# 11장 논리형 언어

## 11.3 이론적 기초

**예 11.39**
**명제**

수학적 논리에서 술어란 상수(원자)나 변수를 값 참(true)과 거짓(false)에 매핑하는 함수다. 술어 연산은 술어 적용, 연산자, 정량자 ∀과 ∃로 구성된 명제(문장)의 구성과 추론을 위한 표기법과 추론 규칙을 제공한다.[1] 연산자에는 그리고(and, ∧), 또는(or, ∨), 부정(not, ¬), 함축(→), 등가(↔)가 있다. 정량자는 람다 연산에서 λ가 변수를 도입하는 것과 유사하게 덧붙여진 명제에 바인딩된 변수를 도입하는 데 사용된다. 보편 정량자 ∀는 명제가 변수의 모든 값에 대해 참임을 나타낸다. 존재 정량자 ∃는 명제가 최소한 변수의 값 하나에 대해 참임을 나타낸다. 아래에 몇 가지 예를 담았다.

$$\forall C[rainy(C) \wedge cold(C) \rightarrow snowy(C)]$$

(모든 도시 C에 대해 C에 비가 오고 C가 춥다면 C에는 눈이 온다)

$$\forall A, \forall B[(\exists C[takes(A,C) \wedge takes(B,C)]) \rightarrow classmates(A,B)]$$

(모든 학생 A와 B에 대해 A가 C를 수강하고 B도 C를 수강하는 수업 C가 있다면 A와 B는 급우다)

$$\forall N[(N > 2) \rightarrow \neg(\exists A, \exists B, \exists C[A^N + B^N = C^N])]$$

(페르마의 마지막 정리)

**예 11.40**
**다양한 표현 방법**

술어 연산의 흥미로운 특징 중 하나는 동일한 것을 표현하는 방법이 다양하다는 것이다. 다음과 같은 예를 들 수 있다.

---

주1. 엄격히 말해서 여기서 설명하는 것은 1차 술어 연산이다. 술어를 원자나 변수뿐만 아니라 술어에도 적용할 수 있는 고차 연산도 존재한다. 프롤로그는 사용자가 call을 사용해서 고차 술어를 정의할 수 있게 해준다. 이러한 술어의 형식화는 이 책의 범위 밖이다.

$(P_1 \rightarrow P_2) \equiv (\neg P_1 \vee P_2)$

$(\neg \exists X[P(X)]) \equiv (\forall X[\neg P(X)])$

$\neg(P_1 \wedge P_2) \equiv (\neg P_1 \vee \neg P_2)$

수식의 이러한 융통성은 사람에게는 편리하지만 자동 정리 증명에는 방해가 될 수 있다. 명제는 어떤 종류의 정규 형태로 존재하는 경우 알고리즘적으로 처리하기가 좀 더 쉽다. 널리 쓰이는 정규 형태로 절 형태가 있으며 이를 아래서 다룬다.

## 【11.3.1】 절 형태

절 형태는 프롤로그 프로그램의 구조와 매우 긴밀히 연관되어 있다. 일단 절 형태로 된 명제가 나오면 이는 비교적 쉽게 프롤로그로 변환할 수 있다. 그러나 이 변환은 애초부터 완벽하지 않다. 술어 연산에는 프롤로그가 나타내지 못하는 면이 있으며, 마찬가지로 프롤로그에는 술어 연산에 유사한 것이 없는 부분(예를 들어 명령형 기능과 데이터베이스 처리 기능)이 있다.

클락신과 멜리쉬[CM94, 10장]는 임의의 1차 술어 명제를 절 형태로 변환하는 (마틴 데이비스[Dav63]의 논문에 크게 기반한) 5단계의 절차를 기술했다. 여기서는 이 절차를 따른다.

**예 11.41**
**절 형태로의 변환**

첫 번째 단계에서는 함축과 등가 연산자를 제거한다. 구체적인 예로 다음과 같은 명제를 보자.

$\forall A[\neg student(A) \rightarrow (\neg dorm\_resident(A) \wedge \neg \exists B[takes(A,B) \wedge class(B)])]$

이 명제는 아래와 같이 변환된다.

$\forall A[student(A) \vee (\neg dorm\_resident(A) \wedge \neg \exists B[takes(A,B) \wedge class(B)])]$

두 번째 단계에서는 개별 항(인자에 적용된 술어)만 부정된 항목으로 남게 부정을 내부로 이동시킨다.

$\forall A[student(A) \vee (\neg dorm\_resident(A) \wedge \forall B[\neg(takes(A,B) \wedge class(B))])]$

$\equiv \forall A[student(A) \vee (\neg dorm\_resident(A) \wedge \forall B[\neg takes(A,B) \vee \neg class(B)])]$

세 번째 단계에서는 스콜렘화(Skolemization, 논리학자 토랄프 스콜렘의 이름을 딴 것)라는 기술을 사용해서 존재 정량자를 제거한다. 이 기술은 11.3.3절에서 논한다. 위의 예는 존재 정량자를 가지지 않으므로 이 단계는 그냥 지나간다.

네 번째 단계에서는 모든 보편 정량자를 명제 밖으로 이동시킨다(이름 충돌이 없는 한 이는 명제의 의미를 변경하지 않는다). 그 후 모든 변수가 보편 양화되었으므로 명시적인 정량자는 제거한다는 관례를 따른다.

$student(A) \vee (\neg dorm\_resident(A) \wedge (\neg takes(A,B) \vee \neg class(B)))$

끝으로 다섯 번째 단계에서는 불 대수(Boolean algebra)의 분배, 결합, 교환 법칙을 사용해서 명제를 논리곱 정규형으로 변환한다. 논리곱 정규형에서는 연산자 ^와 ∨가 세 수준 깊이 이상 내포되지 않으며 외부에 ^, 내부에 ∨ 연산자가 나온다.

```
(student(A) ∨ ¬dorm_resident(A)) ∧ (student(A) ∨ ¬takes(A,B) ∨ ¬class(B))
```

이제 명제를 절 형태로 변환했다. 명확히 말해서 위의 절 형태는 개별 항에만 부정이 있으며 존재 정량자가 없고 모든 변수(즉, 상수와 술어를 제외한 모든 이름)에 대해 묵시적인 보편 정량자를 가지는 논리곱 정규형이다. 절은 바깥쪽 수준에 있는 항목으로 서로 ^로 묶인 것들이다.

**예 11.42**
**프롤로그로의 변환**

명제를 프롤로그로 변환하기 위해 각 논리 절을 프롤로그 사실이나 규칙으로 변환한다. 절 안에서는 부정된 항을 오른쪽으로 옮기고 부정되지 않은 항을 왼쪽으로 옮기기 위해(앞서 구한 절 형태는 이미 이 형태를 가진다) 교환 가능성을 이용한다. 그러면 다음과 같이 논리합을 함축으로 다시 변환할 수 있다.

```
(student(A) ← ¬(¬dorm_resident(A)))
∧(student(A) ← ¬(¬takes(A,B) ∨¬class(B)))
≡(student(A) ← dorm_resident(A))
∧(student(A) ← (takes(A,B) ∧ class(B)))
```

이는 혼 절(Horn clause)들로 이제 프롤로그로의 변환은 매우 쉽다.

## 【11.3.2】 한계점

11.1절의 처음 부분에서 혼 절은 전부는 아니지만 대부분의 1차 술어 연산을 표현하는 데 사용할 수 있다고 알아보았다. 그러면 혼 절이 표현하지 못하는 것은 무엇일까? 변환에서는 무엇이 잘못될 수 있을까? 이에 대한 답은 각 절의 부정되지 않은 항의 수와 관련 있다. 절이 둘 이상의 부정되지 않은 항을 가지는 경우 이를 함축으로 변환하려고 하면 기호의 좌편에 논리합이 존재하게 되며, 이는 혼 절에서 허용되지 않는다. 마찬가지로 부정되지 않은 항으로 끝나는 절의 경우에는 변환한 결과가 헤드 없는 혼 절이 되며, 이는 프롤로그에서 질의로만 사용할 수 있을 뿐 데이터베이스 원소로는 사용할 수 없다.

**예 11.43**
**논리합 좌편**

논리합 헤드의 예로 "살아 있는 모든 것은 동물이거나 식물이다."라는 문장을 생각해 보자. 다음과 같은 절 형태로 나타낼 수 있다.

```
animal(X) ∨plant(X) ∨¬living(X)
```

또는 동치로서 다음과 같이 나타낼 수도 있다.

```
animal(X) ∨plant(X) ← living(X)
```

규칙의 좌편에는 하나의 항만이 올 수 있으므로 이 절 형태에 가장 가까운 프롤로그 코드는 다음과 같다.

```
animal(X) :- living(X), not(plant(X)).
plant(X) :- living(X), not(animal(X)).
```

그러나 프롤로그의 not은 거짓이 아니라 증명할 수 없음을 나타내기 때문에 이는 절 형태와 동일하지 않다.

**예 11.44**
**빈 좌편**

빈 헤드의 예로 페르마의 마지막 정리(심화학습에 있는 예 11.39)를 보자. 수학적 사실로부터 추상화하면 다음과 같이 쓸 수 있다.

$$\forall N[\mathtt{big}(N) \rightarrow \neg(\exists A,\ \exists B,\ \exists C[\mathtt{works}(A,B,C,N)])]$$

이는 다음과 같은 절 형태로 변환할 수 있다.

$$\neg\mathtt{big}(N) \vee \neg\mathtt{works}(A,B,C,N)$$

이는 다음과 같은 프롤로그 질의(종료되지 않는 질의)로 나타낼 수는 있지만 사실이나 규칙으로는 표현할 수 없다.

```
?- big(N), works(A, B, C, N).
```

**예 11.45**
**모순 검색으로서의 정리 증명**

주의 깊은 독자는 사실은 (함축된) 프롤로그 :- 표시의 좌편에 입력되며, 질의는 우편에 입력된다는 것을 알아챘을 수 있다. 다음은 그 예다.

```
rainy(rochester).
```

반면 질의는 우편에 입력된다.

```
?- rainy(rochester).
```

전자는 다음과 같은 의미를 가진다.

$$\mathtt{rainy(rochester)} \leftarrow \mathtt{true}$$

후자는 다음과 같은 의미를 가진다.

$$\mathtt{false} \leftarrow \mathtt{rainy(rochester)}$$

위의 두 명제에 도출(resolution)을 적용하면 모순에 도달하게 된다.

$$\mathtt{false} \leftarrow \mathtt{true}$$

이러한 사실로부터 자동 정리 증명을 위한 기법을 생각해볼 수 있다. 공리 모음이 주어진 상태에서 정리를 증명하고자 한다면 정리의 부정을 임시로 데이터베이스에 추가한 다음 일련의 도출 연산을 통해 모순을 이끌어 내게 시도하면 된다. 모순에 도달하면 해당 정리는 증명된 것이다.

### 【11.3.3】 스콜렘화

**예 11.46**
**스콜렘 상수**

©(심화학습에 있는) 예 11.41에서는 술어 연산의 명제를 존재 정량자에 대한 걱정 없이 절 형태로 변환했다. 그러나 아래와 같은 문장의 경우는 어떤가?

```
∃X[takes(X, cs254) ∧ class year(X, 2)]
```

(cs254에는 최소한 한 명의 2학년이 있다) 존재 정량자를 제거하기 위해 스콜렘 상수 x를 도입할 수 있다.

```
takes(x, cs254), class_year(x, 2)
```

이러한 변경의 수학적 정당화는 선택 공리라는 것에 기반한다. 직관적으로 문장을 참으로 만드는 X가 존재한다면 단순히 하나를 골라 이를 x로 명명한 후 나머지 작업을 진행할 수 있다(문장을 참으로 만드는 X가 존재하지 않는 경우에도 임의의 x를 선택할 수 있다. 이 경우 문장은 여전히 거짓이 된다). 스콜렘 상수가 별개일 필요는 없다. 예를 들어 x는 his201의 2학년 학생을 나타내는 다른 상수 y와 동일한 학생을 명명할 수 있다.

**예 11.47**
**스콜렘 함수**

때때로 존재 양화된 변수는 임의의 상수 x로 대체할 수 있다. 그러나 가끔 주변의 일반적인 정량자에 의해 제약을 받는다. 다음의 예를 보자.

```
∀X[¬dorm_resident(X)∨∃A[campus_address_of(X,A)]]
```

(모든 기숙사 거주자는 캠퍼스 주소를 가진다) 존재 정량자를 제거하기 위해서는 X에 대한 주소를 선택해야 한다. X가 누구인지 모르기 때문에(이는 모든 기숙사 거주자에 대한 일반 문장이다) X에 의존적인 주소를 선택해야 한다.

```
∀X[¬dorm_resident(X)∨campus_address_of(X, f(X))]
```

여기서 f는 스콜렘 함수다. 단순한 스콜렘 상수를 사용했다면 모든 기숙사 거주자가 공유하는 단일 주소가 존재한다고 말하는 것이 된다.

**예 11.48**
**스콜렘화의 한계점**

스콜렘화를 통해 프롤로그로 변환할 수 있는 절 형태를 얻을 수 있는지 여부는 상수가 무엇인지를 알 필요가 있는지 여부에 달려있다. 술어 takes와 class_year를 사용해서 cs254에는 2학년 학생이 있다는 사실을 주장하고자 한다면 다음과 같이 쓸 수 있다.

```
takes(the_distinguished_sophomore_in_254, cs254).
class_year(the_distinguished_sophomore_in_254, 2).
```

마찬가지로 모든 기숙사 거주자가 캠퍼스 주소를 가진다는 것을 다음과 같이 나타낼 수 있다.

```
campus_address_of(X, the_dorm_address_of(X)) :- dorm_resident(X).
```

이제 2학년 학생이 수강하는 수업을 검색할 수 있다.

```
sophomore_class(C) :- takes(X, C), class_year(X, 2).
?- sophomore_class(C).
C = cs254
```

캠퍼스 주소를 가지는 사람도 검색할 수 있다.

```
has_campus_address(X) :- campus_address_of(X, Y).
dorm_resident(li_ying).
?- has_campus_address(X).
X = li_ying
```

불행히도 `cs254`를 수강하는 2학년 학생을 이름으로 식별할 수 없으며, `li_ying`의 주소도 식별할 수 없다.

## ✓ 확인문제

15. 술어 연산에서 절 형태의 개념을 정의하라.
16. 임의의 술어 연산 문장을 절 형태로 변환하는 절차를 간단히 설명하라.
17. 프롤로그에서 표현할 수 없는 절 형태 문장의 특성을 기술하라.
18. 스콜렘화란 무엇인가? 스콜렘 상수와 스콜렘 함수 사이의 차이를 설명하라.
19. 스콜렘화는 어떤 상황에서 프롤로그로 유용하게 표현할 수 있는 절 형태의 생성에 실패할 수 있는가?