논•문 14-2-3

가언적 규칙베이스를 위한 고차선형논리언어 설계
(A Design of Higher-Order Linear Logic Language for Hypothetical Rulebases)

배 민 오
동덕여자대학교 전자계산학과
bai@dongduk.ac.kr

요 약

논리는 데이터 모델링을 위한 유용한 지식 표현 언어이다. 가언적 규칙 베이스에서는 사용자가 규칙 베이스의 현상태와 상관없이 가설을 형성하여 추론할 수 있는 시스템이다. 가언적 규칙 베이스를 논리 언어로 모델링하기 위해 동상적으로 직관적 논리가 이용된다. 직관적 논리는 가언적 삽입을 표현할 수는 있지만 가언적 삭제를 가언적 삽입과 함께 표현하지는 못한다. 본 연구에서는 고차 유전적 해럽식에 Girard의 선형 논리 연산자를 추가하여 고차 선형 유전적 해럽식으로 확장하였다. 유전적 해럽 R-식으로 규칙 베이스의 규칙들을 표현하고 유전적 G-식으로 규칙 베이스의 질의 및 데이터 조작 명령을 표현할 수 있다. 따라서 본 연구의 언어를 이용하면 가언적 삽입 및 삭제를 포함하는 규칙 베이스의 질의뿐만 아니라 데이터 갱신, 삽입, 삭제와 같은 데이터 조작 명령도 한 개의 언어로 통합적으로 균일하게 모델링할 수 있다.

1. 서론

대부분의 연역데이터 베이스에서는 가언적으로 추론할 수 없다. 즉 현상 그대로는 효과적으로 추론할 수 있지만 가언적 동작과 가능성에 대해서 추론해야 하는 계획 및 설계와 같은 기능은 약할 수 밖에 없다. 이와 같은 제한을 완화하기 위해 Bonner는 사용자가 가설을 형성하여 이 가설로부터 추론할 수 있는 논리 언어를 제시하였다[1]. 이미 정립된 직관적 논리로 삽입을 설명할 수 있기 때문에 대부분의 전문가들은 가언적 삽입에 대해서는 많이 연구하였다. 반면에 Bonner는 삽입 뿐만 아니라 가언적 삭제도 행할 수 있는 언어를 제시하였다. 그러나 Bonner는 가언적 삭제를 직관적 논리를 이용하여 구현하기 때문에 삭제의 논리적 의미를 가언적 삽입과 실패적 부정 규칙을 이용하여 우회적으로 모델링하고 있다[1].

본 논문에서는 가언적 질의를 표현하려고 하는 직관적 논리의 한계에 대해서 논하고 [1]의 질의 언어를 선형 논리의 연산자를 추가하여

확장한다. 따라서 본 언어는 선형논리[4]에 근거하고 있으며 자연적으로 선형 논리적 의미를 갖게 된다.

2. 가언적 규칙베이스

술어(predicate)를 기초 술어, 유도 술어, 질의 술어로 나눈다. R을 기초 및 유도 술어들로 구축된 규칙들의 집합이고 DB는 기초 술어의 바탕 아톰식(ground atomic formula)들의 집합이라면, 연역 데이터 베이스는 쌍 [R,DB]로 정의할 수 있다. 여기서 질의 술어는 연역 데이터 베이스의 질의를 표현하는 논리식을 구축할 때만 이용할 수 있다. 규칙은 논리 프로그램 절의 형태를 갖고 있으며 구체적인 형태는 5절에서 논한다.

가언적 규칙 베이스의 질의는 술어를 이용하여 논리식으로 표현한다. [1]에서는 A와 B가 아톰식일 때 가언적 질의를 `A[ add: B]` 또는 `A[ del: B]`의 형태로 표현한다. 이들의 의미는 가언적으로 `B`를 DB에 추가하거나 삭제한 후 `A`를 추론할 수 있는 지를 질의하는 것이다. 따라서 이들을 위한 추론 규칙은 다음과 같다.

$$\frac{R, DB + B \vdash A}{R, DB \vdash A[add: B]} \qquad \frac{R, DB - B \vdash A}{R, DB \vdash A[del: B]}$$

그림 1. 가언적 질의의 추론 규칙

따라서 가언적 질의를 논리 연산자를 통해서 논리적으로 기술한 것은 아니고 결과론적인 동작적 의미를 이용하여 기술한 것이라고 할 수 있다.

3. 단순 타입이론과 선형 논리언어

본 논문에서 사용하는 논리들의 문법으로서는 단순 타입 이론(simply typed theory) [2, 5]의 문법을 사용한다. 따라서 논리는 타입과 단순 타입적인 λ-항으로 구성된다. 타입의 집합은 일련의 기초 타입들을 가지고 있으며 함수 형성에 대해 닫혀있다. 즉 α와 β가 타입이면 $\alpha \to \beta$도 타입이다. 타입구축자 $\to$는 오른쪽으로 결합한다. c가 토큰이고 τ가 (기본 타입들의 고정된 집합에서의) 단순 타입이라면 시그니처(signature) Σ는 쌍 $c:\tau$ 들의 유한집합이다. 한 시그니처에서는 주어진 토큰은 한 개의 타입만을 가질 수 있다. 단순 타입적 λ-항들은 이들 상수와 변수들을 추상(abstraction)과 적용(application)이라는 두 가지 연산에 의해 구축된다. 적용은 왼쪽으로 결합된다. 우리는 먼저 독자들이 [6]에 있는 언어의 대입(substitution)과 λ-변환(conversion)에 관한 기초적인 개념과 정의들을 이해하고 있다고 가정한다. 다만 몇 가지는 아래에서 소개하고자 한다. 두 항이 속박 변수(bound variable)들의 글자만이 다른 경우는 동치이다. 어떤 항이 β-리덱스(즉 $(\lambda x.B)C$ 형태의 부분항)가 없으면 λ-노멀 형태로 되어 있다고 한다. 모든 항은 유일한 λ-노멀 항으로 β-변환이 가능하다. 항 t에 해당되는 유일한 λ-노멀 항을 $\lambda norm(t)$로 표현한다. $B[C/x]$는 B의 모든 자유로운 x 대신 C를 대입한 결과를 표시하며 $\lambda norm((\lambda x.B)C)$로 정의할 수 있다. 어떤 단순 타입적 λ-항 t를 만약 t 내의 모든 비논리기호가 Σ에서 타입을 갖고 있는 경우 Σ-항이라고 한다.

이와 같은 항의 구조에 논리를 포함하기 위해서는 다음과 같이 하면 된다. 먼저 명제(proposition) 타입으로 o를 기초 타입에 포함시켜서 다음과 같은 상수들을 논리적 상수로서 가지게 하면 된다. 예를 들면 이항(binary) 논리 상수들의 타입은 $o \to o \to o$이다. 따라서 $\wedge, \vee, \supset$의 타입은 $o \to o \to o$이다. $\top, \bot$의 타입은 o이다. 모든 타입 τ에 대해서 수량사 $\forall_\tau$

와 $\exists_\tau$의 타입은 $(\tau\to o)\to o$이다. $\forall_\tau\lambda x.B$와 $\exists_\tau\lambda x.B$같은 형태의 표현들은 각각을 $\forall_\tau x.B$와 $\exists_\tau x.B$와 같이 생략하여 쓰거나, 타입 τ가 중요하지 않거나 문맥상 유추할 수 있는 경우에는 $\forall x.B$와 $\exists x.B$같이 생략할 수 있다. 타입이 o인 Σ-항 B를 Σ-식(formula)이라고 부른다. 또한 우리는 Girard 의 논리 연산자를 Church의 단순 타입 이론의 항 구조와 수량사에 포함하여 이용한다. 선형 논리[4] 연산자와 함께 직관적 함의를 나타내기 위해 삽입 기호 ⇒를 추가하여 사용한다. 즉 $B\Rightarrow C$는 $!B\multimap C$ 와 동치이다. 표현 $B\equiv C$는 식 $(B\multimap C)\&(C\multimap B)$를 간략히 나타낸 것이다. 만약 이 식이 선형 논리에서 증명 가능하면 B와 C는 논리적으로 동치라고 한다.

λ-노멀 식은 밖에서부터 보면 여러 개의 적용으로 이루어져 있으며 이들 기호중 제일 왼쪽에 있는 것을 최상위 기호(top-level symbol)라고 한다. 일차 논리식은 λ-노멀 식으로서 기초타입의 변수들과 비논리 상수(타입이 $\alpha_1\to\cdots\to\alpha_n\to\alpha_0$ 이고 $\alpha_1,\cdots,\alpha_n$는 o가 아닌 기초타입이고 α_0는 기초타입이다.) 만을 이용하여 얻어진 것이다.

본 논문에서 다음과 같은 의미를 갖는 문법적 변수들을 사용한다.

$\mathfrak{D}$	어떤 논리 프로그래밍 언어의 가능한 프로그램절로 사용될 수 있는 식들의 집합.
$\mathfrak{G}$	이 프로그래밍 언어의 가능한 질의(query) 또는 목표(goal)가 될 수 있는 식들의 집합.
A	아톰 식(atomic formula). 즉 최상위 기호가 논리 상수가 아닌 식. $\top$와 $\bot$는 아톰 식이 아니다.
A_r	강체(rigid) 아톰 식. 즉 최 상위 기호가 변수가 아닌 아톰 식.
D	$\mathfrak{D}$ 의 원소. 프로그램 절이라고 한다.
G	$\mathfrak{G}$의 원소. 목표 또는 질의라고 한다.
Γ	$\mathfrak{D}$의 유한 부 집합. (논리) 프로그램이라고 한다.
Δ	$\mathfrak{D}$의 유한 부 집합. 선형 문맥이다.

4. 논리 프로그래밍 언어 설계

모든 논리가 논리 프로그래밍 언어로써 적합하다고 할 수는 없다. 호언 절(Horn clause)과 같은 약한 논리는 그렇다고 말할 수는 있겠지만 다른 여러 종류의 논리들은 일반적으로는 그렇지가 못하다. 어떤 의미로는 논리 프로그래밍이 논리적 문법과 의미를 사용하는 만큼 목표 지향적 탐색(goal directed search)을 충분히 지원해야 한다. 예를 들면 일차 논리 전체에는 이 목표 지향적 탐색을 행할 수 없다. Miller등은 [9]에서 이 목표 지향적 탐색을 유니폼 시퀜트 증명(uniform sequent proof)이라는 개념을 이용하여 정립하였다. 여기에서는 유니폼 증명을 정의하고 호언 절보다는 충분히 큰 직관적 논리에 근거한 고차 논리 프로그래밍 언어를 제시하고자 한다. 다음절에서는 이 언어를 선형 논리 연산자를 이용하여 확장하고자 한다.

Miller등은 논리 프로그래밍에서의 계산은 어떤 종류의 무절단(cut-free) 시퀜트[3] 증명의 탐색과 동일시된다고 하였다. 이런 관점에서는 시퀜트 $\Sigma;\Gamma\to G$는 Σ-식으로 이루어 진 프로그램 Γ로부터 Σ-식 G가 따른다는 것을 결정하려고 하는 해석기의 상태로 볼 수 있다. 동작적으로는 목표 지향적 탐색은 우측 소개 규칙을 먼저 적용하고 좌측 소개 규칙은 우측이 아톰일 때만 적용하는 증명의 상향식 구축으로 특징 지울 수 있다. 이것은 목표 내의 논리연산자들이 프로그램과는 상관없이 유니폼하게 분해되는 것이라고 말할 수 있다. 또 프로그램은 목표식이 아톰일 때만 고려된다. 이러한 개념

은 단결론 시퀜트 시스템에서 다음과 같이 정의할 수 있다[7].

정의 1. 어떤 무절단 시퀜트 증명에서 우측이 아톰이 아닌 모든 시퀜트가 우측 소개 규칙의 결론이면 이 증명을 *유니폼 증명*이라고 한다.

정의 2. $\mathfrak{D}$와 $\mathfrak{G}$를 논리식의 집합이라고 하자. 모든 유한 집합 $\Gamma \subseteq \mathfrak{D}$와 모든 $G \in \mathfrak{G}$에 대해서 시퀜트 $\Sigma : \Gamma \rightarrow G$가 증명 시스템 $\vdash$에서 증명을 갖기 위한 필요충분 조건이 이 시스템에서 유니폼 증명을 갖는 것일 경우, $\langle \mathfrak{D}, \mathfrak{G}, \vdash \rangle$가 *추상 논리 프로그래밍 언어*(abstract logic programming language)라고 한다.

$$\frac{}{\Sigma : \Gamma, B \rightarrow B}\ identity \qquad \frac{}{\Sigma : \Gamma \rightarrow true}\ true R$$

$$\frac{\Sigma : \Gamma, B_1, B_2 \rightarrow C}{\Sigma : \Gamma, B_1 \wedge B_2 \rightarrow C}\ \wedge L \qquad \frac{\Sigma : \Gamma \rightarrow B \quad \Sigma : \Gamma \rightarrow C}{\Sigma : \Gamma \rightarrow B \wedge C}\ \wedge R$$

$$\frac{\Sigma : \Gamma \rightarrow B \quad \Sigma : \Gamma, C \rightarrow E}{\Sigma : \Gamma, B \supset C \rightarrow E}\ \supset L \qquad \frac{\Sigma : \Gamma, B \rightarrow C}{\Sigma : \Gamma \rightarrow B \supset C}\ \supset R$$

$$\frac{\Sigma : \Gamma, B[t/x] \rightarrow C}{\Sigma : \Gamma, \forall x.B \rightarrow C}\ \forall L \qquad \frac{y : \tau, \Sigma : \Gamma \rightarrow B[y/x]}{\Sigma : \Gamma \rightarrow \forall x.B}\ \forall R$$

그림 2. 고차 직관적 논리를 위한 증명 시스템 $\mathfrak{I}$

정의 2에서 집합 $\mathfrak{D}$는 프로그램이 될 수 있는 논리식의 클라스를 나타내고 집합 $\mathfrak{G}$는 목표들의 클라스로 생각할 수 있다. 논리 연산자 $true, \wedge, \supset, \forall$에 대한 고차 직관적 증명 시스템 $\mathfrak{I}$를 그림 2에 나타내었다. 여기에서 시퀜트 좌측은 집합이므로 교환 및 축약과 같은 구조 규칙은 필요하지 않다. 또한 *identity* 규칙에 이미 박약(weakening)이 포함되어 있어 별도의 박약 규칙은 필요치 않다. 모든 논리 식들이 λ-노멀 형태로 사용되고 있다고 가정하면 [2, 5]에서 사용되는 λ-변환을 위한 별도의 규칙은 필요하지 않다. $\forall R$ 규칙이 적용되기 위한 가정은 $y : \tau \notin \Sigma$이고 $y : \tau, \Sigma$의 의미는 $\{y : \tau\} \cup \Sigma$이다. 그림 3에는 $\mathfrak{I}$에서의 절단 규칙을 나타내었다. 시퀜트 $\Sigma : \Gamma \rightarrow G$의 $\mathfrak{I}$-증명이 존재하면 $\Sigma : \Gamma \vdash_{\mathfrak{I}} G$로 표현한다.

$$\frac{\Sigma : \Gamma' \rightarrow B \quad \Sigma : \Gamma, B \rightarrow C}{\Sigma : \Gamma' \rightarrow C}\ cut \qquad \Gamma \subseteq \Gamma' \text{이면}$$

그림 3. $\mathfrak{I}$에서의 절단 규칙

정리 1. $\aleph_0$가 논리 상수 $true, \wedge, \supset, \forall$를 사용하는 모든 논리식들의 집합이라면 $\langle \aleph_0, \aleph_0, \vdash_{\mathfrak{I}} \rangle$는 논리 프로그래밍 언어이다.

이 논리에서 유니폼 증명을 좀 더 제한하더라도 완전성을 유지할 수 있는 방법이 있다. 정리 1의 증명에서 좌측 소개 규칙은 역 연쇄 규칙을 지원하기 위해서 필요하다. 이 것을 다음 두 가지로 설명할 수 있다. 첫째, 역 연쇄는 여러 개의 좌측 규칙의 합성이다. 둘째, 아톰을 증명하려고 할 때 이 아톰 식의 부 증명을 제공하기 위해 처리될 수 있는 어떤 식이 좌측에 있어야 한다. 이와 같은 방법을 다음에 제시하는 증명 시스템에서 적용하였다.

B를 논리 연산자 $true, \wedge, \supset, \forall$을 사용하는 Σ-식이라면 $|B|$는 다음 조건을 만족하는 쌍의 최소 집합이 된다.

1. $\langle \emptyset, B \rangle \in |B|$
2. $\langle \Delta, B_1 \wedge B_2 \rangle \in |B|$ 이면 $\langle \Delta, B_1 \rangle, \langle \Delta, B_2 \rangle \in |B|$
3. $\langle \Delta, \forall x.B' \rangle \in |B|$ 이면 모든 닫힌 Σ-항 t에 대해, $\langle \Delta, B'[t/x] \rangle \in |B|$
4. $\langle \Delta, G \supset B' \rangle \in |B|$ 이면 $\langle \Delta \cup \{G\}, B' \rangle \in |B|$

위의 정의를 직관적으로 해석하면 다음과 같다. $\langle \Delta, A \rangle \in |B|$이면 Δ에 있는 모든 식들이 성립하면 A가 성립한다는 것을 증명하는데 Σ-식 B를 사용할 수 있다. 그림 1의 $identity, \supset L, \wedge L, \forall L$규칙들을 그림 4의 역 연쇄 규칙(backchaining rule)으로 대체한 증명

시스템을 $\mathcal{G}'$이라고 하자.

$$\frac{\Sigma;\Gamma\to G_1 \quad \cdots \quad \Sigma;\Gamma\to G_n}{\Sigma;\Gamma\to A}\,BC$$

A는 아톰 Σ-식이고, $B\in\Gamma$이며, $\langle\{G_1,\cdots,G_n\},A\rangle\in|B|$이다.

그림 4. $\mathcal{G}$에서의 역 연쇄 규칙

정리 2. $\Gamma\cup\{B\}$를 $true,\wedge,\supset,\forall$를 사용하는 Σ-식들의 집합이라고 하자. 시퀀트 $\Sigma;\Gamma\to G$가 $\mathcal{G}$-증명을 가질 필요 충분조건은 이 시퀀트가 $\mathcal{G}'$-증명을 갖는 것이다.

$\mathcal{G}'$에는 한 개만의 좌측 규칙 BC만이 있고 BC 규칙이 우측에 아톰이 있는 시퀀트에만 적용되기 때문에 $\mathcal{G}'$-증명은 반드시 유니폼하다.

전 직관적 논리가 유니폼하지 않은 이유는 이접과 실재적 수량사의 좌측 규칙 적용 순서를 빠꿀 수 없기 때문이다. 따라서 이접과 실재적 수량사를 왼쪽에 소개하지 않는다면 사용할 수 있다. 이렇게 하기 위해서는 증명에서 이들을 양의 경우에만 허용하면 된다.

$\mathcal{D}_0$와 $\mathcal{G}_0$를 다음의 순환에 의해 정의되는 D 및 G-식이라고 하자.

$$D::=true \mid A_r \mid D_1\wedge D_2 \mid G\supset D \mid \forall x.D$$

$$G::=true \mid A \mid G_1\wedge G_2 \mid D\supset G \mid \forall x.G \mid G_1\vee G_2 \mid \exists x.G$$

$\mathcal{G}$-증명 시스템을 $\vee$와 $\exists$의 소개 규칙을 추가하여 확장한 $\langle\mathcal{D}_0,\mathcal{G}_0,\vdash_{\mathcal{G}}\rangle$는 추상 논리 프로그래밍 언어이다. 이 언어는 [9]의 고차 유전적 해럽식(higher-order hereditary Harrop formula)이라는 것이다. 고차 유전적 해럽 식의 결론(함의의 우측에 있는 식)이 강체 아톰 식이어야 한다는 제한한 동기는 다음과 같다. 첫째, 프로그램 절에 부여되는 동작적 해석에 의하면 함의의 결론에 있는 아톰의 술어 기호는 이 절이 정의하려고 하는 프로시저의 이름이다. 따라서 이 술어 기호를 상수로 제한하므로서 각 함의가 어떤 프로시저의 정의의 일부분이 되게 한다. 둘째, 이렇게 제한함으로서 프로그램 절들의 모든 집합이 일관(consistent)되게 된다. 예를 들면 이 제한을 완화하면 모든 Σ-식 B에 대해서 시퀀트 $\Sigma;p,\forall x.(p\supset x)\to B$를 증명할 수 있으므로 프로그램 $\{p,\forall x.(p\supset x)\}$는 일관되지 못하다(inconsistent).

5. 고차 선형 논리 프로그래밍 언어

$\mathcal{G}'$-증명은 다음의 성질이 성립한다고 할 수 있다. 두 개의 시퀀트 $\Sigma;\Gamma\to B$와 $\Sigma';\Gamma'\to B'$가 증명 상에서 같은 경로 상에 있고 $\Sigma;\Gamma\to B$가 종 시퀀트에 더 가깝게 있다면 $\Sigma\subseteq\Sigma'$이고 $\Gamma\subseteq\Gamma'$이다. 따라서 어떤 계산에서 증명을 밑에서 위로 상향으로 구축해 갈 때 시퀀트의 좌측이 줄어드는 경우는 없다. 이 변화의 가능성에 대한 제한은 직관적 문맥은 논리 프로그램이 사용하려는 여러 가지 바람직한 문맥으로서는 너무 간단하다. 문맥이 비축소적이기 때문에 이 안에 있는 논리식에 BC 규칙을 몇 번을 사용해도 된다. 반면에 선형 논리는 제한적 자원과 무제한 자원이라는 개념을 도출하기에 적합한 환경을 제공한다.

$$\frac{}{\Sigma;B\to B}\,identiiy \qquad \frac{}{\Sigma;\Delta\to\top}\,\top R$$

$$\frac{\Sigma;\Delta,B_i\to C}{\Sigma;\Delta,B_1\&B_2\to C}\,\&L \qquad \frac{\Sigma;\Delta\to B \quad \Sigma;\Delta\to C}{\Sigma;\Delta\to B\&C}\,\&R$$

$$\frac{\Sigma;\Delta_1\to B \quad \Sigma;\Delta_2,C\to E}{\Sigma;\Delta_1,\Delta_2,B\multimap C\to E}\,\multimap L \qquad \frac{\Sigma;\Delta,B\to C}{\Sigma;\Delta\to B\multimap C}\,\multimap R$$

$$\frac{\Sigma;\Delta,B_1,B_2\to C}{\Sigma;\Delta,B_1\otimes B_2\to C}\,\otimes L \qquad \frac{\Sigma;\Delta_1\to B \quad \Sigma;\Delta_2\to C}{\Sigma;\Delta_1,\Delta_2\to B\otimes C}\,\otimes R$$

$$\frac{\Sigma;\Delta\to C}{\Sigma;\Delta,!B\to C}\,!W \qquad \frac{\Sigma;\Delta,!B,!B\to C}{\Sigma;\Delta,!B\to C}\,!C$$

$$\frac{\Sigma;\Delta,B\to C}{\Sigma;\Delta,!B\to C}\,!D \qquad \frac{\Sigma;!\Delta\to B}{\Sigma;!\Delta\to !B}\,!R$$

$$\frac{\Sigma;\Delta,B[t/x]\to C}{\Sigma;\Delta,\forall x.B\to C}\,\forall L \qquad \frac{y:\tau,\Sigma;\Delta\to B[y/x]}{\Sigma;\Delta\to\forall x.B}\,\forall R$$

그림 5. 고차 선형 논리를 위한 시스템 LL

앞절에서 기술한 논리 언어를 제공하기 위해

서는 이 절에서는 선형 논리 연산자 $\top, \&, \otimes, \multimap, \forall$ 를 고려한다. 이들 연산자를 위한 증명 규칙들은 그림 5에 표현하였다.

$$\frac{\Sigma;\Delta \to B \quad \Sigma;\Delta', B \to C}{\Sigma;\Delta, \Delta' \to C}\ cut$$

그림 6. *LL*에서의 절단 규칙

또 이 증명 시스템에 대한 절단 규칙을 그림 6에 나타내었다. 여기에서 시퀜트 왼쪽은 다집합(multiset)이다. 따라서 교환을 위한 구조 규칙은 명시적으로 언급할 필요는 없다. 축약 및 박약(weakening)을 위한 구조 규칙은 축약을 위해 $!C$ 규칙이 박약을 위해 $!W$ 규칙이 주어졌다. 이들 규칙은 $!B$ 형태의 식들에만 적용가능하다. 여기서 우리는 연산자 $!$을 전형(exponential)이라고 하고 $!B$형태의 식을 전형식이라고 한다. $!\Delta$는 다집합 $\{!C \mid C \in \Delta\}$를 표시한다. 또 $\forall R$ 규칙이 적용되기 위한 조건은 앞절의 $\forall R$규칙과 동일하다. 시퀜트 $\Sigma;\Delta \to B$가 그림 4의 증명 시스템에서 증명을 가질 때 우리는 $\Sigma;\Delta \vdash_{LL} B$ 로 쓴다. 그림 5의 모든 시퀜트는 단결론만을 가지므로 우리는 직관적 부분에서만 연구한다고 말할 수 있다.

선형 논리는 여기에서 고려한 논리연산자에 대해서도 논리 프로그래밍 언어가 될 수 없음을 쉽게 알 수 있다. 예를 들면 시퀜트 $a \otimes b \to b \otimes a$, $!a \to !a \otimes !a$, $!a \& b \to !a$, $b \otimes (b \multimap !a) \to !a$들은 선형 논리에서는 증명가능 하지만 유니폼 *LL*-증명은 갖지 못한다. 여기에서의 문제점은 $\otimes R$과 $!R$규칙들이 모든 좌측 규칙들을 뛰어 넘어 내려가지 못하는 데에 있다. 이러한 이유로 여기에서는 연산자로 !와 $\otimes$를 가지고 있지 않은 선형 논리 부분만을 고려한다. 이렇게 하기 위해서 그림 5에 주어진 논리에 두 가지 변화를 주고자 한다. 먼저 시퀜트를 $\Sigma;\Gamma;\Delta \to B$와 같은 형태로 표현한다. 여기에서 Γ는 Σ-식들의 집합이고 Δ는 Σ-식들의 다집합이다. 이러한 형태의 시퀜트는 문맥이 두 부분으로 나누어져 있다. 직관적 시퀜트의 좌측에 해당하는 무제한 부분 Γ와 전형 $!$가 없는 순수 선형 논리 부분에서 시퀜트 좌측에 해당되는 Δ이다. 축약과 박약은 무제한 문맥에서는 허용되지만 제한 문맥에서는 허용되지 않는다. 따라서 시퀜트 $\Sigma;B_1, \cdots, B_n; C_1, \cdots, C_m \to B$은 선형 시퀜트로

$$\Sigma;!B_1, \cdots, !B_n, C_1, \cdots, C_m \to B$$

와 같이 표현할 수 있다. 위와 같은 스타일로 시퀜트를 표현한다고 가정하면 선형 논리에 다음과 같이 두 가지의 함의를 추가하므로서 두 번째의 변화를 꾀할 수 있다. 우측 소개 규칙에서 가정을 제한 문맥에 첨가하는 선형 함의($\multimap$)와 가정을 무제한 문맥에 첨가하는 직관적 함의($\Rightarrow$)가 있다. 물론 $B \Rightarrow C$의 의미는 $(!B) \multimap C$이다.

여기까지는 논리연산자 $\multimap$와 $\Rightarrow$의 동기에 대해서만 언급하였다. 제한 논리식들이 아톰인 시퀜트를 고려해 보자. 만약 논리 연산자가 $\multimap$와 $\Rightarrow$뿐이라면 제한 문맥에 있는 모든 식들이 정확히 한번 사용되어야 한다. 즉 이들은 시퀜트의 우측에 있는 동일한 식과 매치 시키는 *identity* 규칙에서 사용되어야 한다. 예를 들면 만약 데이터 베이스를 제한 문맥에 보관한다면 한 항목에 대한 질의를 하면 이 항목은 다른 곳에서는 사용 불가능하게 될 것이다. 또 이 데이터 베이스에서의 계산이 끝나려면 모든 항목을 이와 같이 읽어 내어야 할 것이다. 따라서 연산자 $\top$와 $\&$를 추가하므로써 $\top$를 사용하여 제한 문맥의 일부분을 지우거나 $\&$을 사용하여 복제할 수 있다. 따라서 데이터 베이스에서 비파괴적으로 읽어 내기 위해서는 먼저 이 데이터 베이스의 복사본을 만들어서 한 항목을 파괴적으로 읽어 내고 나머지는 삭제하면 된다. 이 과정에서 원본은 변하지 않는다.

$$\frac{}{\Sigma\,\Gamma;A\to A}\ identity \qquad \frac{\Sigma\,\Gamma,B;\Delta,E\to C}{\Sigma\,\Gamma,B;\Delta\to C}\ absorb$$

$$\frac{}{\Sigma\,\Gamma;\Delta\to\top}\ \top R \qquad \frac{\Sigma\,\Gamma;\Delta,B_i\to C}{\Sigma\,\Gamma;\Delta,B_1\&B_2\to C}\ \&L$$

$$\frac{\Sigma\,\Gamma;\Delta\to B \quad \Sigma\,\Gamma;\Delta\to C}{\Sigma\,\Gamma;\Delta\to B\&C}\ \&R$$

$$\frac{\Sigma\,\Gamma;\Delta_1\to B \quad \Sigma\,\Gamma;\Delta_2,C\to E}{\Sigma\,\Gamma;\Delta_1,\Delta_2,B-C\to E}\ -L$$

$$\frac{\Sigma\,\Gamma;\Delta,B\to C}{\Sigma\,\Gamma;\Delta\to B-C}\ -R \qquad \frac{\Sigma\,\Gamma;\varnothing\to B \quad \Sigma\,\Gamma;\Delta,C\to E}{\Sigma\,\Gamma;\Delta,B\Rightarrow C\to E}\ \Rightarrow L$$

$$\frac{\Gamma,B;\Delta\to C}{\Gamma;\Delta\to B\Rightarrow C}\ \Rightarrow R \qquad \frac{\Sigma\,\Gamma;\Delta,B[t/x]\to C}{\Sigma\,\Gamma;\Delta,\forall x.B\to C}\ \forall L$$

$$\frac{y:\tau,\Sigma\,\Gamma;\Delta\to B[y/x]}{\Sigma\,\Gamma;\Delta\to\forall x.B}\ \forall R$$

그림 7. 증명 시스템 $\mathcal{L}$

그림 7은 논리 연산자 $\top,\&,-,\Rightarrow,\forall$을 위한 증명 시스템 $\mathcal{L}$이고 그림 8은 $\mathcal{L}$에서의 절단 규칙이다. 우리는 시퀀트 $\Sigma\,\Gamma;\Delta\to B$가 $\mathcal{L}$에서 증명을 가지면 $\Sigma\,\Gamma;\Delta\vdash_{\mathcal{L}} B$와 같이 쓴다.

$$\frac{\Sigma\,\Gamma';\Delta_1\to B \quad \Sigma\,\Gamma;\Delta_2,B\to C}{\Sigma\,\Gamma';\Delta_1,\Delta_2\to C}\ cut$$

$$\frac{\Gamma';\varnothing\to B \quad \Gamma,B;\Delta\to C}{\Gamma';\Delta\to C}\ cut!$$

그림 8. $\mathcal{L}$의 두 가지 절단 규칙

정리 3. B를 Σ-식, Γ를 Σ-식들의 집합, Δ를 Σ-식들의 다집합이라고 하고 이들에 들어 있는 모든 Σ-식들이 $\top,\&,-,\Rightarrow,\forall$만 사용한다고 하자. B°를 B내에 있는 모든 $C_1\Rightarrow C_2$를 $(!C_1)-C_2$로 치환한 결과라고 하자. 그러면 $\Sigma\,\Gamma;\Delta\vdash_{\mathcal{L}} B$일 필요충분 조건은 $\Sigma\,!(\Gamma^{\circ}),\Delta^{\circ}\vdash_{LL} B^{\circ}$이다.

정리 4. 시퀀트 $\Sigma\,\Gamma;\Delta\to B$가 $\mathcal{L}$-증명을 가질 필요충분 조건은 유니폼 $\mathcal{L}$-증명을 갖는 것이다.

N_I을 $\top,\&,-,\Rightarrow,\forall$을 사용하는 모든 논리식들의 집합이라고 하자. 그러면 정리 4에 의해 $<N_I, N_I, \vdash_{\mathcal{L}}>$이 논리 프로그래밍 언어가 된다. 여기에서 N_I에 있는 논리식들은 제한 및 무제한 문맥에 있을 수 있다고 가정한다.

시스템 $\mathcal{G}'$과 같이 좌측 규칙들의 적용을 역연쇄 형태로 제한한 유니폼 증명으로 제한할 수도 있다. 다음 정의를 보자. B를 $\top,\&,-,\Rightarrow,\forall$만을 논리 연산자로 가지고 있는 논리식이라고 하자. Γ를 Σ-식들의 집합, Δ를 Σ-식들의 다집합이라면 $||B||$는 다음의 조건을 만족하는 $<\Gamma,\Delta,B'>$형태의 원소들의 최소 집합이다.

1. $<\varnothing,\varnothing,B>\in||B||$
2. $<\Gamma,\Delta,B_1\wedge B_2>\in||B||$이면
   $<\Gamma,\Delta,B_1>,<\Gamma,\Delta,B_2>\in||B||$
3. $<\Gamma,\Delta,\forall x.B'>\in||B||$이면 모든 닫힌 Σ-항 t에 대해 $<\Gamma,\Delta,B'[t/x]>\in||B||$
4. $<\Gamma,\Delta,B_1\Rightarrow B_2>\in||B||$이면
   $<\Gamma\cup\{B_1\},\Delta,B_2>\in||B||$
5. $<\Gamma,\Delta,B_1-B_2>\in||B||$이면
   $<\Gamma,\Delta\uplus\{B_1\},B_2>\in||B||$

5에서의 $\uplus$는 다집합의 합집합 연산자이다. $\mathcal{L}'$을 그림 7의 $identity,-L,\Rightarrow L,\&L,\forall L$ 규칙들을 그림 9의 역연쇄 규칙으로 치환한 증명 시스템이라고 하자.

$$\frac{\Sigma\,\Gamma;\to B_1 \ \cdots\ \Sigma\,\Gamma;\to B_n \quad \Sigma\,\Gamma;\Delta_1\to C_1 \ \cdots\ \Sigma\,\Gamma;\Delta_m\to C_m}{\Sigma\,\Gamma;\Delta_1,\cdots,\Delta_m,B\to A}\ BC$$

$n,m\geq 0$, A는 아톰이고, $<\{B_1,\cdots,B_n\},\{C_1,\cdots,C_m\},A>\in||B||$

그림 9. $\mathcal{L}$을 위한 역연쇄 규칙

정리 5. 시퀀트 $\Sigma\,\Gamma;\Delta\to B$가 L-증명을 갖기 위한 필요충분 조건은 $\mathcal{L}'$-증명을 갖는 것이다.

$\mathcal{G}'$시스템과는 다르게 $\mathcal{L}'$-증명은 *absorb* 규칙 때문에 반드시 유니폼한 것은 아니다. 이 규칙은 아톰이 아닌 우측이 있는 시퀀트에 적용될 수도 있다. 그럼에도 불구하고 이 규칙의 모든 적용은 증명 트리에서 BC 규칙 바로 밑에까지 밀어 올릴 수 있다. 이와 같은 $\mathcal{L}'$-증명은 유니폼하다.

무절단 증명만이 의미가 있기 때문에 시퀀트

좌우에 서로 다른 집합의 식들이 있게 할 수 있다. 논리 연산자로써 $\top, 1, \&, \otimes, \oplus, -\!\circ, \Rightarrow, \forall, \exists$를 가지고 있는 논리식들의 다음과 같은 두 가지 클래스의 정의를 보자.

$$R ::= \top \mid A_r \mid R_1 \& R_2 \mid G -\!\circ R \mid G \Rightarrow R \mid \forall x.R$$

$$G ::= \top \mid A \mid G_1 \& G_2 \mid R -\!\circ G \mid R \Rightarrow G \mid \forall x.G \mid G_1 \oplus G_2 \mid 1 \mid G_1 \otimes G_2 \mid !G \mid \exists x.G$$

여기서 자원식(resource formula)이라고 하는 R-식들은 시퀜트의 좌측의 제한이든 무제한이든 간에 어느 문맥에도 있을 수 있다. 반면에 목표식이라고 하는 G-식들은 시퀜트 우측에만 있을

$$\frac{}{\Sigma\Gamma;\varnothing\to 1}\,1R \qquad \frac{\Sigma\Gamma;\varnothing\to B}{\Sigma\Gamma;\varnothing\to !B}\,!R$$

$$\frac{\Sigma\Gamma;\Delta\to B_i}{\Sigma\Gamma;\Delta\to B_1\oplus B_2}\,\oplus R \qquad \frac{\Sigma\Gamma;\Delta\to B[t/x]}{\Sigma\Gamma;\Delta\to \exists x.B}\,\exists R$$

$$\frac{\Sigma\Gamma;\Delta_1\to B_1 \quad \Sigma\Gamma;\Delta_2\to B_2}{\Sigma\Gamma;\Delta_1,\Delta_2\to B_1\otimes B_2}\,\otimes R$$

그림 10. $1, \oplus, \otimes, !, \exists$ **의 우측 규칙**

수 있다. 이렇게 확장하였다면 증명 시스템 $\mathcal{L}'$에 $1, \otimes, \oplus, !, \exists$를 위한 우측 규칙들을 추가하여야 한다. 따라서 $\mathcal{L}''$을 그림 10에 있는 우측 규칙들을 $\mathcal{L}'$에 추가한 결과라고 하자. 1은 논리적으로 $!\top$와 동치이기 때문에 그림 10에서 삭제될 수 있다.

6. 유전적 해럽식의 임베딩(embedding)

고차 직관적 선형 논리 언어의 프로그래밍 언어의 신택스를 설계하기 위해서 λProlog와 같은 고차 유전적 해럽식을 다음과 같이 고차 선형 논리의 R과 G-식으로 번역하는 두 함수 $(\cdot)^+$ 와 $(\cdot)^-$을 고려해 보자.

$$(A)^+ = (A)^- = A$$

$$(true)^+ = 1$$

$$(true)^- = \top$$

$$(B_1 \wedge B_2)^+ = (B_1)^+ \otimes (B_2)^+$$

$$(B_1 \wedge B_2)^- = (B_1)^- \& (B_2)^-$$

$$(B_1 \supset B_2)^+ = (B_1)^- \Rightarrow (B_2)^+$$

$$(B_1 \supset B_2)^- = (B_1)^+ -\!\circ (B_2)^-$$

$$(\forall x.B)^+ = \forall x.(B)^+$$

$$(\forall x.B)^- = \forall x.(B)^-$$

$$(B_1 \vee B_2)^+ = (B_1)^+ \oplus (B_2)^+$$

$$(\exists x.B)^+ = \exists x.(B)^+$$

정리 6. $\Sigma\Gamma \vdash_{\mathcal{G}} B$의 필요충분 조건은 시퀜트 $\Sigma\Gamma^-;\varnothing\to B^+$ 가 $\mathcal{L}''$에서 무절단 증명을 갖는 것이다.

호언 절이나 유전적 해럽식의 $\mathcal{G}'$-증명은 정리 6의 결과에 의해서 이들의 번역된 선형 식들의 $\mathcal{L}''$-증명과 같은 것이 된다. 따라서 Prolog나 λProlog 프로그램들이 이 새로운 시스템에 임베드될 때 해석이 변하지 않게 하기 위해서 고차 선형 논리 프로그램의 콘크리트 신택스를 어떻게 설계해야 되는지를 알 수 있다. 예를 들면 Prolog에서

$$\mathtt{A_0 :- A_1, \cdots, A_n}$$

은 다음의 식을 의미한다.

$$(A_1 \wedge \cdots \wedge A_n) \supset A_0$$

음 번역 $(\cdot)^-$을 이용하면 위의 호언 절은 다음의 선형 논리식으로 번역된다.

$$(A_1 \otimes \cdots \otimes A_n) -\!\circ A_0$$

따라서 Prolog의 콤마는 $\otimes$에 해당되고 `:-`는 $-\!\circ$의 역이다.

λProlog에서의 다음의 절을 보자.

```
true (A imp B) :- true A => true B
```

이 절은 다음과 같이 번역된다.

$$(true\ A \Rightarrow true\ B) -\!\circ true\ (A\ imp\ B)$$

따라서 λProlog의 `=>`는 선형 논리에서의 $\Rightarrow$이어야 한다.

본 연구의 논리 프로그래밍 언어의 신택스를 다음 표와 같이 나타내면 기존의 Prolog나 λProlog 프로그램이 문법적 및 동작적으로 동일하게 될 것은 정리 6의 결과에 의해 자명하다.

연산자	음/양	신택스
⊤	+	erase
1	+	one
&	+	&
	-	&
⊗	+	,
⊕	+	;
⊸	+	-o
	-	:-
⇒	+	=>
	-	<=
!	+	bang(…)
∀x.B	+	pi x\B
	-	pi x\B
∃x.B	+	sigma x\B

7. 규칙베이스 질의의 선형 논리 의미론

[R,DB]가 규칙 베이스라면 R은 규칙들의 집합이므로 변하지 않는다. 반면 DB는 규칙 베이스에서 질의 처리가 진행됨에 따라서 변하는 것이므로 R은 선형 시퀜트의 직관적 문맥에 있어야 하고 DB는 선형 문맥내에 있어야 한다. 따라서 시퀜트 $\Gamma;\Delta \rightarrow G$가 규칙 베이스의 질의 처리기의 상태를 나타내는 것이라면 Γ는 R을 포함하는 직관적 문맥이고 Δ는 DB의 현상태를 나타내는 것이다. 한편 add:와 del:이 각각 가언적 삽입 및 삭제 질의 술어라 하고, upd, del, ins를 각각 갱신, 삭제, 삽입 데이터 베이스 명령을 위한 질의 술어라면 이들의 정의를 유전적 해럽 선형 논리식으로 다음과 같이 나타낼 수 있고 이들 규칙들은 Γ에 포함되어 있다고 가정한다.

```
B add: D :- (D -o B)⊗⊤
B del: D :- D⊗B⊗⊤
upd B D G :- B⊗(D-oG)
del B G :- B⊗G
ins B G :- B-oG
```

정리 7. Γ와 Δ는 위와 같고, B와 D는 바탕 아톰식, G는 G-식이라면,

$\Gamma;\Delta \vdash_{\mathcal{L}'}$ B add: D $\Leftrightarrow$ $\Gamma;\Delta, D \vdash_{\mathcal{L}'}$ B

$\Gamma;\Delta$ D$\vdash_{\mathcal{L}'}$ B del: D $\Leftrightarrow$ $\Gamma;\Delta \vdash_{\mathcal{L}'}$ B

$\Gamma;\Delta, B \vdash_{\mathcal{L}'}$ upd B D G $\Leftrightarrow$ $\Gamma;\Delta, D \vdash_{\mathcal{L}'}$ G

$\Gamma;\Delta, B \vdash_{\mathcal{L}'}$ del B G $\Leftrightarrow$ $\Gamma;\Delta \vdash_{\mathcal{L}'}$ G

$\Gamma;\Delta \vdash_{\mathcal{L}'}$ ins B G $\Leftrightarrow$ $\Gamma;\Delta, B \vdash_{\mathcal{L}'}$ G

예 가언적 삭제: 다음과 같이 혼 규칙 한 개로 구성된 규칙 베이스 R을 보자.

```
grad(S)←take(S,his101)⊗take(S,eng101).
```

이 규칙이 의미하는 바는 "어떤 학생 S가 his101과 eng101을 이수하면 졸업한다"이다. "kim이 his250을 이수하지 않았다면 졸업할 수 있는가?"라는 질의를 [1]에서는 식 grad(kim)[del:take(kim,his250)]으로 표현하였고 다음과 같은 성질을 갖는다.

```
R, DB⊢grad(kim)[ del:
take(kim,his250)] ⇔
R, DB-take(kim,his250)⊢grad(kim)
```

B, D, H, E를 각각 바탕 아톰식 grad(kim), take(kim,his250), take(kim,his101), take(kim, eng201)이라고 하면 본 언어에서는 다음 그림 11과 같은 증명을 갖는다.

8. 결론

논리는 데이터 모델링을 위한 유용한 지식 표현 언어이다. 선형 논리는 이미 [5]에서 데이

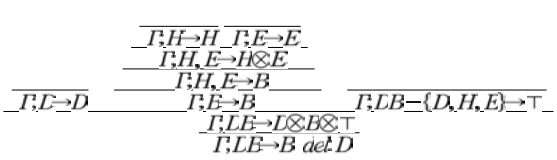

그림 11. 질의의 증명

터 베이스 갱신, 삽입, 삭제와 같은 거래 사건의 의미를 모델링하기 위해 사용되었다. 그러나 여기에서는 질의를 포함하지 않은 거래 사건을 모델링하였기 때문에 거래 사건을 표현하는 논리식이 시퀜트의 좌측에 있다. 한편 [1]에서는 가언적 삭제의 의미를 가언적 삽입과 실패적 부정 규칙의 조합을 이용하여 우회적으로 표현하였다. 본 논문에서는 [5]의 모든 데이터 베이스 거래 사건과 [1]의 가언적 질의를 선형 논리 연산자를 이용하여 모델링하였다.

감사의 글

본 연구는 한국과학재단 핵심 전문 연구(과제번호: 971-0903-024-2)와 특정 기초연구 (과제번호: 2000-1-30300-010-3)의 부분적 지원에 의한 것입니다.

참고문헌

[1]A. J. Bonner. A Logical Semantics for Hypothetical Rulebases with Deletion. *The Journal of Logic Programming*, pp 121-170, 1997.

[2]A. Church. A Formulation of Simple Theory of Types. *The Journal of Symbolic Logic*, 5:56-68, 1940.

[3]G. Gentzen. Investigations into Logical Deductions. In M. E. Szabo, editor, *The Collected Papers of Gerhard Gentzen*, pp 68-131, North-Holland, 1969.

[4]J.-Y. Girard. Linear Logic. *Theoretical Computer Science*, 50:1-102, 1987.

[5]L. Henkin. Completeness of the Theory of Types. *The Journal of Symbolic Logic*, 15:81-91, 1950.

[6]J. R. Hindley, and J. P. Seldin. *Intorduction to Combinators and λ-Calculus*. Cambridge University Press, 1986.

[7]J. S. Hodas. *Logic Programming in Intuitionistic Linear Logic: Theory, Design, and Implementation*. PhD Thesis, University of Pennsylvania, 1994.

[8]D.-T. Lee. Semantic Data Modelling using Linear Logic. *Information Processing Letters*, 60:19-27, 1996.

[9]D. A. Miller, G. Nadathur, F. Pfenning, and A. Scedrov. Uniform Proofs as a Foundation for Logic Programming. *Annals of Pure and Applied Logic*, 51:125-157, 1991.

배민오

1977년~1981년 서울대학교 전자공학과 (학사).

1981년~1983년 한국과학기술원 전산학 (석사).

1988년~1992년 Syracuse University 전산학(박사).

1987년~1992년 Syracuse University T.A.

1993년~1996년 삼성 SDS 정보기술연구소 수석연구원

1996년~현재 동덕여자대학교 조교수

관심분야는 논리프로그래밍, 전자상거래, XML 프로그래밍, 웹 데이타베이스.