Journal of Knowledge Information Technology and Systems(JKITS), Vol. 15, No. 2, pp. 185-193, April 2020
https://doi.org/10.34163/jkits.2020.15.2.004

Journal of Knowledge Information Technology and Systems

ISSN 1975-7700

http://www.kkits.or.kr

A Study on the Relationship Between Hoare Logic and Curry-Howard Correspondence

Gyesik Lee[*1], **Hwajeong Kim**[2]

[1]*School of Computer Engineering & Applied Mathematics, Hankyong National University*

[2]*Department of Mathematics, Hannam University*

ABSTRACT

The relation between computer programs and mathematical logic is well known, Although one can easily find comments on the relationship when one reads books and articles about computer programming, it is difficult to find proper materials explaining the relationship in a scientific manner. Nevertheless, people used to mention how important the relationship between computer programs and mathematical logic is. The problem is that it is more focused on the fact that how helpful it is to learn one area in learning the other area. However, there is more than that between computer programs and mathematical logic. In this paper, we introduce some researches about the relationship and show different and common aspects of them. In particular, Hoare logic and Curry-Howard correspondence is compared. Hoare logic was invented for rigorous verification of programs in a C-like imperative programming language, while Curry-Howard correspondence explains the relationship between functional programming language like Haskell and mathematical logic. We show that Hoare logic and Curry-Howard correspondence are essentially the same in the sense that they represent the same things just in different manners. We close the paper with some remarks about Hoare logic and the Curry-Howard correspondence and ideas of further research with respect to teaching and learning programming languages in the university level based on the relationship between logic and programming.

KEYWORDS: Hoare logic, Curry-Howard correspondence, Correctness proof, Mathematical logic, Lambda calculus

ARTICLE INFO: Received 22 March 2020, Revised 7 April 2020, Accepted 10 April 2020.

[*]Corresponding author is with the School of Computer Engineering & Applied Mathematics, Hankyong National University, 327 Jungang-ro Anseong-si Gyeonggi-do, 17579, KOREA.

E-mail address: gslee@hknu.ac.kr

1. 서 론

컴퓨터 프로그램과 논리와의 관계는 널리 알려져 있으며, 두 분야가 밀접하게 관련된다는 사실이 자주 언급된다. 반면에 컴퓨터 프로그램과 논리가 구체적으로 어떻게 연관되어 있는가를 배울 수 있는 기회는 많지 않다. 실제로 컴퓨터 공학 분야의 학과에서 전산 논리를 전공 교과목으로 수강할 기회가 매우 적다. 국내 대학에서뿐만 아니라 국외 대학의 경우도 크게 다르지 않다. 컴퓨터 공학 분야 내에서 전산논리 관련 학회와 관련된 전문가들은 소수에 불과하다는 사실이 이를 반증한다.

그럼에도 불구하고 컴퓨터 프로그램과 논리 사이의 연관성의 중요성은 매우 강조된다. 다만 그 강조가 한쪽이 다른 한쪽에 도움된다는 정도로 머무는 것이 일반적이다. 예를 들어, 컴퓨터 프로그래밍을 이용하여 논리적 사고를 높인다거나, 논리적 사고를 잘하면 컴퓨터 프로그래밍에 도움된다 등과 같은 주장들이다. 그런데 컴퓨터 프로그램과 논리의 관계는 그 이상이다.

본 논문에서는 컴퓨터 프로그램과 논리의 연관성에 대한 기존의 연구들 사이의 차이점과 공통점을 밝힌다. 보다 구체적으로는 명령형 프로그래밍 언어로 작성된 프로그램의 건전성(soundness)을 논리적으로 검증하기 위해 사용되는 호어 논리(Hoare logic)와 함수형 프로그래밍 언어의 기반을 제공한 커리-하워드 대응관계(Curry-Howard correspondence)가 본질적으로 동일한 내용을 다른 방식으로 표현한다는 사실을 밝힌다.

논문의 구성은 다음과 같다. 2장은 일상에서의 프로그램과 컴퓨터 프로그램의 차이점과 공통점을 살펴본다. 3장은 컴퓨터 프로그래밍의 핵심 요소를 살펴본다. 4장은 명령형 프로그램의 건전성을 호어 논리로 검증하는 방법을 간략하게 소개한다. 5장은 커리-하워드 대응관계를 다루며, 6장에서 호어 논리와 커리-하워드 대응관계가 사실상 동일함을 밝힌다. 7장은 본문에서 다룬 내용을 정리 요약한 후 남겨진 과제들을 설명한다.

2. 프로그램 vs. 컴퓨터 프로그램

일반적으로 언급되는 '프로그램'이란 단어는 아래와 같이 이해된다.

- 시간관리: 계획적인 시간 배분을 통한 관리 또는 프로젝트 진행계획 및 관리 등.
- 일정(스케줄): 예정된 일, 사건 등을 발생 시간 순서대로 나열한 시간목록, 버스/기차/비행기 시간표, 음악 콘서트/ 스포츠 프로그램 등.
- 계획(planning): 특정 목표를 달성하기 위해 요구되는 활동에 대한 사고 처리과정 또는 사고 과정과 관련된 뇌의 연속처리기능 등.

프로그램을 사용하는 목적은 보통 (1) 특정 목표를 달성학기 위한 처리과정 명시, (2) 출발점에서 도착지까지의 여정 묘사, (3) 현재 준비된 것을 이용하여 원하는 결과를 이뤄내는 과정, (4) 전제로부터 결론을 유도하는 과정 등으로 이해된다.

반면에 컴퓨터 프로그램은 컴퓨터가 수행해야 할 작업들을 모아 놓은 명령문들의 모음이다. 즉, 특정 문제를 해결하기 위한 일련의 명령문 집합체이다. 또한 특정 프로그래밍 언어를 이용하여 컴퓨터 프로그래머가 작성한다.

일반 프로그램과 컴퓨터 프로그램 사이의 주요 차이점은 사용되는 언어이다. 일반 프로그램은 한국어, 영어, 중국어, 스페인어 등 자연어를 사용하며, 컴퓨터 프로그램은 C, 자바, 파이썬, 자바스크립트, 하스켈 등 컴퓨터 프로그래밍 언어로 작성된다. 이외에 다른 근본적인 차이점은 없다.

실제로 컴퓨터 프로그래밍은 특정 목적을 달성

하기 위해 컴퓨터가 실행할 수 있는 프로그램을 디자인하는 과정이며, 프로그램 디자인은 목적 및 과제 분석, 알고리즘 작성, 알고리즘 정확도 및 시간/공간 복잡도 분석, 알고리즘 구현 등을 포함한다. 또한 컴퓨터 프로그램은 특정 프로그래밍 언어로 작성된 소스코드로 제시되는데 이는 시간 관리, 일정, 계획 등을 보여주는 프로그램 안내책자와 동일한 기능을 갖는다.

3. 프로그래밍 언어

컴퓨터 프로그래밍 언어는 컴퓨터 프로그램 작성을 위해 사용되는 언어이며, 컴퓨터가 수행해야 할 명령문들을 작성할 수 있도록 도와주는 기호와 문법으로 구성된다. 또한 자연어와 다음 두 가지 측면에서 구별된다.

첫째, 구문(syntax)에 엄격하다. 즉, 컴퓨터가 수행해야 할 명령문을 작성할 때 작성문법을 철저하게 지켜야 한다. 둘째, 의미(semantics)에 엄격하다. 상황과 문맥에 따라 다른 의미를 가질 수 있는 자연어 문장과는 달리 명령문 수행방식 또한 지정되어서 명령문의 의미는 변하지 않는다.

프로그래밍 언어는 일반 언어와 다른 구조와 형식을 갖는다. 처음 프로그래밍 언어를 접하는 사람들의 경우 한국인이 러시아어, 아랍어 등을 처음 접할 때의 느낌을 받을 것이다. 하지만 문법과 어휘를 학습하면 자연어보다 쉽게 프로그래밍 언어에 익숙해진다.

가장 오래된 프로그래밍 언어이면서도 현재 가장 많이 사용되는 언어 중 하나인 C 프로그래밍 언어와 유사한 미니 명령형 언어를 이용하여 프로그래밍의 구문과 의미를 소개한다. 예를 들어, 아래 코드를 살펴보자. <표 1>의 프로그램은 콜라츠 추측(Collatz conjecure)을 구현한 간단한 프로그램이다. 임의의 양의 정수에 대해 짝수이면 2로 나누고, 홀수이면 세 배 더하기 1을 하는 과정을 결과가 0이 될 때까지 반복적용하며, 최종적으로 0에 다다를 때까지 적용된 횟수(count)를 내준다.

표 1. 콜라츠 추측 프로그램
Table 1. Collatz conjecture program

```
input n:
  count = 0
  while (n>0){
    if n%2 == 0 then
      n = n/2
    else
      n = 3*n+1;
    count = count+1
  }
return count
```

콜라츠 추측은 위 프로그램이 임의의 양의 정수 n에 대해 항상 종료한다는 주장이다.* 위 프로그램에 사용된 프로그래밍 요소는 변수 선언, 조건 제어문, while 반복문 등 몇 되지 않는다. 하지만 언급된 요소들만으로 충분히 강력한 프로그래밍 언어**를 구현할 수 있다. <표 1>의 프로그램 작성을 가능하게 하는 미니 명령형 언어의 구문은 다음과 같다.

표 2. 미니 명령형 언어 구문
Table 2. Mini imperative language syntax

```
Prog ::= input x1, …, xn:
           C
         return E

C ::=  skip
     | V = E
     | if E then S else S
     | while E { C }
     | C ; C

E ::= N | V | ( E O E )
O ::= + | - | < | =
```

* https://en.wikipedia.org/wiki/Collatz_conjecture
** 튜링-완전한 프로그래밍 언어를 의미한다.

<표 2>의 미니 명령형 언어에 사용된 구문은 다음과 같다.

Prog : 프로그램 집합
C : 명령문 집합
V : 변수 집합
E : 표현식 집합
N : 정수 집합
O : 연산자 집합

위 구문에 포함된 요소들의 의미(semantics)하는 바는 일반적인 의미와 동일하다. 예를 들어, 표현식은 정수와 변수, 그리고 연산자들을 조합하여 생성한다. 명령문은 변수 선언, 조건 제어문, while 반복문 등의 조합으로 생성되며, 프로그램은 입력값을 받아 명령문을 실행하여 생성된 값을 내주도록 정의되었다. 미니 명령형 프로그래밍 언어를 이용하여 콜라츠 추측 프로그램을 구현할 수 있음을 쉽게 확인할 수 있다.*

4. 프로그래밍과 논리의 관계

컴퓨터 프로그램의 건전성 증명, 즉, 구현된 프로그램이 원하는 성질을 갖는다는 것을 확인해주는 수학적 증명의 중요성이 점점 커져 왔다. 관련된 연구가 1960년대부터 이루어져 왔으며, 컴퓨터 프로그램의 건전성을 확인하는 다양한 기법과 도구들이 개발되었다.

알려진 많은 기법과 도구들이 수학적 증명 보다는 테스팅에 중점을 둔다. 하지만 테스팅과 증명은 엄연히 다르다. 테스팅은 구현된 프로그램이 의도한 대로 작동하는가를 관찰한다. 하지만 테스팅 결과가 완전하다는 보장을 절대 할 수 없다. 반면에 증명은 구현된 프로그램이 의도된 성질을 갖는다는 것을 수학적으로 완전하게 보장한다. 테스팅에 비해 상대적으로 어렵고 많은 비용을 요구하지만 산업적 중요성이 점차 증가하고 있다.

4.1 호어 논리

프로그램 검증은 1969년에 호어(C.A.R. Hoare)가 발표한 호어 논리[5,7,8]의 발표와 함께 본격적으로 시작하였다. 호어 논리는 명령형 프로그래밍 언어로 작성된 컴퓨터 프로그램의 건전성을 증명하기 위한 형식 체계(formal system)이다.

호어 논리에 포함된 호어 규칙(Hoare rules)은 호어 트리플(Hoare triples)을 기본 논리식으로 사용하며, 연속되는 명령문을 호어 트리플 사이의 추론 관계로 묘사한다. 호어 트리플의 기본 형식은 다음과 같다.

$$\{P\}\ C\ \{Q\}$$

C는 명령문을 가리킨다. P와 Q는 1계 논리식(first-order formula)이며 메모리의 상태를 묘사한다. 위 논리식이 의미하는 바는 다음과 같다. P가 참인 상태에서 명령문 C 실행한 후, C의 실행이 멈추면 Q가 참인 상태가 된다.

호어 논리를 구성하는 호어 규칙들은 <표 3>에 모아놓았다.

<표 3>에 포함된 규칙들 중에서 마지막 규칙은 while 반복문에 대한 호어 트리플 규칙이다. 이를 위해 반복 과정에서 항상 참이어야 하는 불변식 I를 사용하였으며, B는 조건식을 나타낸다.

적절한 불변식을 찾는 일은 일반적으로 매우 어렵다. 앞서 <표 1>에서 구현된 콜라츠 함수가 임의의 정수에 대해 실행을 멈춘다에 대한 증명이 아

* 콜라츠 추측 프로그램에 사용된 *, /, %, >, == 등의 기호는 다른 연산자를 이용하여 정의될 수 있다.

직도 알려지지 않았다. 또한 불변식을 찾는 일은 괴델의 불완전성 정리와 밀접하게 연관된다.

표 3. 호어 규칙
Table 3. Hoare rules

$$\frac{}{\{Q[E/x]\}\ x := E\ \{Q\}}$$

$$\frac{\{P\}\ S_1\ \{R\} \quad \{R\}\ S_2\ \{Q\}}{\{P\}\ S_1; S_2\ \{Q\}}$$

$$\frac{P \Rightarrow P' \quad \{P'\}\ S\ \{Q'\} \quad Q' \Rightarrow Q}{\{P\}\ S\ \{Q\}}$$

$$\frac{\{P \wedge B\}\ S_1\ \{Q\} \quad \{P \wedge \neg B\}\ S_2\ \{Q\}}{\{P\}\ \texttt{if}\ B\ \texttt{then}\ S_1\ \texttt{else}\ S_2\ \{Q\}}$$

$$\frac{\{I \wedge B\}\ S\ \{I\}}{\{I\}\ \texttt{while}\ B\ S\ \{I \wedge \neg B\}}$$

아래 표는 호어 트리플 예제를 소개한다.

표 4. 호어 트리플 예제
Table 4. Hoare triple examples

선행조건	명령문	후행조건
{true}	x = 5	{x=5}
{x=y}	x = x + 3	{x=y+3}
{x>-1}	x = x*2 + 3	{x>1}
{x=a}	if (x < 0) then x = -x else x = x	{x=|a|}
{false}	x = 3	{x=8}
{x<0}	while (x != 0){ x = x - 1}	{?}

명령문에서 사용되는 등호 기호는 변수 할당을 가리키고, 논리식에서 사용되는 등호 기호는 양변의 값이 동일함을 의미한다. 예를 들어, 아래 호어 트리플

{true} x = 5 {x=5}

의 의미는 다음과 같다.

- 'true'는 메모리가 초기상태, 즉, 아무런 정보도 포함하지 않는 상태임을 의미한다. 이 상태에서 변수 할당 명령문 'x = 5'를 실행한다.
- 변수 할당문을 실행한 결과 메모리 상에 변수 x가 정수 5를 가리키는 내용이 추가되었다. 따라서 논리식 'x=5'이 참이된다.

4.2 프로그램 검증 예제

호어 논리를 이용하여 프로그램을 검증하는 방법을 살펴본다. 호어 논리의 실용성을 확인하기 위해 미니 명령형 언어 대신에 C 언어로 구현된 프로그램을 살펴본다.

표 5. 틀린 배열 항목들의 합
Table 5. Sum of array elements with errors

```
float sum(float *arr) {
  int N = len(arr);
  float total = 0;
  int j = 0;
  while (j < N) {
    j = j + 1;
    total = total + arr[j]
  }
  return total
}
```

<표 5>에서 정의된 sum 함수는 실수들의 배열 가리키는 포인터를 인자로 입력받아, 배열에 포함된 실수들의 합을 계산하여 내주도록 함수로 구현되었다. sum 함수가 의도한 성질을 갖는다는 사실을 아래 호어 트리플로 표현할 수 있다.

$$\{\mathrm{N} = \mathrm{len}(\mathrm{arr}) \geq 0\}\ \mathrm{C}\ \{\mathrm{total} = \sum_{k=0}^{N-1} \mathrm{arr}[k]\}$$

단, C는 sum 함수의 본체를 나타낸다.

위 식이 성립하려면 while 반복문에 대한 규칙 또한 적용가능해야 하며, 그러기 위해서는 아래 호어 트리플이 성립해야 한다.

$$\{P\}\ D\ \{Q\}$$

여기서 D는 while 반복문의 본체를 가리키며, 논리식 P와 Q는 다음과 같다.

$$P = (0 \le j \le N) \wedge (total = \sum_{k=0}^{j-1} arr[k]) \wedge (j < N)$$

$$Q = (0 \le j \le N+1) \wedge (total+arr[j+1] = \sum_{k=0}^{j} arr[k])$$

그런데 위 호어 트리플이 성립하려면 P가 성립한다고 가정했을 때 Q가 성립해야 하는데 그렇지 않기 때문이다. 위 호어 트리플이 성립하지 않는 실제 이유는 total = total + arr[j] 가 실행 되기 전에 j = j + 1을 먼저 실행되어 결국 arr[0]이 total에 추가되지 않기 때문이다. 아래 표는 수정된 sum 함수 구현 프로그램이다.

표 6. 수정된 배열 항목들의 합

Table 6. Corrected sum of array elements

```
float sum(float *arr) {
  int N = len(arr);
  float total = 0;
  int j = 0;
  while (j < N) {
    total = total + arr[j]
    j = j + 1;
  }
  return total
}
```

호어 논리를 활용한 다른 예제는 보안 프로토콜의 건전성 검증에 많이 사용된다. 관련된 예를 [lee]에서 찾아볼 수 있다.

5. 커리-하워드 대응관계

커리-하워드 대응관계는 논리적 증명과 프로그램 사이의 대응관계를 묘사하며, 1934년 커리(H. Curry)[2,1]에 의해 처음으로 알려졌다.

<표 7>은 명제논리와 람다대수(Lambda calculus) 사이의 대응관계를 묘사한다. 그리고 이 대응관계는 프로그래밍과 논리를 하나로 다룰 수 있는 시스템으로까지 발전하였다.

표 7. 명제논리와 람다대수

Table 7. Propositional logic and Lmabda caculus

$$\frac{}{\Gamma, A \vdash A} \qquad \frac{}{\Gamma, u : A \vdash u : A}$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \supset B} \qquad \frac{\Gamma, u : A \vdash \pi : B}{\Gamma \vdash \lambda u.\pi : A \subset B}$$

$$\frac{\Gamma \vdash A \supset B \quad \Gamma \vdash A}{\Gamma \vdash B} \qquad \frac{\Gamma \vdash \pi_1 : A \supset B \quad \Gamma \vdash \pi_2 : A}{\Gamma \vdash \pi_1\, \pi_2 : B}$$

컴퓨터와 수학을 하나의 시스템으로 다루는 시도는 드 브로인(de Bruijn)이 1967년에 발표한 Automath[3]이다. 이후 1980년을 전후에 모든 수학적 증명을 컴퓨터로 다룰 수 있는 시스템의 기초를 제공하는 마틴-뢰프(Martin-Löf) 유형론[11,12]이 발표되었다. 마틴-뢰프 유형론은 80년대 초반부터 본격적으로 발전하기 시작한 증명보조기, 정리증명기 등의 기초 이론을 제공하였다[4,6]. 대표적으로 Coq*, Agda**, Lean*** 등이 대표적이다.

6. 호어 논리 vs. 커리-하워드 대응관계

호어 논리는 C언어 계의 명령형 프로그래밍 언어를 대상으로 작동한다. 반면에 커리-하워드 대응관계는 하스켈*과 같은 함수형 프로그래밍 언어와 밀접히 연관된다. 하지만 호어 논리와 커리-하워드 대응관계는 사실 동일하다.

실제로 하스켈 프로그래밍 언어[10,13]는 명령형 프로그래밍 언어의 핵심인 메모리 상태를 제1종 객체(first-class object)로 지원한다. 이를 통해 명령형 프로그래밍과 함수형 프로그래밍의 가장 근본적인 차이점인 부수 효과(side effect)를 동반하는 프로그램을 하스켈에서 구현할 수 있다.

반면에 호어 트리플 {P} C {Q}에서 C를 프로세서, 즉, 일종의 함수로 볼 수 있으며, P와 Q는 각각 P와 Q라는 성질을 만족하는 메모리 상태로 간주할 수 있다. 즉, C를 메모리 상태들의 집합에서 메모리 상태들의 집합으로 가는 함수로 본다. 보다 구체적인 설명은 다음과 같다.

- State: 모든 메모리 상태들의 집합
- C: $\{\sigma \in \text{State} \mid P(\sigma)\} \rightarrow \{\sigma \in \text{State} \mid Q(\sigma)\}$

이런 식으로 하면, C를 커리-하워드 대응관계에 의해 "P이면 Q이다"에 대한 증명으로 간주할 수 있게 된다.

7. 결 론

프로그래밍과 논리는 서로 다른 모습을 갖지만 실제로는 동일한 뿌리에서 출발했다. 본 논문에서 컴퓨터 프로그램과 논리의 연관성에 대한 기존의 연구들 사이의 차이점과 공통점을 살펴 보았다.

보다 구체적으로는 명령형 프로그래밍 언어로 작성된 프로그램의 건전성을 논리적으로 검증하기 위해 사용되는 호어 논리(Hoare logic)와 함수형 프로그래밍 언어의 기반을 제공한 커리-하워드 대응관계가 본질적으로 동일한 내용을 다른 방식으로 표현한다는 사실을 보였다.

앞으로 남은 과제는 본 논문에서 확인된 결과를 컴퓨터 프로그래밍 교육법과 접목하는 일이다. 특히 계산적 사고의 중요성이 점차 커지면서 컴퓨터 프로그래밍 교육의 중요성 또한 증대하고 있다[14-17]. 바로 이 점에서 컴퓨터 프로그램과 논리 사이의 관계가 중요한 역할을 수행할 수 밖에 없다. 하지만 이 부분에 대한 연구가 주로 경험에 기반하며, 컴퓨터 프로그래밍과 논리의 관계에 기반한 교육법에 대한 연구는 별로 알려지지 않았다. 이후 이 부분에 대한 보다 깊은 연구를 수행하고자 한다.

* https://coq.inria.fr
** https://wiki.portal.chalmers.se/agda/
*** https://leanprover.github.io/
* https://www.haskell.org/

References

[1] G. Boolos, J. Burgees, and R. Jeffrey, *Computability and logic*, 5th Edition, Cambridge University Press, 2007.

[2] H. Curry, *Functionality in combinatory logic*, Proceedings of the National Academy of Science of the USA, Vol. 20, No. 11, pp. 584-50, 1934.

[3] N. G. de Bruijn, *Verification of mathematical proofs by a computer, A preparatory study for a project AUTOMATH*, Unpublished, 1967.

[4] JR. J. G. B. de Queiroz, A. G. de Oliveira, and D. M. Gabbay, *The functional interpretation of logical deduction*, Advances in Logic 5, Imperial College Press/World

Scientific, 2011.

[5] R. W. Floyd, *Assigning meanings to programs*, Proceedings of the American Mathematical Society Symposia on Applied mathemaics, Vol. 19, pp. 19-31, 1967.

[6] J-Y. Girard, *Proofs and Types*, Translated and with appendices by Y. Lafont and P. Taylor, Cambridge University Press, 2007.

[7] C. A. R. Hoare, *An axiomatic basis for computer programming*, Communications of the ACM, Vol. 12, No. 10, pp. 576-580, 1969.

[8] M. Huth, M. Ryan, Logic in Computer Science: Modelling and reasoning about systems, 2nd Edition, Cambridge University Press, 2004.

[9] G. Hutton, Programming in Haskell, 2nd Edition, Cambridge University Press, 2016.

[10] G. Lee, *Formal specification of cryptographic security protocols*, Journal of Knowledge Information Technology and Systems, Vol. 14, No. 6, pp. 711-718, 2019.

[11] P. Martin-Löf, *Intuitionistic type theory: Notes by Giovanni Sambin of a series of lectures given in Padua, June 1980*, Bibliopolis, 1984.

[12] B. Nordström, K. Petersson, and J. M. Smith, *Programming in Martin-Löf's type theory,* Oxford University Press, 1901.

[13] S. Thompson, *Type theory and functional programming*, Addison-Wesley, 1991.

[14] J. M. Wing, *Computational thinking,* Communications of the ACM, Vol. 49, No. 3, pp. 33-35, 2006.

[15] J. M. Wing, *Computational thinking and thinking about computing,* Philosophical Transaction of the Royal Society A: Mathematical, Physical and Engineering Sciences, Vol. 366, pp. 3717-3725, 2008.

[16] J. M. Wing, *Computational thinking's influence on research and education for all,* Italian Journal of Educational Technology, Vol. 25, No. 2, pp. 7-14, 2017.

[17] G. Yang, *The effect of software education using pair programming on learning motivation and academic achievement*, Journal of Knowledge Information Technology and Systems, Vol. 15, No. 1, pp. 57-65, 2020

호어 논리와 커리-하워드 대응관계 사이의 연관성 분석

이계식[1], 김화정[2]

[1]*한경대학교 컴퓨터응용수학부 교수*
[2]*한남대학교 수학과 교수*

요 약

컴퓨터 프로그램과 논리와의 관계는 널리 알려져 있으며, 두 분야가 밀접하게 관련된다는 사실이 자주 언급된다. 반면에 컴퓨터 프로그램과 논리가 구체적으로 어떻게 연관되어 있는가를 배울 수 있는 기회는 많지 않다. 그럼에도 불구하고 컴퓨터 프로그램과 논리 사이의 연관성의 중요성은 매우 강조된다. 다만 그 강조가 한쪽이 다른 한쪽에 도움된다는 정도로 머무는 것이 일반적이다. 예를 들어, 컴퓨터 프로그래밍을 이용하여 논리적 사고를 높인다거나, 논리적 사고를 잘하면 컴퓨터 프로그래밍에 도움된다 등과 같은 주장들이다. 그런데 컴퓨터 프로그램과 논리의 관계는 그 이상이다. 본 논문에서는 컴퓨터 프로그램과 논리의 연관성에 대한 연구를 소개한 후 기존의 연구들 사이의 차이점과 공통점을 밝힌다. 보다 구체적으로는 명령형 프로그래밍 언어로 작성된 프로그램의 건전성을 논리적으로 검증하기 위해 사용되는 호어 논리(Hoare logic)와 함수형 프로그래밍 언어의 기반을 제공한 커리-하워드 대응관계가 본질적으로 동일한 내용을 다른 방식으로 표현한다는 사실을 밝힌다. 논문의 결론에서 호어 논리와 커리-하워드 대응관계에 대한 몇 가지 성질을 언급한다. 또한 이어지는 프로그래

밍 언어 교육과 관련된 몇 가지 연구 아이디어를 소개한다.

감사의 글

제1저자의 연구는 2018학년도 한경대학교 연구년 경비의 지원에 의한 것임. 제2저자의 연구는 2019학년도 한남대학교 연구지원에 의한 것임.

Gyesik Lee received his bachelor degree in the Dept. of Mathematics from Seoul National University in 1992. He received his M.S. degree and the Ph.D. degree in the Dept. of Mathematics and Computer Science from University of Muenster in 1996 and 2005, respectively. He had research positions at INRIA, AIST, and Seoul National University. He is currently a professor in the School of Computer Engineering & and Applied Mathematics at Hankyong National University. His research interests include logic in computer science, data analysis.
E-mail address: gslee@hknu.ac.kr

Hwajeong Kim received her bachelor and M.S. degrees in the Department of Mathematics from Seoul National University in 1993 and 1996, respectively. She received her Ph.D. degree in the Department of Mathematics from Saarland University in 2004. She had research positions at Humboldt University and Seoul National University. She is currently a professor in the Department of Mathematics at Hannam University. Her research interests include differential geometry and manifold theory.
E-mail address: hwajkim@hnu.kr