Extensible Overloaded Functions

(extended abstract)

Maria João Frade

Departamento de Informática, Universidade do Minho, Campus de Gualtar, 4710-057 Braga, Portugal,
mjf@di.uminho.pt

Type systems are pervasive in modern functional programming. These systems support inductive datatypes and the definition of functions by pattern matching. However, there are cases where the standard type systems are not flexible enough and some interesting programs are rejected or are assigned some uninformative types. So, it is important to devise flexible type systems that allow for more informative typings.

Constructor subtyping [1] is a basic form of subtyping, which formalizes the view that an inductively defined set τ is a subtype of an inductively defined set σ if σ has more constructors than τ. As such, constructor subtyping captures in a type-theoretic context the ubiquitous use of subtyping as inclusion between inductively defined sets. For example, constructor subtyping enforces subtyping from natural numbers to integers or from non-empty lists over σ to lists over σ, and it is adequate to work with mutually recursive datatypes such as odd and even numbers. In this context, constructor subtyping provides a solution to the problem of datatypes in typed λ-calculi with subtyping. An important application of constructor subtyping is thus typed functional programming languages. Constructor subtyping combines subtyping between datatypes and overloading of constructors. In order to integrate constructor subtyping safely to typed λ-calculus, maintaining properties such as subject reduction and confluence of reductions in the resulting system, one needs to constrain the overloading of constructors to strict overloading [1], a form of guaranteeing coherence between domain and codomain of overloaded constructors.

Constructor subtyping is specially adequate for re-usability, allowing the definition of new types by restricting or by expanding the set of constructors of an already defined datatype. In order for constructor subtyping to be usable in practice, one must allow users to define overloaded and extensible recursive functions. More precisely, recursive definitions must be: overloaded (i.e. to have several types) and extensible (i.e. to be scalable from a datatype to another datatype). The combination of subtyping between datatypes and overloading of constructors allows the definition of new datatypes by restricting or by expanding the set of constructors of an already defined datatype. This flexibility in the definition of datatypes induces a convenient form of code reuse for recursive functions, allowing the definition of new functions by restricting or by expanding already defined ones.

We enrich a calculus featuring constructor subtyping with a mechanism to define extensible overloaded functions. We introduce the system $\lambda_{\mathsf{CS+fun}}$ a simply typed λ-calculus, *à la* Curry, with mutually recursive parametric datatypes, constructor subtyping and extensible overloaded recursive functions defined by patterns-matching. We formalize the concept of well-formed environment of function declarations and establish that under such environments the properties of confluence, subject reduction and decidability of type-checking hold. Moreover, we prove that the requirements imposed for the well-formed environments are decidable and show how standard techniques can still be used for compiling pattern-matching into case-expressions.

References

1. G. Barthe and M.J. Frade. Constructor subtyping. In D. Swiestra, editor, *Proceedings of ESOP'99*, volume 1576 of *Lecture Notes in Computer Science*, pages 109–127. Springer-Verlag, 1999.