하드웨어 기술을 위한 선형 람다 셈법의 타입 유추

임정표° 박성우
포항공과대학교
jplim@postech.ac.kr, gla@postech.ac.kr

Type Inference for a Linear Lambda Calculus for Hardware Description

Jeongpyo Lim° Sungwoo Park
Pohang University of Science and Technology

산업체에서 자주 사용되는 하드웨어 기술 언어들(Verilog 또는 VHDL)의 복잡한 문법과 불분명한 의미론을 극복하기 위하여 최근에는 함수형 언어들의 특성을 도입한 새로운 하드웨어 기술 언어들을 만들기 위한 연구들이 계속 수행되고 있다. 특히, 기존 함수형 언어를 변형하여 새로운 언어를 정의하거나 또는 하드웨어 기술 기능을 함수형 언어 내부에 내장하려는 시도를 많이 하고 있다.

이에 한 걸음 더 나아가 함수형 언어들의 핵심적인 셈법이라 할 수 있는 람다 셈법(Lambda Calculus)을 변형하여 하드웨어를 기술하는 새로운 셈법을 정의하려는 시도도 등장하였다. '선형 람다(Linear Lambda)'라고 불리는 이 셈법은 선형 논리(Linear Logic)와 람다 셈법을 융합한 것으로서, 하드웨어를 람다 셈법에서의 함수로 볼 수 있으며 함수 호출은 함수와 인자라는 두 요소(하드웨어 또는 비트스트림으로 해석할 수 있음)의 결합으로 볼 수 있다는 관점에서 출발한다. 즉, 선형 람다에서는 하드웨어를 함수로 표현하며 하드웨어에 비트스트림을 입력하여 결과 스트림을 얻는 것을 함수에 인자를 적용하여 결과값을 얻는 것으로 표현한다. 예를 들어 간단한 함수

$$\lambda x : 1.(x, x)$$

는 비트 1개를 입력으로 받은 뒤 입력으로 받은 것과 똑같은 비트 2개를 출력하는 하드웨어라고 볼 수 있다. 또한

$$(\lambda x : 1.(x, x))\ 0$$

는 위 하드웨어에 비트 0을 입력한 결과, 즉 값이 0인 2개의 비트라고 생각할 수 있다. 이 때 비트스트림과 달리 하드웨어는 오직 한 번만 사용할 수 있는 자원으로 보아야하기 때문에 기존 람다 셈법의 타입 시스템과 달리 선형 람다는 선형 논리를 사용하여 하나의 함수는 표현식 내부에서 오직 한 번만 사용해야 한다는 규칙을 반영한 타입 시스템을 정의한다는 점이 특징이다. 그리고 선형 람다는 람다 셈법과는 약간 다른 방식으로 고차 함수의 사용을 지원한다. 예를 들어 간단한 고차 함수

$$g = \lambda f : 1 \to 1.f\ (f\ 0)$$

는 기존 람다 셈법에서는 문제가 없는 함수이지만 선형 람다에서는 함수가 하드웨어를 표현하기 때문에 위 함수처럼 하나의 하드웨어(함수 f)를 표현식 내부에서 두 번 사용하는 것은 논리적으로 옳지 않다. 위와 같은 함수는 선형 람다에서 아래와 같이 표현해야 한다.

$$g' = \hat{\lambda} f_1 : 1 \to 1.\hat{\lambda} f_2 : 1 \to 1.f_1\ (f_2\ 0)$$

선형 람다에서의 고차 함수는 복잡한 하드웨어를 좀 더 쉽게 표현할 수 있도록 도와주는 역할을 한다. 또한 선형 람다에서도 기존 람다 셈법과 마찬가지로 다형성(polymorphism)을 쉽게 도입할 수 있다. 선형 람다에서의 다형성은 구조는 동일하지만 비트스트림의 개수가 다른 하드웨어들을 동일한 표현식을 사용하여 기술할 수 있게 지원하는 역할을 한다. 예를 들어 다형성이 지원되지 않는 경우에는 입력 비트스트림을 그대로 출력시키는 함수를

$$f_1 = \lambda x : 1.x$$

$$f_2 = \lambda x : 1 \times 1.x$$

과 같이 입력 타입에 따라 여러 개 정의해야 하지만 다형성이 지원될 경우 오직 하나의 함수

$$f = \Lambda\alpha.\lambda x : \alpha.x$$

만으로 표현할 수 있다. 따라서 다형성을 통해 중복된 표현식을 제거하여 훨씬 더 간결하게 하드웨어를 기술할 수 있다는 장점이 있다.

그러나 기존 선형 람다는 모든 표현식의 타입을 프로그래머가 직접 기술해야 한다는 문제점을 가지고 있다. 실제로 선형 람다 기반의 언어를 사용하여 FFT(Fast Fourier Transform)를 구현하였을 때 전체 코드의 절반 이상을 타입 기술 영역이 차지하고 있는 것을 볼 수 있었다. 이와 같이 과도한 타입 기술은 프로그래머에게 불필요한 짐이 될 뿐만 아니라 코드를 이해하기 힘들게 만든다는 단점이 있다.

따라서 본 논문에서는 선형 람다의 불필요한 타입 기술을 해소하기 위하여 자동으로 타입을 유추하는 타입 유추 알고리즘을 제안한다. 이를 위해 우선 기존 선형 람다의 핵심 문법 구조 및 타입 체계를 타입 유추 방식에 맞게 변형하였다. 기존 선형 람다는 모든 표현식에 타입 기술이 존재한다고 가정하는데 반해 본 논문의 선형 람다는 불필요한 타입 기술 부분들을 제거하였으며, 각 타입에 대응하는 타입 변수들을 새롭게 도입하였다. 또한 let 표현식을 도입하여 let-polymorphism을 지원할 수 있도록 수정하였다. 이를 바탕으로 하여 선형 람다의 타입 유추 알고리즘인 lW을 정의하였다. lW은 람다 셈법의 타입 유추 알고리즘인 W를 변형한 알고리즘으로서 기존 W 알고리즘에 선형 논리를 결합한 형태이다. 알고리즘 lW의 형식은 아래와 같다.

$$(S, \tau, \Delta_r) = lW(\Gamma, \Delta, \Omega, e)$$

lW은 총 네 개의 인자를 필요로 한다. 첫 번째 인자 Γ는 비트스트림을 나타내는 변수들의 타입 정보를 유지하는 타입 문맥이고, 두 번째 인자 Δ는 함수(선형 람다에서는 하드웨어를 의미)를 나타내는 변수들의 타입 정보를 유지하는 타입 문맥을 나타낸다. 기존 람다 셈법과 달리 선형 람다는 일반 변수와 함수를 나타내는 변수를 구분해야 하기 때문에 이와 같이 각 변수들의 타입 정보들을 따로 관리한다. 세 번째 인자 Ω는 let 표현식에서 정의되었기 때문에 다형적으로 사용될 수 있는 변수들의 타입 정보를 유지하는 타입 문맥을 의미하며 마지막 인자 e는 타입을 유추할 표현식을 나타낸다. 알고리즘은 타입 유추 결과로서 S, τ, 그리고 Δ_r을 반환한다. 이 때 S는 타입 문맥 내의 타입 변수와 알고리즘 내부에서 도입된 타입 변수의 타입 유추 결과를 나타내고, τ는 입력으로 들어온 표현식 e의 타입을 의미하며 Δ_r은 Δ 내부의 변수들 중에서 표현식 e에서 사용되지 않은 변수들을 의미한다. 알고리즘은 타입 유추가 불가능한 경우에는 에러를 반환한다. 또한 본 논문에서는 알고리즘의 정확성을 검증하기 위하여 알고리즘의 안전성(soundness) 및 완전성(completeness)을 증명하였다.

위와 같은 타입 유추 알고리즘을 선형 람다에 적용함으로서 프로그래머가 불필요한 타입 기술에 매달리는 낭비를 줄이고 오직 프로그램 구현에만 집중할 수 있도록 지원할 뿐 아니라, 코드를 훨씬 더 간결하게 만들어 주기 때문에 가독성을 높일 수 있다. 하지만 아직도 선형 람다가 실용적인 언어로서 사용되기 위해서는 여전히 많은 점이 보완되어야 할 것이다. 특히, 모듈 프로그래밍을 지원하는 부분이 전무하기 때문에 앞으로 모듈 시스템을 선형 람다에 탑재하는 연구도 좋은 연구 방향이 될 수 있을 것이다.