프로그래밍 언어의 메타이론의 정형화에 대한 locally nameless 방식과 두 종류의 변수를 사용하는 nominal 방식의 비교 분석

이계식
일본 국립 산업기술총합연구소 (AIST)
gyesik.lee@aist.go.jp

A comparison between locally nameless approach and nominal approach with two sorts of variables to formalization of metatheory of programming languages

Gyesik Lee
National Institute of Advanced Industrial Science and Technology (AIST), Japan

1. 서 론

프로그래밍 언어와 관련된 증명들이 점점 복잡해지고 길어짐에 따라 프로그램 전체에 대한 조망은 보다 어려워질 수밖에 없다. 작은 실수 하나가 프로그램 전체에 미치는 영향도 함께 증가한다따라서 정의와 정리 및 증명을 보다 체계적이면서 일관되게 유지하고,그것들의 상호 유기적인 관계를 보다 효과적으로 입증하기 위한 요구가 점점 커지고 있다 이에 대한 하나의 해결책으로 기계화된 자동화된 증명, 즉 컴퓨터를 이용한 엄밀한 증명이 제시되었다 1960년대 후반 드 브로인(de Bruijn)이 고안한 Automath [1]을 시작으로 해서 자동화된 증명보조(automated proof assistant) 툴들이 보다 효과적이면서 엄밀한 증명을 완성시켜 나가고 있다

이와 더불어 두 분야, 즉 프로그래밍 언어와 자동화된 증명보조를 상호 보완 발전시키려는 시도가 최근에 시작되었다. 대표적으로 POPLmark challenge를 언급할 수 있다. 즉, 프로그래밍 언어의 메타이론의 기계화 정도를 평가하려는 시도가 타입이론(type theory) 분야와 프로그래밍 언어 분야 자체에서 이루어지고 있다. 프로그래밍 언어의 정형화(formalization)를 평가하는 기준으로는① 복잡하지 않은 인프라 구조, ② 일상의 프로그래밍 언어로부터 너무 동떨어지지 않은 언어체계, ③ 쉬운 접근성이 제시되어 있다. 이 기준들을 만족시키는 메타언어와 그 특성에 대한 많은 연구가 이루어지고 있으며 변수묶기(variable binding)의 구현이 중요 과제 중의 하나로 부상했다

본 논문에서는 α-전환의 구현과 관련된 최근의 두 시도인Aydemir et al. [2]의 locally nameless 방식과 Herbelin과 Lee [3]의 두 종류의 변수를 사용하는nominal 방식을 장단점을 λ-calculus의 구현을 예제로 해서 비교 분석한다 두 시도 모두 Coq 증명보조 툴 [4]을 이용하였다.

2. Locally nameless식 접근

'드 브로인 인덱스'(de Bruijn index) 방식을 사용하면 α-동치한 λ-표식(λ-term)들은 동일한, 즉 유일한 방식으로 표현된다 따라서 변수치환의 필요성이 아예 발생하지 않는다드 브로인 인덱스를 사용하는 방식은 상당히 기계적인 언어체계(syntax)를 제공한다. 표현하고자 하는 표식의 오른쪽에서 시작하여 묶임수준(binding level)을, 즉 변수의 왼편에 자신의 묶임을 제외하고 몇 번의 변수묶임이 이루어졌는가를 숫자로 0,1,2,3등으로 나타낸다. 예를 들어 $\lambda x.(yx)$를 $(\lambda.10)$으로 표현하는데 숫자 0은 변수 x가 최초로 묶였음을 나타내고, 숫자 1은 y 자신을 기준으로해서 왼편에 변수x가 한 번 묶여 있음을 의미한다. 즉, 숫자는 주어진 변수의 일종의 묶임수준(binding level)을 나타낸다. 하지만 이러한 기계성은 드 브로인 인덱스를 따르는 표현들이 실용상의 프로그래밍 언어의 그것들과는 매우 이질적이라는 치명적인 한계를 갖고 있다 이러한 한계를 극복하려는 시도가 바로locally nameless 접근이다. 이 접근에서 변수묶기는 드 브로인 인덱스 형식을 따르지만 자유 변수는nominal 형식, 즉, 일상에서처럼

x, y, z 등의 구체적인 이름을 사용한다. 아래는 Coq 코드이다. var은 변수들의 집합을 의미한다.

```
Inductive lambda_term : Set :=
| Var : var -> lambda_term
| App : lambda_term -> lambda_term -> lambda_term
| Lambda : var -> lambda_term -> lambda_term.
```

3. 두 종류의 이름을 사용하는 nominal식 접근

Herbelin과 Lee [3]는 숫자 대신에 새로운 종류의 이름을 사용하는 방식을Coq에서 구현하였다. 즉 자유 변수와 묶인 변수에 각기 다른 종류(different sort)의 이름을 부여한다. 가독성 및 일상의 프로그래밍 언어와의 유사성은 자유 변수와 묶인 변수를 구분하는 관습을 따르면서 자연스럽게 보장되었다.사실 변수의 구분은 메타논리적으로도 의미가 있다. 앞서 설명하였듯이 묶인 변수는 일종의 자리지킴이(place holder)에 불과하다. 즉 어떤 집합의 전부를 대상으로 한다 반면에 자유 변수는 임의지만 어떤 집합의 특정한 하나의 원소를 대변한다.따라서 x, y 등의 서로 다른 이름은 일반적으로 서로 다른 값을 내포한다. 아래는 Coq 코드이다. fvar과 bvar는 각각 자유 변수와 묶인 변수의 집합을 의미한다.

```
Inductive pre_nterm : Set :=
| NameFvar : fvar -> pre_nterm
| NameBvar : bvar -> pre_nterm
| NameApp : pre_nterm -> pre_nterm -> pre_nterm
| NameLambda : bvar -> pre_nterm -> pre_nterm.
```

4. 활용성 및 결론

묶인 변수의 구현으로 제안된 locally nameless와 두 종류의 변수를 사용하는 방식 모두 일상 활용에 훌륭하게 사용될 수 있다. 전자의 경우 [2]를 비롯하여 이미 다양한 형식으로 그 응용성을 인정받았다. 후자의 경우는 [3]에서 수리논리학에서 다뤄지는 일차논리체계(first-order logic)의 언어체계와 그것의 의미론(semantics)을 완벽하게 구현해낼 수 있음을 입증하면서 그 가능성을 보였다.

결론적으로 변수충돌의 해결책은 본질적으로 자유 변수와 묶인 변수의 구문론적 구분에 있다고 말할 수 있다. 하지만 어떤 방식이 보다 우수하다의 식으로 결론을 내리는 건 어려워 보인다.두 시도 모두 장단점을 갖고 있으며 어디에 어떤 목적으로 활용되는가에 따라 무엇을 사용할 것인가를 결정할 필요가 있어 보인다. 저자의 경험에 의하면 구문론적인(syntactic) 정형화를 시도하는 경우에는locally nameless가, 의미론적(semantic)인 정형화의 경우에는nominal 방식이 좀 더 원래의 의도에 부합하는 것처럼 보인다.

5. 참고 문헌

[1] N. G. de Bruijn, AUTOMATH, a language for mathematics. Technical Report 68-WSK-05, T.H.-Reports, Einhoven University of Technology, 1968.

[2] B. Aydemir, A. Charguéraud, B. C. Pierce, R. Pollack, and S. Weirich. Engineering formal metatheory. In *POPL '08: Proceedings of the 35th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 3-15. ACM Press, 2008.

[3] H. Herbelin and G. Lee. Semantical normalisation using Kripke models for full predicate logic (a case study on the representation of binders). In preparation, 2008.

[4] Y. Bertot and P. Castéran. *Interactive Theorem Proving and Program Development (Coq'Art: The Calculus of Inductive Constructions)*, volume XXV of *EATCS Texts in Theoretical Computer Science*. Springer-Verlag, 2004.