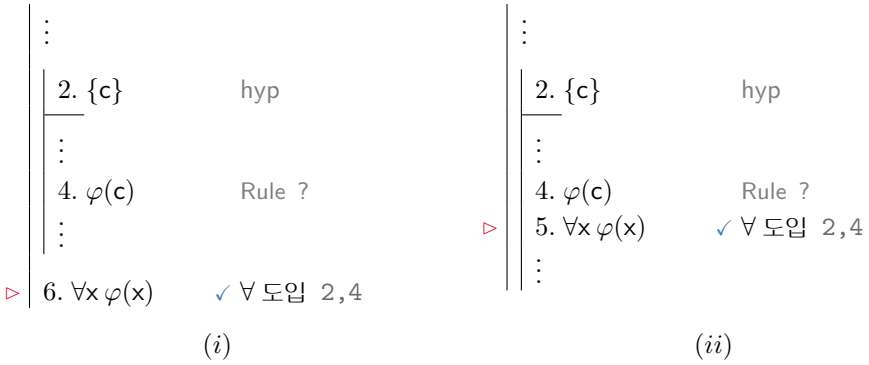
그림 37: ∀ 소거, 도입의 예

	1. ∀x∀y φ(x, y)	hyp
	2. {x, y}	hyp
	3. ∀y φ(x, y)	✓ ∀ 소거 1
▷	4. φ(x, y)	✓ ∀ 소거 3
	5. ∀y∀x φ(x, y)	✓ ∀ 도입 2–4

22 Exercise 존재한정사의 교환법칙을 쓰고 피치 증명을 찾으시오. ∃ 소거에서는 두 개의 한정사를 동시에 처리하는 규칙이 꼭 필요하지는 않음을 보이시오. ⊣

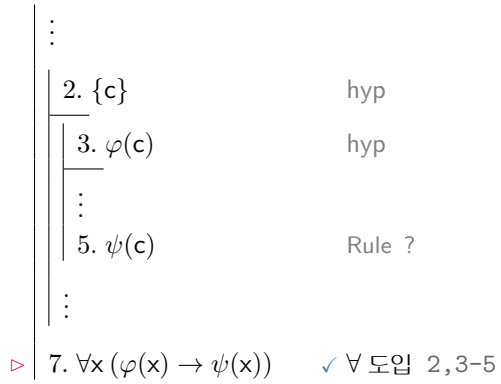
∀ 도입은 전제로 원래는 하나의 부분증명을 사용하게 되어 있는데, 그림 38.(i)과 같이 두 개의 라인을 전제로 사용해도 된다. 그리고 그림 38.(ii)와 같이 결론 ∀x φ(x)를 부분증명 내에 두어도 된다.

그림 38: ∀ 도입 추론규칙 3



∀ 도입 규칙의 또 하나의 버전으로서 하나의 라인과 하나의 부분증명을 전제로 사용하는 것을 그림 39에 보였다. 이 규칙이 없어도 결론 라인 7은, 그림 39에서 부분증명 3–5 바로 아래에

그림 39: ∀ 도입 추론규칙 4



명논귀결을 써서 $\varphi(c) \to \psi(c)$ 를 넣은 다음 그림 38.(i) 규칙을 사용하여 얻을 수 있다. 그러므로 이 규칙은 라인 하나를 절약하는 효과밖에 없으나 실제 피치 증명을 경험해 보면 상당히 편리함을 알 수 있을 것이다. 결론 라인 7은 라인 2를 가설로 하는 부분증명 내부에 넣어도 된다. 왜 그런지 생각해 보라. 이 규칙은 실은 ∀ 도입과 → 도입을 합쳐 놓은 것인데 Proofmood에서는 그냥 ∀ 도입 메뉴에서 처리하도록 하였다.

그림 38과 39에서 상수기호 대신 개체변수를 쓰거나 이 기호들을 2개 이상 사용하는 것도 허용된다. 그리고 그림 36에서 한정사 3개 이상을 사용하는 것도 허용된다.

∀ 소거를 바탕으로 하는 유용한 심화 규칙 2개를 그림 40의 (i) 과 (ii) 에 보였다. 물론 (ii) 에서 $\varphi(t)$ 대신 $\psi(t)$ 를 써도 된다. 여기서 논리항 t가 적절한 치환조건을 만족해야 하는 것은 당연하다. 이 규칙들도 라인 하나를 절약하는 효과를 가진다. 이 규칙들은 실은 각각 ∀ 소거와 → 소거, 혹은 ∧ 소거를 합쳐 놓은 것인데 Proofmood에서는 그냥 ∀ 소거 메뉴에서 처리하도록 하였다. 연속 2항술어기호문도 그림 41에서 보듯이 ∀ 소거에서 제대로 처리된다. ∀ 소거와 ↔ 소거를 합쳐 놓은 규칙도 그림 42에 구현되어 있다.

그림 40.(i) 과 그림 42에 보인 추론규칙에서 전제의 순서는 지켜져야 한다. 즉, ∀ 소거 2,3을 ∀ 소거 3,2로 쓰면 안 된다.

∃ 소거는 전제로 하나의 라인(존재문)과 하나의 부분증명을 사용하게 되어 있는데, 그림 43.(i) 과 같이 3 개의 라인을 전제로 사용하는 규칙을 사용해도 된다. 그리고 그림 43.(ii) 와

그림 40: ∀ 소거 추론규칙 3

	⋮		
	2. $\forall x\,(\varphi(x) \to \psi(x))$	Rule ?	
	3. $\varphi(t)$	Rule ?	
	⋮		
▷	5. $\psi(t)$	✓ ∀ 소거 2,3	

(i)

	⋮		
	2. $\forall x\,(\varphi(x) \wedge \psi(x))$	Rule ?	
	⋮		
▷	4. $\varphi(t)$	✓ ∀ 소거 2	
	⋮		

(ii)

그림 41: ∀ 소거, 연속 2 항술어문

	⋮	
	2. $\forall x\,(f(x) < g(x) < h(x))$	Rule ?
	⋮	
▷	3. $f(t) < g(t)$	✓ ∀ 소거 2

같이 θ 대신에 $\psi(c1, c2)$ 를 사용해도 된다. 즉, 신규기호를 포함한 논리식을 사용할 수 있다는 것이다. 단, 이때는 결론 라인 7에서 보듯이 존재한정사를 붙여 주어야 한다.

∀ 도입 규칙의 그림 38의 4번 라인에 있는 논리식 $\varphi(c)$ 를 그림 39에서는 부분증명 3–5로 바꾸어 결론으로 전칭한정함의문을 얻었는데 ∃ 소거에서도 이와 똑같이 할 수 있다. 그림 44에 이러한 추론의 예를 보였다.

23 Exercise 그림 38 ~ 44에서 보인 추론규칙들의 타당성을 확인하시오. ⊣

지금까지 한정사의 도입과 소거에 대한 추론규칙들의 확장 버전을 알아 보았다. 이제 등호에 대해서 알아 보자. 등호 도입 규칙에는 그림 34에서 보인 것 이상의 확장 버전이 없다. 등호 소거의 확장 규칙은 45와 46 및 48에 보였다.

그림 45.(i) 은 등호의 대칭성이며 이 규칙의 타당성의 증명은 의미론적인 방법을 사용할 필요가 없이 피치 시스템을 사용하여 다음과 같이 얻을 수 있다.

	2. $t = s$	Rule ?
	3. $t = t$	✓ = 도입
▷	4. $s = t$	✓ = 소거 2,3

$\varphi(x) \equiv (x = t)$ 로 두면 아래 그림의 3번 라인은 $\varphi(t)$ 이고 4번 라인은 $\varphi(s)$ 이므로 그림 35에서 보인 등호 소거 규칙이 4번 라인에 적용이 가능함을 알 수 있다. 이것이 그림 45.(i) 의 증명이다.

그림 45.(ii) 는 등식논리 (equational logic) 의 기본 규칙 중 하나이며 이 규칙의 타당성의 증명은 의미론적인 방법을 사용할 필요가 없이 다음과 같이 얻을 수 있다.

	2. $t1 = s1$	Rule ?
	3. $t2 = s2$	Rule ?
	4. $r(t1, t2) = r(t1, t2)$	= 도입
▷	5. $r(t1, t2) = r(s1, s2)$	✓ = 소거 2,3,4

그림 42: ∀ 소거 추론규칙 4

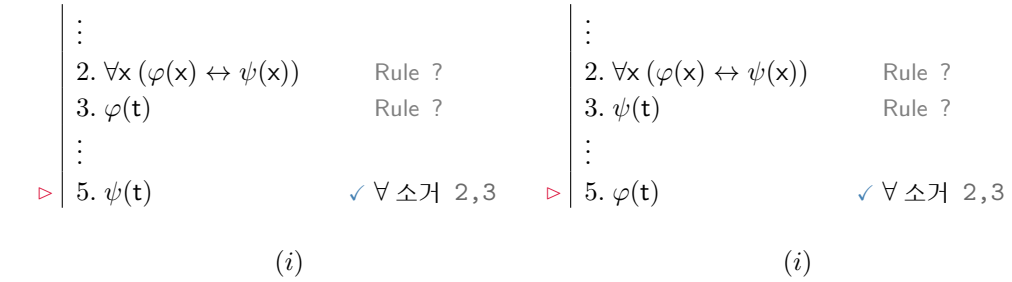
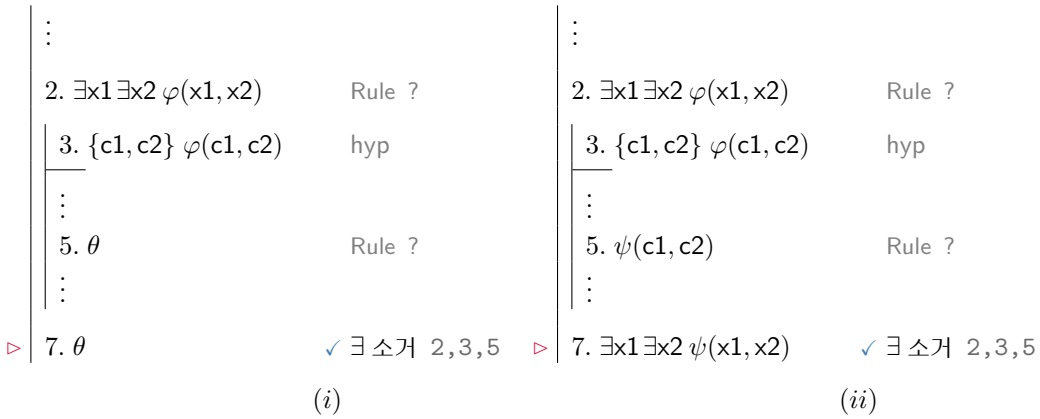


그림 43: ∃ 소거 추론규칙 3



$\varphi(\mathsf{x1},\mathsf{x2}) \equiv (\mathsf{r}(\mathsf{t1},\mathsf{t2}) = \mathsf{r}(\mathsf{x1},\mathsf{x2}))$ 로 두면 오른편 그림의 4 번 라인은 $\varphi(\mathsf{t1},\mathsf{t2})$ 이고 5 번 라인은 $\varphi(\mathsf{s1},\mathsf{s2})$ 이므로 그림 35 에서 보인 등호 소거 규칙이 5 번 라인에 적용이 가능함을 알 수 있다. 이것이 그림 45.(ii) 의 증명이다.

그림 45.(ii) 에서 $\mathsf{t} = \mathsf{t}$ 형태의 라인은 항상 전제에 있는 것으로 가정해도 좋다. 예를 들어 아래 그림의 3 번 라인은 = 소거 추론규칙 2 를 통하여 얻었는데 이것은 전제에 라인 $\mathsf{t} = \mathsf{t}$ 와 $\mathsf{s} = \mathsf{s}$ 가 있는 것으로 가정하였기에 가능한 것이다.

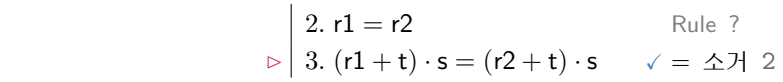


그림 45.(ii) 에 보인 추론 규칙은 등호의 가로 소거(*horizontal elimination*) 라고 부르고, 그림 35 에서 보인 추론 규칙은 등호의 세로 소거(*vertical elimination*) 라고 부르기로 하자. 이렇게 부르는 이유는 등호로 연결된 항들이 (t_i와 s_i가) 전자의 경우에는 가로로 놓여 있고 후자의 경우에는 세로로 놓여 있기 때문이다.

그림 46 은 전제가 3 개인 경우의 등호의 추이율(*transitivity*) 이며 `Proofmood` 에는 이러한 등호의 추이율이 전제의 개수에 제한 없이 구현되어 있다. 이 규칙의 타당성은 의미론적 방법으로 쉽게 증명되며, 전제의 개수가 2 인 경우에는 다음의 그림과 같은 형식 증명이 가능하다.

24 Exercise 위 그림의 증명을 그림 35 의 틀에 맞추어 설명하시오. ⊣

등호의 추이율은 부분항을 이해한다. 예를 들면 다음과 같은 증명이 가능하다.

그림 44: ∃ 소거 추론규칙 4

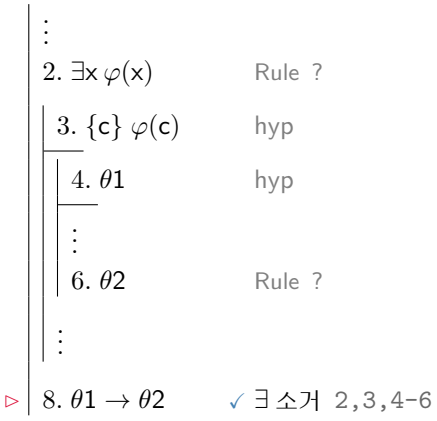
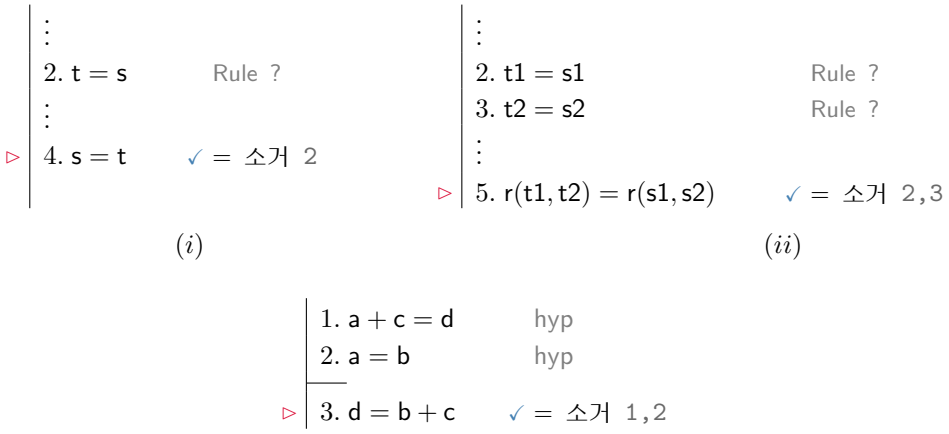


그림 45: = 소거 추론규칙 2



라인 2와 라인 1에는 공통되는 항이 없어 원래는 추이율을 적용할 수 없지만, 라인 2의 좌변 a는 라인 1의 좌변 a + c의 부분항임에 착안하면 라인 2가 a + c = b + c인 것처럼 사용해도 된다. 그래서 라인 3을 결론으로 얻을 수 있다. 등호의 추이율이 부분항을 이해한다는 것은 바로 이런 의미이다. 다시 말하면 아래의 그림에서 라인 2의 좌변이 라인 1의 좌변의 부분항인데, 이때 라인 2 대신 라인 2′이 존재하는 것으로 가정하고 = 소거를 적용하여 라인 3을 얻을 수 있다는 것이다.

라인 2의 t가 라인 1의 한 항의 부분항으로 두 가지 이상의 방법으로 나타날 때는 문제가 발생할 수 있다. 아래의 예를 보면 이 문제를 파악할 수 있을 것이다.

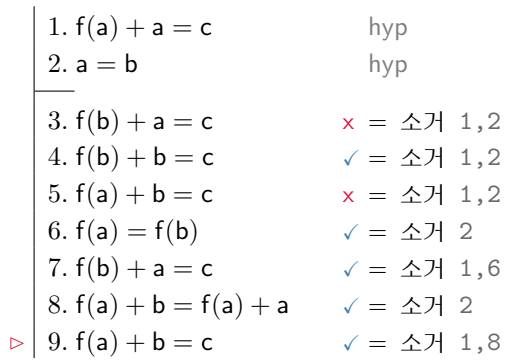


그림 46: = 소거 추론규칙 3

```
  ⋮
  3. t1 = t2        Rule ?
  4. t2 = t3        Rule ?
  5. t3 = t4        Rule ?
  6.                Rule ?
▷ 7. t1 = t4        ✓ = 소거 3,4,5
```

그림 47: 추이율의 증명, 전제가 2 개일 경우

```
  2. t1 = t2        Rule ?
  3. t2 = t3        Rule ?
  4. t2 = t1        ✓ = 소거 2
  5. t2 = t2        ✓ = 도입
▷ 6. t1 = t3        ✓ = 소거 4,3,5
```

라인 2의 a를 라인 1의 $f(a)+a$의 부분항으로 보는 방법은 3가지가 있다.

$$r1(x) \equiv f(x)+a,\ r2(x)=f(x)+x,\ r3(x)=f(a)+x$$

로 두었을 때 $r1(a/x)=r2(a/x)=r3(a/x)=f(a)+a$가 되는 것이다. Proofmood의 현재 버전에서는 이 셋 중에 r2만 채택하여 사용한다.

입증에 실패한 라인 3은 라인 6을 도입한 후에 라인 7에서 입증하였고, 라인 5는 라인 8을 도입한 후에 라인 9에서 입증하였다. 일반적으로 = 소거에서 부분항 처리에 관련된 문제는 추이율에서만 나타나므로 가로 소거, 혹은 세로 소거를 이용하여 하나의 라인을 추가로 얻은 후에 추이율을 쓰면 해결된다.

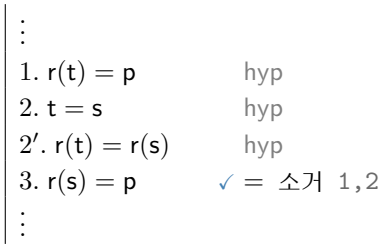
수학에서 흔히 3등식, 즉 $t_1=t_2=t_3$ 형태의 표현을 사용하는데 이것은 $(t_1=t_2)\wedge(t_2=t_3)$를 줄여 쓴 것이다. 이 경우 등호의 추이율에 의하여 $t_1=t_3$도 성립한다. 따라서 Proofmood는 t1 = t2 = t3 형태의 전제가 있으면 이를 마치 3 개의 전제 t1 = t2, t2 = t3, t1 = t3가 있는 것처럼 취급한다. 예를 들어 다음과 같은 증명이 가능하다.

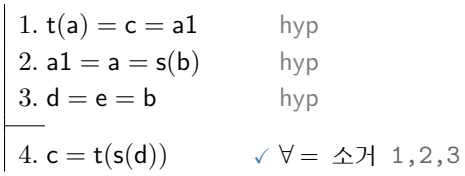
```
  ⋮
  2. t1 = t2 = t3          Rule ?
  3. s1 = s2               Rule ?
▷ 4. r(t1,s1) = r(t3,s2)   ✓ = 소거 2,3
```

그림 35, 45 및 46에 기술된 = 소거 규칙은 전제들과 결론의 일부, 혹은 전부의 좌변과 우변을 바꾸어도 잘 적용된다. 다만 결론의 주석에서 전제들의 순서는 때로 중요할 수 있는데 그림 35에서 결론의 주석의 마지막 전제는 반드시 5가 되어야 한다. 즉, 세로 소거에서는 마지막 전제의 위치가 지켜져야 한다. 그리고 그림 46에서는 전제들의 순서가 하나라도 뒤바뀌면 안 된다. 즉, 추이율에서는 모든 전제들이 제 자리에 있어야만 한다.

등호의 추이율의 적용 예를 하나 더 아래에 보였다.

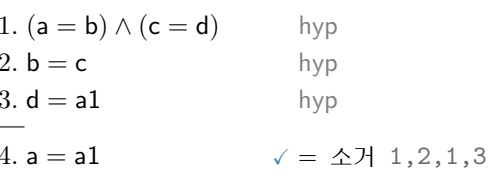
그림 48: = 소거 추론규칙 4





위의 증명은 등호의 추이율을 사용하고 있는데, 라인 1과 라인 2의 연결은 2가지 방법으로 가능하다. 즉 라인 2의 a1이 라인 1에 나타나고 있음을 이용해도 되고, 라인 2의 a의 초항(superterm) t(a)가 라인 1에 있음을 이용해도 된다. 그런데 여기서 전자를 취하면 라인 4까지 이어지지 못하고 후자를 취하면 라인 4까지 이어진다. Proofmood는 이렇게 추이율의 적용에 2가지 이상의 경로가 존재할 경우 잘못된 경로는 버리고 올바른 경로를 취할 수 있도록 구현되어 있다.

= 소거 규칙에서는 전제로 등식, 혹은 3등식을 사용할 수 있음을 보았는데, 여기에 더하여 등식 2개의 논리곱, 혹은 등식 3개의 논리곱도 전제로 사용할 수 있도록 구현되어 있다. 아래의 그림에 하나의 예를 보였다.



이 그림의 라인 4의 주석을 보면 전제에 1이 2번 나온다. 두 번째 1을 넣을 때는 첫 번째 1이 지워지는 것을 막기 위하여 컨트롤키(Ctrl)를 누른 상태에서 오른쪽 마우스를 클릭해야 한다.

이제까지 증명의 효율을 높이기 위한 확장된 추론 규칙들을 몇 개 소개하였는데, 실제 수학에서 사용되는 증명 규칙은 이보다 훨씬 더 많다.[†] 현재 Proofmood에 구현된 것만 해도 ∀= 소거, 결합법칙, 결합교환, 1논공식(1계 논리 추론 공식), 귀납법 및 메타치환이 있으며 이들에 대한 설명은 다음 기회로 미룬다.

[†]물론 수학자들은 이런 규칙들을 사용하고 있다는 사실을 의식하지도 않고 그냥 자연스레 사용한다.