LISP의 철학적 기초

이 영 의, 이 초 식
고려대학교 철학과

Philosophical Foundations of LISP

Young-Eui Rhee and Cho-Sik Lee
Department of Philosophy, Korea University

요 약

본 논문은 LISP의 기본적인 특성으로 간주되는 조건 표현과 회기에 대한 철학적 분석이다. 특히 여기서는 논리적 결합사들과 LISP의 원초적 용어들을 비교하고 있다. LISP은 현대 철학자들이 구성했던 이상언어의 사상을 계승한 것으로 여겨진다. 이러한 점에서 인공지능 연구와 철학 간의 학제적 연구가 절실히 필요하다는 논지를 몇가지 검토해 보았다.

1. 서론

LISP은 인공지능 연구에서 가장 널리 이용되고 있는 프로그래밍 언어이다. 1958년 J.McCarthy가 LISP을 만든 이래로 대표적인 전문가 시스템들이 그것을 이용하여 작성되고 있으며, 기호조작과 리스트 처리와 같은 인공지능 언어가 지녀야 할 요건을 훌륭하게 충족시키고 있음이 증명되고 있다. 더구나 LISP 머신(machine)의 등장으로 자동 프로그래밍(automatic programming)이 가능해짐에 따라 LISP의 사용이 보다 용이해지고 있다.

컴퓨터 프로그래밍 언어는 자연언어와 달리 특정 목적을 위해 구문론과 규칙등이 용법에 앞서 인위적으로 구성된 인공언어(artificial language)이다. 예를 들어 FOTRAN과 COBOL은 각각 주로 수학적 계산이나 경영 정보를 처리하기 위한 목적으로 만들어졌

다. 이와 달리 LISP은 여타의 프로그래밍 언어와 다른 목적을 위해 고안되었다. 즉 그것은 인간의 마음이 사용하고 있다고 가정되는 규칙과 개념을 컴퓨터에게 부여하기 위한 수단으로서 제안되었다.

그런데 이러한 인공언어를 구성하려는 시도는 역사적으로 볼 때 철학에서 비롯되었다. R.Descrtes와 G.Leibniz는 17세기에 이미 논리의 법칙을 확장하여 사고의 모든 영역에 적용될 수 있는 추론의 보편적 계산법과 그러한 계산법을 위한 보편적 기호언어를 추구했다. 그들의 생각은 기호적 추론 계산법으로 발전하지 못했으나 그 이후의 사람들에게 많은 영향을 주었다. 20세기 초에 등장한 분석철학은 철학의 주임무를 사고의 명료화라고 보고, 자연언어를 이용하여 철학함으로써 나타나는 의미상의 혼란을 시정하고자 했다. 분석철학자들은 의미의 다양성을 그 특징으로 하는 자연언어 대신 의미의 엄밀성을 보장하기 위해 인공언어에 관심을 갖게 되었다. L.Wittgenstein의 전기철학(1922)은 이러한 관심이 구체적으로 드러난 것이며 완전한 언어 모델로서 이상언어를 제시한다. 그의 이상언어에서 표현의 변항들의 값은 사실에 관한 문장이므로 B.Russell과 A.N.Whitehead(1913)가 제시한 바와 같은 완전한 논리적 구조를 지니며, 언어와 세계는 일대 일의 대응관계에 있다.

인공언어를 만들어 철학적 문제를 해결하려는 의도는 R.Carnap에서 보다 분명하게 나타난다. Carnap(1934:279)에 의하면 전통적인 철학의 문제들은 대부분 논리적 문제이다. 즉 그것들은 실질적 화법(material mode of speech)과 형식적 화법(formal mode of speech)을 혼동함으로써 발생하는 사이비 문제(pseudo-problem)이다. 그는 K.Gödel의 공리화 방법과 A.Tarski의 의미론등을 수용하여 형식체계를 구성하고, 구체적으로 그 체계를 이러한 문제들을 해소하는데 적용시키고 있다. W.S.McCulloch와 W.H.Pitts(1965)는 Carnap의 인공언어가 연결주의적 인공지능 연구에 활용될 수 있음을 보여준다. 본 논문은 철학과 인공지능간의 학제적인 연구와 생산적인 대화의 통로가 절실히 요청된다는 관점(이초식,1991)에서 논의를 하고 있다. 먼저 LISP의 구문론을 살펴보고, 그다음 McCarthy(1978:221)가 LISP의 기본적인 특성이라고 간주한 조건표현(conditional expression)과 회기(recusion)에 대해 논의하기로 한다.

2. LISP의 구문론

LISP은 구문론과 의미론이 수학적 함수 이론에서 유래되었다는 점에서 함수적 언어(functional language)이다. 수학적 함수에서는 표현들의 평가 순서가 명령형 언어에 일반적인 반복(iteration)이 아니라 조건 표현과 회기에 의해 결정된다. 이러한 점에서 LISP은 폰 노이만 방식의 구성에 기반을 둔 FOTRAN이나 C와 차이가 난다.

수학적 함수는 정의 집합(domain set)에서 대상 집합(range set)에로의 대응이다. 함수 정의(function definition)는 그러한 두 집합을 대응에 따라 규정한다. 일반적으로 정의는 함수 명칭(function name)으로 주어지는데, 그 형식은 (1)과 같이 함수명칭과 괄호안에 있는 파라미터들의 리스트, 그리고 대응이 표현된다.

(1) λ cube(x) ≡ x * x * x ; x는 실수이다.

A.Church는 명칭이 없는 함수를 표기하기 위한 방법으로서 λ 표기법(Lambda notation)을 개발했는데 그것은 자신의 λ 형식체계(Lambda calculus, LC)에 기반을 두고 있다. 결합 논리(combinatory logic)라고도 불리우는 LC는 1차 논리로서 특별한 기호가 선행하지 않고 단지 괄호에 의해 표시되며, 그결과 (1)은 (x)x * x * x 로 표현된다. (x,y)는 함수 x를 논항 y에 적용한 결과를 의미한다. LC가 지니는 중요한 특징은 x와 y, 그리고 그 값 z=(x,y)가 동일한 유형(type)에 속하므로 (x,x) = x 와 같은 표현이 가능하다는 점이다. 이러한 회기적 성질은 전산과학에서 중요한 의미를 지니며, 5절에서 자세히 논의될 것이다.

McCarthy는 기호처리 방식으로서의 리스트 처리가 계산가능성(computability)을 연구하는데 이용될 수 있다고 보았다. 계산가능성 문제에 있어 이미 모든 튜어링 머신의 작동을 모방할 수 있는 범용 튜어링 머신(universial Turing machine)을 구성할 수 있음이 증명되고 있었다. McCarthy는 범용 튜어링 머신에 대응되는 함수, 즉 LISP의 모든 다른 함수를 평가(evaluate)할 수 있는 범용 리습 함수(universial LISP function, EVAL)를 구성했다.

McCarthy는 범용 리습함수를 구성하는데 있어 데이터가 표현되는 방식과 동일하게 함수를 표현하기 위해 LISP의 데이터로서 괄호가 있는 리스트 기호법을 채용했다. 리스트 기호법에서 함수 호출(function call)은 (2)와 같은 폴란드식 표기법(Polish notation)으로 규정되었다.

(2) (function_ name argument_1 ... argument_n)

폴란드식 표기법은 논리학자 J.Lukasiewicz에 의해 개발된 것인데, 그것은 논리적 정식이 지닌 문법적 특성에 기반을 두고 있다. 즉 모든 논리적 정식은 하나의 관항어(functor)와 논항(argument)으로 분석되며, 이 분석단계에서 복합정식으로 남아있는 논항들은 다시 관항어와 논항으로 분석된다. p → (p v ~q) 와 같이 표준적으로 표현된 정식은 CpApNq로 표현된다. 폴란드식 표기법은 다른 방식에 비해 개개의 기호들의 구문론적 역할이 표현내의 위치에 따라 분명히 나타난다는 점과 괄호에 대한 규칙을 형성규칙(formulation rule)에 포함시킬 필요가 없다는 장점등을 갖는다. 이외에도 논리적 정리를 증명하는데 있어 장점(heuristic merit)도 지니고 있다고 보는 사람도 있다(J.J.Leman). 폴란드식 기호법을 채용함으로써 LISP은 몇가지 특징을 보여준다. ① 적용되는 함수를 분명하게 드러낸다. 즉 그것은 항상 리스트에서 첫번째 기호(symbol)이다. ② 많은 수의 논항을 표현할 수 있다. 예를 들어 (* 2 4 5 9 1)와 같은 표현이 가능하다.

함수 정의는 (3)에서 볼 수 있듯이 앞에서 언급된 λ 표기법을 이용하여 주어진다. 함수 명칭이 λ 표기법의 앞에 놓이고 그외의 것은 리스트안에 놓인다.

(3) (function_name LAMBDA(arg_1, ... arg_n) map_expression)

이상의 논의를 요약하면 LISP의 표현은 폴란드식 표기법과 λ 표기법을 이용하며,

그 결과 그것의 구문론은 매우 단순한 구조를 갖게 된다. LISP의 유일한 데이터 유형(data type)은 (2)와 같은 리스트 구조(list structure)이며, 일반적으로 하나의 데이터 형식은 아톰(atom)과 리스트이다. 하나의 형식체계로서 볼 때 LISP의 원초적 용어들(primitives)은 ① 괄호 (,) ② 아톰과 리스트 ③ FIRST, REST, APPEND, CONS등의 함수이다. LISP의 형성규칙(formation rule)에 따르면 아톰과 리스트만이 정형식(well formed formula)이다.

3. 조건표현

LISP은 술어(predicate)와 더불어 조건기호들(conditionals)를 사용하여 프로그램의 절차를 제어하므로 그것의 조건기호들은 이러한 절차적 특성이 반영되어 있다. 술어는 참과 거짓을 리턴하는 절차를 의미하며, 술어의 역할을 하는 함수는 참일 때 T, 거짓일 때는 NIL을 리턴한다. 여기서 중요한 점은 LISP에서는 NIL을 제외한 모든 것이 T로 간주된다는 점이다. 이러한 사실은 논리학의 경우와 근본적인 차이점을 보여준다. 논리학에서는 T와 F는 상호 배타적이므로 nonT ≡ NIL이고 nonNIL ≡ T이다. 그러나 LISP의 경우에는 T는 nonNIL 이지만 nonNIL은 항상 T가 되는 것은 아니다.

LISP의 조건기호는 IF, WHEN, UNLESS, COND 등이다. IF 표현은 (IF <test> <then form> <else form>) 이며, test form의 값이 T인 경우 then form을 평가하고 test form의 값이 NIL일 경우 else form을 평가한다.

IF 함수의 변형으로서 WHEN과 UNLESS가 있다. WHEN 표현은 (WHEN <test> <then form>) ≡ (IF <test> <then> nil) 이며 test form이 T인 경우 then form을 평가하고 그렇지 않은 경우 NIL을 리턴한다. UNLESS 표현은 (UNLESS <test> <else form>) ≡ (IF <test> nil <else form>)이고 test form이 NIL인 경우 then form을 평가하고 그렇지 않을 경우 NIL을 리턴한다.

LISP의 대표적인 조건표현인 COND 표현은 (4)와 같이 많은 검사절(test clauses)과 귀결절(test-and-consequent clauses)로 구성된다.

```
(4)     (COND (<test 1> <consequent 1-1> ...)
                ....
              (<tset n> <consequent m-1> ...)
```

COND는 각 절을 순차적으로 평가한다. 특정 절의 test form이 T로 평가되면 COND는 귀결절을 평가하고 그 값을 리턴하며, 더이상 다른 절들을 고려하지 않는다. 그렇지 않을 경우 COND는 해당 절의 귀결절을 평가하지 않고 다음 절을 검사하며, 모든 test form이 NIL이면 COND는 NIL을 리턴한다.

위에서 논의된 LISP의 조건표현들을 명제논리의 진리표(truth table)를 이용하여 나타내면 다음 표와 같다 (진리값은 위에서 언급된 차이는 있으나 논의의 편의상 LISP의 경우 NIL은 F로 표시하고 nonNIL은 T로 표시한다).

	p	q	p → q	(WHEN p q)	(UNLESS p q)	(IF p q)	(COND p q)
ⓐ	T	T	T	T	F	T	T
ⓑ	T	F	F	F	F	F	F
ⓒ	F	T	T	F	T	?	?
ⓓ	F	F	T	F	F	?	?

자연언어에는 "만약 ...이라면 _ _ _이다"(if-then)와 같은 많은 조건문이 있는데, 논리학에서은 그러한 용법 중 일부를 조건 기호 →를 이용하여 p → q로 기호화한다. p → q는 ~(p ∧ ~q)의 축약으로 정의된다. 이와같이 →에 의해 진리 함수적으로 기호화된 조건문을 실질함축(material implication, MI)이라고 하는데, 문제는 MI가 일상적인 직관과 맞지 않는다는데 있다. MI는 전건과 후건 간의 어떠한 논리적 관계도 요구하지 않으며, ~(p ∧ ~q) ≡ ~p ∨ q이므로 전건의 거짓만으로도 참이 된다. 이러한 문제를 MI의 문제라고 하자. 위의 진리표에 의하면 LISP의 조건기호들은 MI의 문제를 해소할 수 있는 것 처럼 보인다. (IF p q) 표현은 ⓐ와 ⓑ의 경우 q의 값에 따라서 진리함수적으로 전체 표현의 값이 결정되지만 ⓒ와 ⓓ의 경우는 p와 q의 값에 의해 결정될 수 없다. LISP의 조건 기호들은 기본적으로 test form이 T인 경우에 값을 평가하는 절차를 규정하며, 가능한 모든 검사의 경우를 다루고 있으므로 MI의 문제는 발생하지 않는다.

이제 MI의 문제를 다루는 논리학자들의 시도 중에서 현재의 논의와 관련되는 몇가지 이론을 살펴보기로 한다.

① 조건문에서 전건과 후건간의 관계에 대해 강한 함축관계를 도입함으로써 MI의 문제에 접근할 수 있다. C.I.Lewis는 귀결(entailment)을 연역가능성(deducibility)의 역으로 보고, 양상 개념을 이용하여 엄밀함축(strict implication)을 주장한다. 엄밀함축은 다음과 같이 정의된다: p ⥽ q ≡ □(p → q). A.R.Anderson과 N.D.Belnap은 귀결이 연역가능성의 역이라는 점을 인정하지만 표준적인 연역가능성은 적합성(relevance)을 고려하지 않으므로 결함이 있다고 주장한다. 대신 그들이 제안하는 적합적 함축(relevant implication)은 다음과 같이 정의된다 : p ⇒ q ≡ p ∧ (p → q).

② MI의 문제는 반사실적 조건문(counterfactuals,CF)에서 분명히 드러난다. 다음의 조건문을 살펴보자.

(5) 그 얼음 조각을 30 C로 가열하였더라면, 그것은 녹지 않았을 것이다.

(5)에서 전건은 현실상으로 실제로 발생하지 않았던 사태이므로 그것을 MI로 해석한다면 전건이 거짓이므로 참이 될 것이다. N.Goodman은 CF는 전건과 후건 사이의 인과적 혹은 논리적 연관성(connection)이 성립한다고 보고, 그러한 연관성이 있으면 CF는 참이된다고 주장한다. N.Goodman에서 비롯된 이러한 관점은 R.Stalnaker와 D.Lewis에 의해 체계화되고 있는데, 그들은 양상논리의 가능세계(possible world) 개념을 통해 CF를 분석하고 있다. 그들에 따르면 반사실적 조건문 p > q는 가능세계로부터 진리값에로의 함수이다. 따라서 p가 참이고 실제세계와 가장 유사한 가능세계에서 q 또한 참일 때 CF는 참이거나(Stalnaker), 또는 p와 q가 모두 참인 가능세계가 p가 참이고 q가

거짓인 가능세계보다도 실제세계에 더 가까울 때 CF는 참이다(Lewis). 이러한 접근들은 조건문이 지닌 다양한 특성을 드러낸다. LISP의 경우 p와 q 이외에 r을 도입함으로써 MI의 문제를 해소할 수 있다. 그러나 논리학의 경우 조건문의 형식이 아니라, 그 형식에 대한 다양한 해석체계를 제시하여 MI의 문제에 접근하고 있다.

4. 연언표현과 선언표현

이제 논리학의 결합사(connective)에 해당되는 AND와 OR를 살펴보자. LISP의 AND와 OR는 조건기호들이 그러했듯이 논리학의 ∧나 ∨와 다른 의미를 갖고 있다. LISP-AND의 평가 규칙은 다음과 같다. 즉 순차적으로 평가하되 특정 항이 NIL이면 평가를 멈추고 NIL을 리턴한다. 그렇지 않은 경우 다음 항을 평가한다. 만약 모든 항이 T이면 마지막 항의 값을 리턴한다. LISP-OR의 평가 규칙은 그와 대조적이다. 특정 항이 T이면 평가를 멈추고 그 값을 리턴한다. 그렇지 않을 경우 다음 항으로 진행하거나 왼편에 항이 없으면 NIL을 리턴한다.

LOGICAL-AND와 LISP-AND의 차이점은 다음과 같다. ① 위에서 알 수 있듯이 LISP-AND와 LISP-OR는 모든 항의 평가를 요구하지 않는다. 특정 항이 NIL이거나(LISP-AND의 경우) 특정 항이 T이면(LISP-OR의 경우) 다음 항이 평가되지 않는다. 이러한 특징이 없다면 에러를 피하기 위해 평가를 중단해야 할 경우가 있다. 다음의 예를 살펴 보자.

```
(6)  * (defun posnump (x)
         (lisp-and (numberp x) (plusp x)))
```

(6)에서 정의된 POSNUMP는 입력이 숫자이고 양수인 경우 T를 리턴한다. PLUS 술어는 어떤 숫자가 양수인가를 검사하기 위해 사용되는데 숫자가 아닌 다른 것이 입력되면 "WRONG TYPE INPUT" ERROR 라는 메시지를 리턴한다. 그러므로 POSNUMP에 대한 입력에 있어 PLUSP를 호출하기 전에 숫자임을 분명히 해야 할 필요가 있다.

부울 함수(Boolean function)는 입력과 출력이 T나 NIL을 의미하는 진리값이다. 부울함수는 논리값 true와 false를 사용하므로 논리적 함수라고도 한다. 다음과 같이 LOGICAL-AND를 정의해 보자.

```
(7)  * (defun logical-and (x y) (and x y t))
```

② LOGICAL-AND는 2항 (diadic) 결합사이므로 (and a b c d)와 같이 많은 변항을 입력할 수 있는 방법(nest or cascade)이 없다. ③ LOGICAL-AND는 2치 논리학(two-valued logic)의 결합사이다. 그러므로 개개의 논항은 T와 F 이외의 값을 가질 수 없다. LOGICAL-AND가 갖는 이러한 기본적 전제는 어느 경우에는 만족스럽지 않으므로 몇가지 방식으로 해결할 수 있다. ⓐ 다치논리(many-valued logic)를 수용하는 방법이 있다. T와 F 이외의 제3의 값을 인정하는 3치논리(Lukasiewicz)와 문자 그대로의 다치를 허용하는 퍼지논리(fuzzy logic)는 이러한 방식에 속한다. ⓑ 일단 F를 인정하

고 T를 not false로 이해하는 방식이 있다(반대의 경우도 마찬가지이다). LISP에서 NIL을 인정하고 T를 nonNIL로 간주하는 것은 여기에 속한다. 이러한 방식은 (8)에서 볼 수 있듯이 다양한 T를 리턴할 수 있게 한다.

(8)
```
* (logical-and 'tweet 'woof)
T
* (and 'tweet 'woof)
WOOF
* (and 'woof 'tweet)
TWEET
```

④ 위의 예는 LISP-AND와 LOGICAL-AND의 또 다른 차이점을 보여준다. LOGICAL-AND와 달리 LISP-AND는 교환법칙이 성립되지 않는다((8)에서 두번째와 세번째의 경우). OR의 경우에도 마찬가지이다.

Wittgenstein(1922)은 LOGICAL-AND의 의미가 진리 함수적인 $\wedge$에 의해 완전히 기호화되지 않는다고 지적했다. 진리함수적 LOGICAL-AND를 $\wedge_1$이라고 하고, 새로운 $\wedge_2$를 (9)와 같이 정의해보자.

(9) p $\wedge_1$ q and the number π is irrational.

위에서 정의된 $\wedge_2$는 $\wedge_1$과 동일한 진리값을 갖는다. 그러나 진리표는 p $\wedge$ q 가 참인가를 알기 위해 검사해야할 절차를 기술한다고 보는 것이 옳을 것이다. 즉 진리표는 절차적인 측면에서 해석되어야 하며 LISP-AND는 이러한 점을 반영하고 있다.

⑤ LOGICAL-AND는 매크로(macro)가 아니라는 점에서 LISP-AND와 구별된다. LOGICAL-AND는 논항이 평가되어야 하는가의 여부를 제어할 수 없다. 다음의 예를 보자.

(10)
```
* ( and (numberp 'fred) (oddp 'fred))
NIL
* (logical-and (numberp 'fred) (oddp 'fred))
ERROR! FRED wrong type input to ODDP.
```

이상에서 볼 수있듯이 LOGICAL-AND는 LISP-AND보다 더 제한된 결합사이다. 널리 알려진 바와 같이 C.Shannon은 부울 대수의 0과 1들은 전기 스위치 회로로 표현될 수 있음을 보였고, 그에 기반을 둔 컴퓨터가 등장했다. 그 경우 0과 1은 컴퓨터 회로가 구성되어지는 원초적인 논리적 연산자(primitive logical operator)이다. 마찬가지로 LOGICAL-AND와 LOGICAL-OR와 같은 부울함수는 LISP의 함수들이 구성되는 원초적 연산자라고 볼 수 있다.

5. 회기

회기(recursion)는 전산과학에서 가장 근본적인 개념 중의 하나이다. 회기 개념은 LISP에서 대체로 다음과 같이 세가지 의미로 사용되고 있다. ① 가장 일반적인 의미에서 회기는 LISP 함수들이 작업을 수행하기 위해 하위 차원의 함수들을 호출하는 것을 의미한다. LISP이 회기적 제어 구조를 가지고 있다는 것은 이러한 넓은 의미에서 볼 경우이다. ② 좀더 좁은 의미로는 LISP 함수가 그 자신을 직간접적으로 호출하는 경우를 말한다. 간단한 회기의 예를 보기로 하자.

```
(11)   * (+ 4 (+ 2 6))
       12
(12)   * (+ 4 (* 2 (+ 3 1))
       12
```

(11)에서 + 함수는 직접적으로 하위 차원의 + 함수를 호출한다. (12)에서 + 함수는 * 함수를 호출하고, 다시 * 함수는 + 함수를 호출한다. 따라서 + 함수는 간접적으로 회기적이다. ③ 가장 특수한 의미에서의 회기는 함수가 자신의 함수 정의 안에서 자신을 호출하는 경우를 말한다. 이러한 회기 개념은 조작을 수행하기 위한 강력한 메카니즘을 제공한다. 회기의 본질은 하나의 프로그램을 보다 작은 그 자신의 버전(version)으로 나누는 것이다. 그 결과 작은 버전에 의해 리턴된 값은 상위수준의 총체적인 해에 제공된다. FACTORIAL을 계산하는 다음의 예는 이러한 의미에서의 회기 개념이다.

```
(13)   (DEFUN FACTORIAL(N)
          (COND ((= N 0)    1    )
                (t (* N (FACTORIAL (- N 1))))))
```

회기는 일종의 정의 형식으로 보면 피정의항(definiendum)에 정의항(definiens)이 들어가는 경우에 해당한다. 예 (13)에서 FACTORIAL(N)을 구하기 위해 다시 FACTORAIL을 이용하고 있으므로 순환적 정의라고 볼 수 있다. 다음은 FACTORIAL(3)을 구하는 과정이다.

```
(14)   value of (FACTORIAL 3) = 3 x value of (FACTORIAL 2)
       value of (FACTORIAL 2) = 2 x value of (FACTORIAL 1)
       value of (FACTORIAL 1) = 1
```

회기를 이용한 순환적 정의는 잘못 사용될 경우 역설적인 상황을 낳을 수 있으므로 형식논리학은 회기적 논증을 피하려고 한다. 그러나 전산과학분야 이외에도 이러한 회기 개념이 많이 이용되고 있다. 널리 알려진 「천일야화」(Arabian Nights)는 "이야기안에 이야기안에 이야기"를 포함하고 있으며 그결과 회기적인 소설의 묘미를 제공한다.

D.Hofstadter는 J.S.Bach의 음악과 M.C.Escher의 그림(Drawing Hands, Ascending and Descending), Gödel의 불완전성정리등에서 회기 개념이 중요한 역할을 하고 있음을 관찰하고, 이어서 회기는 의식의 핵심적 현상임을 주장한다. 그에 의하면(1989:648) 부울이 의도한 바와 같은 의미의 사고의 법칙(law of thought)은 없다.

철학에서 회기가 사용되어 역설적인 상황에 이르게 된 대표적인 예로 거짓말장이 역설(liár paradox,LP)이 있다. 그 역설의 구조를 단순화시키면 (15)와 같이 구성된다. X라는 사람이 다음과 같이 말했다고 하자.

(15) 내가 지금 말하고 있는 것은 거짓말이다.

이제 두가지 경우가 있다. 즉 X가 거짓말을 했을 경우 (15)는 거짓말이고 따라서 X는 거짓말을 하지 않았다(다른 경우도 마찬가지이다). Russell은 LP와 유사한 역설을 발견했는데, LP가 참이라는 의미론적 요소와 관련이 있는데 비해, Russell의 역설은 집합 개념과 관련된다. 자신의 원소가 아닌 모든 집합들의 집합 $R = \{ x | x \notin R \}$ 을 고려해보자. 이제 어떤 집합이 R에 속하기 위해서는 그 자신의 원소가 아니어야 한다. R은 그 자신의 원소인가? 만약 R이 그 자신의 원소라면 R에 속하기 위한 필요조건을 만족시킬 것이고 따라서 R은 그 자신의 원소가 아니다. 그러므로 R이 자신의 원소라면 그것은 자신의 원소가 아니다 : $R \in R \equiv R \notin R$ (반대의 경우도 마찬가지이다).

이러한 역설들은 공통적으로 자기 지시성(self reference)을 갖고 있다. Russell은 이러한 역설들이 악순환의 원리(vicious circle principle)을 위반함으로써 발생한다고 진단한다. 악순환의 원리란 어떤 전체도 그 전체의 구성원으로서 그 전체에 의해서만 정의가 가능한 그런 구성원은 포함될 수 없다는 것이다. 다시 말하면 역설에 빠지지 않으려면 자기지시적이지 말아야 한다는 것이다.

철학자들이 이러한 역설을 해결하려는 시도는 대략 두가지로 나타났는데, 그 시도들은 LISP과 관련되어 분석될 수 있다. ① Russell의 유형론(type theory)은 다수의 유형을 가정함으로써 역설을 해결하고자 한다. 집합론에서 언급하는 모든 실체들은 하나의 특정 유형 T에만 속한다. 우선 모든 대상은 T_0에 속한다. T_1은 T_0의 대상들의 집합이며 T_m에 속하는 원소는 T_{m+1}의 유형의 원소가 아니다. 그러므로 $x \in y$는 x가 y에 속한 유형 보다 하나 더 하위차원의 유형이다. 따라서 (15)는 T_1에 속하므로 역설이 발생하지 않으며, $x \in x$와 같은 표현은 무의미하게 된다. 이점은 앞에서 보았던 성질과 근본적으로 대조된다. 회기 함수이론에서는 (x,x) = x와 같은 표현이 의미가 있으며, 괄호속의 x와 그 값(표현)으로서의 x가 동일한 유형에 속한다.

② A.Tarski는 대상언어(object language) 와 메타언어(metalanguage)를 구별함으로써 역설을 해결하고자 했다. Tarski(1949)는 참(true)이란 용어는 반드시 어떤 특정 언어에 대한 상대적인 의미로 파악해야 한다고 주장했다. 그에 따르면 LP가 발생한 이유는 ① 언어는 의미론적으로 닫혀 있다(semantically closed). 즉 자기 지시적이다. ② 일반적인 고전 논리규칙이 성립한다는 두 전제에서 비롯된다. Tarski는 ②를 거부할 수는 없으므로 ①을 거부함으로써 역설을 방지하고자 한다. 따라서 참을 정의하는데 두가지 언어를 사용해야 한다. 즉 언어 L_1에서 참이라는 것을 표현하기 위해서는 그와는 다른 언어 L_2, 즉 메타언어를 이용해야 한다. L_2는 참을 (16)과 같은 형식으로

정의한다.

(16) X is true if and only if p.

(16)에서 p는 L_1 중의 임의의 문장이다. 그러므로 L_1에서 발생하는 모든 문장은 반드시 L_2로 나타나야 한다. Tarski의 주장을 수용한다면 (15)는 L_1이 아니라 L_2이므로 LP는 발생하지 않는다.

LISP을 이용하여 LISP을 정의할 수 있다는 점에서 LISP은 메타언어로 사용된다. 이러한 점은 Carnap이 자신의 언어 I 의 구문론을 언어I에서 구성하고 있는 것과 동일하다. LISP으로 정의된 LISP을 MICRO LISP이라고 하면 MICRO-FIRST, MICRO-EVAL, MICRO-APPLY등이 정의된다. 이러한 MICRO LISP이 수정되고 향상되어 하나의 언어가 되고 그 결과 LISP 그 자체와 같아질 수 있다. 이러한 방식으로 LISP은 새로운 언어(LISP-like LISP)를 위한 해석기(interpreter)를 구성한다. MICRO-PLANNER (Sussman and McDermott,1972)나 CONNIVER (Sussman, Winograd, and Charniak,1971)와 같은 언어는 이러한 방식으로 구성되었다.

6. 맺는말

본 논문은 LISP의 조건 표현과 회기에 관하여 몇가지 철학적인 분석을 했다. 이러한 분석은 새로운 이론을 제시한 것은 아니며, 단지 LISP이 갖고 있는 그러한 기본적인 성질들은 철학적인 문제와 관련되어 있음이 지적되었다. 철학자들은 우리의 사고를 분명히 표현할 수 있는 도구로서 많은 형식체계를 제안하고 있으며, 그러한 해석되지 않은 체계에 대한 의미론을 부여하는 작업을 수행하고 있다. 인공지능 연구가 효과적으로 그 목적을 달성하는데에는 이러한 철학자들의 작업이 도움이 될 수 있다고 생각된다. 본 논문이 의도한 바는 바로 그러한 교류의 가능성이다.

참고문헌

이초식(1991). 인공지능과 카르납의 인공언어. 1991년도 한국인지과학회 춘계 발표대회 논문집 : 33-43.

Carnap,R.(1937). The Logical Syntax of Language. RKP.

Hofstadter,D.R.(1979). Gödel,Escher,Bach: An Eternal Golden Braid. Basic Books.

McCarthy,J.(1978). History of LISP. in D.Wexelblat(ed), History of Programming Languages. Academic Press.

Tarski,A.(1949). The Semantic Conception of Truth. in Readings in the Philosophical Analysis. H.Feigl and W.Sellas(ed). Appleton-Century-Crofts : 52-84.

Wittgenstein,L.(1922). Tractatus Logico-Philosophicus. RKP.