論文 / 著書情報
Article / Book Information

題目(和文)	バージョンを言語要素に持つプログラミング言語の研究
Title(English)	A Programming Language with Versions
著者(和文)	田辺裕大
Author(English)	Yudai Tanabe
出典(和文)	学位:博士(理学), 学位授与機関:東京工業大学, 報告番号:甲第12329号, 授与年月日:2023年3月26日, 学位の種別:課程博士, 審査員:増原 英彦,鹿島 亮,南出 靖彦,脇田 建,西崎 真也,五十嵐 淳
Citation(English)	Degree:Doctor (Science), Conferring organization: Tokyo Institute of Technology, Report number:甲第12329号, Conferred date:2023/3/26, Degree Type:Course doctor, Examiner:,,,,,
学位種別(和文)	博士論文
Category(English)	Doctoral Thesis
種別(和文)	論文要旨
Type(English)	Summary

（博士課程）
Doctoral Program

論 文 要 旨

THESIS SUMMARY

系・コース： Department of, Graduate major in	数理・計算科学 系 数理・計算科学 コース	申請学位（専攻分野）： Academic Degree Requested	博士 Doctor of （理学/Science）
学生氏名： Student's Name	田邉 裕大	指導教員（主）： Academic Supervisor(main)	増原 英彦
		指導教員（副）： Academic Supervisor(sub)	

要旨（英文 800 語程度）
Thesis Summary（approx.800 English Words ）

One of the problems facing software developers today is the cost of keeping dependent packages up-to-date. While updates bring many improvements, versions can also cause problems with client software. Although such incompatible updates are common, the differences between the old and new versions are often implicitly described, thus making it difficult to debug. This problem has been getting worse in recent years due to the increasing number of transitive package dependencies. One of the main factors making updates difficult is that most software systems insist on making only a single version of a particular package available anywhere. This fact leads to the lack of support for the simultaneous use of multiple versions in most programming languages. Therefore, even when only a small portion of a program needs to be updated, developers will face much work, which tends to delay the adoption of new versions.

Our goal is to establish the basis for a new language design called programming with versions, which allows multiple versions of a single package and encourages developers to update software. In this regard, we develop a programming language called VL, a functional language with versions as an intrinsic language element. In contrast to existing languages, which assume a single version for each package, VL allows external values to refer to an individual version of external modules. Lambda VL is developed as a core calculus to realize programming with versions, and we establish the proper notion of type safety for an unprecedented language. Furthermore, for realizing programming with versions in general-purpose functional languages, the author presents an inter-compilation method between a lambda-based functional language and Lambda VL and a type inference algorithm for the Lambda VL.

Lambda VL is an extension of the linear lambda calculus, allowing a single value to have multiple variations of versions. Such values are called versioned values and store version-specific definitions in record-like entities, along with version labels indicating which version the value exists. The function application of versioned values represents multiple possibilities of computations of a function and an argument. The internal computation is extracted by specifying the label in the same way as in a normal record. The type safety of Lambda VL programs is explained by version consistency: it ensures that every subterm has a consistent version definition, thereby preventing Lambda VL programs from being evaluated with versions where no definition exists. Furthermore, the type system is designed as an instance of coeffect calculus; a substructural calculus can analyze various computational resources.

We developed a compilation from lambda-calculus-based surface language modules to a Lambda VL program that behaves in a cross-version manner. This compilation consists of two compilations: (1) Girard's translation from a single version of the surface language program to GrMini, a subset of Lambda VL. We define the translation as a generalization of the well-known Girard's transformation for linear lambda calculus. The compilation intuitively replaces lambda abstractions with cotextual-let and dually inserts a promotion for the function application argument. The extended Girard's translation allows functions to capture version constraints of their argument value in the type system. (2) Bundling that bundles top-level declarations of each version of a GrMini program into a single versioned value. Bundling allows a top-level symbol with the same name across versions to behave as a versioned value representing multiple possibilities of computations.

We develop algorithmic type inference rules based on the declarative type system following HM type inference. Versioning constraints are generated from either (1) bundling that generates a dependency on a particular version for a top-level symbol or (2) type inference that generates version consistency among variables. These constraints will be resolved after all modules' type inference/bundling is completed.

Finally, the author implements a VL language system with all the above concepts of programming with versions. The VL system is implemented by Haskell (GHC 9.2) and uses z3 as the constraint solver. The code generation specializes in a Lambda VL program into a version-specialized Haskell program using the solution of z3. Through some examples, we observe how the proposed language system guarantees safety in multiple versions. Furthermore, since the initial implementation strictly guarantees version consistency, we introduce a syntax extension that can communicate the user's intent to the type inference system and enable the simultaneous use of multiple versions.

In this paper, we show that for traditional functional languages such as Haskell and the ML language family, multiple versions of a single package can be used without compromising type safety. Although this paper focuses on Haskell-like minimal languages, it will be possible to support higher-level language constructs such as user-defined data types and version polymorphisms. The basic idea that values have versions also provides a theoretical basis for more granular dependency-checking features such as link-time dependency checking and semantic versioning into language semantics.

備考 ： 論文要旨は、和文 2000 字と英文 300 語を 1 部ずつ提出するか、もしくは英文 800 語を 1 部提出してください。
Note : Thesis Summary should be submitted in either a copy of 2000 Japanese Characters and 300 Words (English) or 1copy of 800 Words (English).