論文の内容の要旨

Dependent Type Inference for Program Verification (プログラム検証のための依存型推論)

海野 広志

Our society relies on computer systems, which are composed of many programs. Therefore, it is important to verify correctness of those programs. Currently, system developers rely on manual program inspection and testing to ensure that their programs work as per their specifications. However, manual inspection and testing may overlook potential bugs of programs. In contrast, formal verification methods can exhaustively verify correctness of programs, and are expected to play a central role in program verification. In the framework of type-based program verification, specifications are expressed as types, and type inference is used to verify that programs conform to those specifications. Dependent types are a special kind of types that may depend on values, and are useful for statically verifying detailed specifications of programs such as the absence of array bounds and pattern match errors. Compared to other formal program verification methods such as model checking and abstract interpretation, the method based on dependent types can straightforwardly deal with extended features of programming languages such as higher-order functions and recursive data structures.

For designing a type inference algorithm for a dependent type system, we need to put restrictions on the type inference algorithm since the typability problem of a dependent type system is usually undecidable. However, restrictions imposed by existing work are not desirable for program verification. For example, Dependent ML (DML) proposed by Xi et al. requires users to declare dependent types for all functions, so that the burden imposed on users is too heavy. The type inference algorithm for sized types proposed by Chin et al. fixes the shape of dependent types a priori, and tries to infer as precise dependent types as possible. Thus, verifiable properties and data structures are limited to predefined ones in their algorithm, and it may waste computational costs for inferring unnecessarily precise dependent types for program verification.

In this thesis, we propose a novel program verification method based on dependent types which solves the problems of the existing methods mentioned

above. We introduce dependent types to a higher-order functional language with recursive data structures, and propose two type inference algorithms. The first algorithm infers dependent types which are precise enough to verify given specifications of a program on demand. The algorithm basically infers an output specification of a function from a call-site of the function, and propagates that specification backward to infer the input specification. We have implemented a prototype type inference system, and conducted two experiments. In the first experiment, our system successfully verified that insertion and merge sort programs always return an ordered list by automatically inferring dependent types of the auxiliary functions of these programs. Note that the type inference algorithm for sized types cannot verify such property since the algorithm can only infer the lengths of lists. DML requires users to write complex type annotations for verification of the sorting programs. In the second experiment, we have inferred dependent types of the standard list library functions for OCaml programming language by using call-sites of those functions collected from existing application programs written in OCaml. The results of the experiment indicate that our algorithm can infer preferable dependent types for not only program verification but also a documentation purpose.

One of the problems of existing type inference algorithms and our first algorithm described above is that if verification of a program fails, these algorithms cannot find out a reason of the failure automatically. Especially, the type inference algorithms cannot offer enough information to users for judging whether their programs are really incorrect: The type inference algorithms may reject a correct program due to the lack of analysis precision. In order to address the problem, in this thesis, we propose the second type inference algorithm which can find a counter-example against typability of an ill-typed program, namely a program input that leads to a run-time error, as an explanation of why the program is ill-typed. The algorithm can iteratively refine dependent types with interpolants until the type inference succeeds or a counter-example is found. We have implemented a prototype type inference system and conducted two experiments to evaluate the second type inference algorithm. In the first experiment, our system successfully verified that array programs obtained from DML sample programs cause no array bounds error without requiring type annotations. We then intentionally modified the array programs to obtain incorrect programs, which actually cause array bounds errors. We analyzed these programs with our system and successfully found counter-examples. In the second experiment, we successfully verified that insertion and merge sort programs always return an ordered list.