람다-추상화, 함수 변수 그리고 단순 단일화를 포함한 논리형 언어

Dale Miller

Abstract

특히 대상 언어가 속박변수와 스코프의 개념을 포함할 때에, 항 중에 함수 변수와 λ-추상화도 있는 논리형 프로그래밍 언어가 좋은 메타 프로그래밍 언어를 만든다는 주장이 다른 곳에서 제기되어 왔습니다. It has been argued elsewhere that a logic programming language with function variables and λ-abstraction within terms makes a good meta-programming language, especially when an object language contains notions of bound variables and scope. 논리형 언어 λProlog와 그와 관련된 Elf와 Isabelle 체계는 함수 변수와 람다-추상화를 포함한 메타 프로그램들을, 고차 단일화 알고리즘을 내장함으로써, 제공합니다. The λProlog logic programming language and the related Elf and Isabelle systems provide meta-programs with both function variables and λ-abstractions by containing implementations of higher-order unification. 이 논문은 L_λ라 불리는 – 함수 변수와 람다-추상화를 포함하지만 함수 변수의 나타남에 일부 제한을 가하는 – 논리형 언어를 발표할 것입니다. This paper presents a logic programming language, called L_λ, that also contains both function variables and λ-abstractions, although certain restrictions are placed on occurrences of function variables. 이러한 제한을 가함으로써, L_λ의 구현체는 완전한 고차 단일화 알고리즘을 요구하지 않습니다. As a result of these restrictions, an implementation of L_λ does not need to implement full higher-order unification. 대신, 속박변수들의 이름과 스코프를 존중하는 일차 단일화 알고리즘의 확장이 요구될 뿐입니다. Instead, an extension to first-order unification that respects bound variable names and scopes is all that is required. 그러한 단일화 문제들은 결정가능하다는 것과 unifier가 존재할 때 m.g.u.를 가진다는 것이 증명되었습니다. Such unification problems are shown to be decidable and to posses most general unifier when unifier exists. 후술할 단일화 알고리즘과 논리형 언어의 인터프리터는 정확하다는 것이 증명되었습니다. A unification algorithm and logic programming interpreter are described and proved correct. 메타 프로그래밍 언어로서 L_λ를 사용한 몇 가지 예들이 발표될 것입니다. Several examples of using L_λ as a meta-programming language are presented.

1 서론

메타 프로그래밍 언어는 프로그램, 논리식, 타입, 증명과 같은 구문론적 구조들을 나타낼 수 있어야 하고 조작할 수 있어야 합니다. A meta-programming language should be able to represent and manipulate such syntactic structures as programs, formulas, types and proofs. 이 구조들의 공통적인 특징은 추상화, 스코프, 속박변수와 자유변수, 대입 인스턴스 그리고 속박변수의 이름을 바꾸는 것까지의 동등성의 개념과 관련되었다는 것입니다. A common characteristic of all these structures is that they involve notions of abstractions, scope, bound and free variables, substitution instances, and equality up to alphabetic change of bound variables. 대부분의 컴퓨터 프로그래밍 언어들에서 사용할 수 있는 자료형들이, 당연히, 이러한 종류들의 구조들을 충분히 표현할 수 있음에도 불구하고, 그러한 자료형은 그러한 자료형들은 이 공통적인 특징에 대하여 직접적인 지원을 하지 않습니다. Although the data types available in most computer programming languages are, of course, rich enough to represent all these kinds of structures, such data types do not have direct support for these common characteristics. 예를 들어, Lisp에서 1차 논리식들을 표현할 수 있음은 자명하지만, 항을 논리식에 대입하되 속박변수와 충돌하지 않도록 신경써서 정확히 하는, α-변환까지 동등함을 시험하는, 그리고 특정 변수의 나타남이 자유로운지 묶여있는지를 결정하는 Lisp 프로그램을 작성하는 것은 더 복잡한 문제입니다. For example, although it is trivial to represent first-order formulas in Lisp, it is a more complex matter to write Lisp programs that correctly substitute a term into a formulas (being careful not to capture bound variables), to test fort the equality of formulas up to alphabetic variation, and to determine if a certain variable's occurrences is free or bound. 이러한 상황은, Lisp 대신, Pascal, Prolog, 그리고 ML 같은 다른 프로그래밍 언어들에서도 (자연 연역) 증명과 같은 구조를 다룰 때에 똑같이 일어납니다. This situation is the same when structures like programs or (natural deduction) proofs are to be manipulated or when other programming languages, such as Pascal, Prolog, and ML, replace Lisp.

메타 프로그래밍 언어가 대상-레벨의 구문론의 여러 가지 면에 대한 언어-레벨 지원을 가지는 것은 바람직합니다. It is desirable for a meta-programming language to have language-level support for these various aspects of object-level syntax. 이러한 구조들을 나타내는 공통적인 프레임워크는 무엇일까요? What is a common framework for representing these structures? Church, Curry, Howard, Martin-Löf, Scott, Strachey, Tait 그리고 다른 사람들의 연구에 따르면, 타입 있는 람다-계산법과 타입 없는 람다-계산법이 이 구조들 전부에 대한 공통적인 문법

적 표현법을 제공한다고 합니다. Early work by Church, Curry, Howard, Martin-Löf, Scottm, Strachey, Tait, and others concluded that typed and untyped λ-calculi provide a common syntactic representation for all these structures. 그러므로 그러한 λ-계산법들을 이용하여 직접적으로 항을 표현할 수 있는 메타-프로그래밍 언어는 이 저자들에 의하여 묘사된 기술들을 이용하여 이 구조들을 나타내는 하는 데 사용될 수 있을 것입니다. Thus a meta-programming language that is able to represent terms directly in such λ-calculi could be used to represent these structures using the techniques described by these authors.

λ-항에 대한 자료형을 설계하는 데 생기는 문제 중 하나는 그것들을 해체하는 데 쓰이는 방법들은, 보통 α-변환을 포함하는, λ-항의 동등성 개념의 측면에서 변하지 않아야 합니다. One problem with designing a data type for λ-terms is that methods for destructuring them should be invariant under the intended notion of equality of λ-terms, with usually includes α-conversion. 그러므로, λ-항 $\lambda x.fxx$을 그것의 속박변수 x와 몸통 fxx로 해체하는 것은 α-변환에 대하여 달라질 수 있습니다: 이 항은 $\lambda y.fyy$로 α-변환 가능하지만 이 동등한 항을 분해한 결과는 동등한 답을 산출하지 않습니다. Thus, destructuring the λ-term $\lambda x.fxx$ into its bound variable x and body fxx is not invariant under α-conversion: this term is α-convertible to $\lambda y.fyy$ but the result of destructuring this equal term do not yield equal answers. 비록 이름 없는 모조품들을 이용하는 것[2]이 – 이 두 항 모두를 $\lambda\,(f11)$와 같이 동등한 구조로 나타내기 때문에 – 이러한 문제들을 단순화하는 데 도움이 될지라도, 이 표현은 λ-항들에 원하는 모든 범위의 연산들을 나타내기 위해 여전히 꽤 복잡한 조작을 필요로 합니다. Although the use of nameless dummies [2] can help simplify this one problem since both of these terms are represented by the same structure $\lambda\,(f11)$, that representation still requires fairly manipulations to represent the full range of desired operations on λ-terms. α-변환과 β-변환까지 동등하게 다루는 λ-항들에 대한 더 높은 레벨의 접근법은 단순 타입 λ-항들의 단일화를 이용했습니다 [13, 20, 33, 34]. A more high-level approach to the manipulation of λ-terms modulo α and β-conversion has been the use of unification of simply typed λ-terms [13, 20, 33, 34]. Huet과 Lang[14]은 그러한 접근법들이 – 이차-매칭으로 제한했을 때 – 어떻게 단순한 함수형 프로그램들과 명령형 프로그램들을 분석하고 조작할 수 있는 데 사용될 수 있는지를 기술했습니다. Huet and Lang [14] described how such an approach, when restricted to second-order matching, can be used to analyze and manipulate simple functional and imperative programs. α, β 그리고 η-변환까지 동등하게 다루는 단일화에 대한 그것들의 의존성은 그것들의 메타-프로그램을 우아하게, 적기 단순하게, 증명하기 쉽게 만들었습니다. Their reliance on unification modulo α, β and η-conversion made their meta-programs elegant, simple to write, and easy to prove correct. 그들은 이차-매칭을, 그것이

일정량의 템플릿 매칭 프로그램 변환을 구현할 수 있을 만큼 충분히 강하고 결정 가능하기 때문에, 선택했습니다. They chose second-order matching because it is strong enough to implement a certain collection of template matching program transformations and it is decidable. 단순 타입 λ-항들의 2차 이상의 일반적인 단일화 문제는 결정 불가능합니다[8]. The general problem of the unification of simply typed λ-terms of order 2 and higher is undecidable [8].

메타 프로그래밍에서 λ-항 단일화의 사용처는 최근 몇 논문과 컴퓨터 시스템에서 더욱 넓어졌습니다. The use of λ-term unification in meta-programming has been extended in several recent papers and computer systems. [6, 9, 10, 21]에서 증명 보조기와 프로그램 변환기를 포함하는 다양한 메타 프로그램들은 단순 타입 λ-항들의 단일화를 수행하는 논리형 언어 λProlog로 작성되었습니다[24]. In [6, 9, 10, 21] various meta-programs, including theorem provers and program transformers, were written in the logic programming language λProlog [24], which performs unification of simply typed λ-terms. Paulson은 증명보조기 Isabelle에서 그러한 단일화를 활용했습니다[28, 29]. Paulson [28, 29] exploited such unification in the theorem proving system Isabelle. Pfenning과 Elliot 곱 타입들도 사용된다고 주장했습니다[32]. Pfenning and Elliot [32] argued that product types are also of