外国語要旨

学位論文題目：Practicable Type Debugging for Functional Languages
氏名：Kanae Tsushima

This thesis presents how to build a practicable type debugger. From the time the Hindley-Milner type system was first proposed, programmers have received benefits from types. At the same time, they have to struggle with type errors. Many approaches have been developed to help programmers locate the source of type errors. Although their implementations help programmers a lot, existing compilers often lack such support. We feel this situation puts too much of a burden on programmers: debugging an ill-typed program takes up a lot of their time, and compiler's error messages are too difficult for many new learners to understand. This situation is a shame for many languages.

To address this situation, we believe practicable type debugging is needed. First, we establish a manifesto of practicable type debugging. The properties of the manifesto can be grouped into two categories. One category is the producer side, where the properties focus on the implementation. By satisfying these properties, a type debugging system can be applied to many languages. The other category is the consumer side, where the properties focus on the usability. If a type debugger is not user-friendly and forces the programmers to deal withtoo big a burden, the programmers feel that debugging by hand would be better. Therefore, the usability of a type debugger is a crucial factor.

The main part of this thesis consists of two parts.

First, we focus on the producer side of type debugging. To this end, we propose a type debugger *without* implementing any dedicated type inferencer. Conventional type debuggers require their own type inferencers separate from the compiler's type inferencer. The advantage of our approach is threefold. First, by *not* implementing a type inferencer, it is guaranteed

that the debugger's type inference never disagrees with the compiler's type inference. Second, we can avoid the pointless reproduction of a type inferencer that should work precisely as the compiler's type inferencer. Third, our approach can withstand updates of the underlying language. The key element of our approach is that the interactive type debugging, as proposed by Chitil, does not require a type inference tree but only a tree with a certain simple property. We identify the property and present how to construct a tree that satisfies this property using the compiler's type inferencer. The property shows us how to build a type debugger for various language constructs. In this paper, we describe our idea and first apply it to the simply typed lambda-calculus. After that, we extend it with let-polymorphism and objects to see how our technique scales.

Second, we focus on the customer side of type debugging. To this end, we propose a *weighted* type error slicer. Conventional type error slicers enable users to narrow the area for type debugging. The advantage of our approach is the *weight* for each subexpression of slices, which here means the level of relation to the type errors. By weighted type error slices, type debuggers can ask questions in an order that relates to the type errors. The problem with conventional type error slicers is that the slice becomes large when the original ill-typed program is too big. Using weighted type error slicing solves this program by the ordering.