Blame Calculus: Exercises

Philip Wadler

December 2011

Do *one* of the following two exercises.

Exercise 1 (more practical)

Pick a dynamically typed programming language of your choice (for instance, Javascript or Python) and use the blame calculus to devise a system of optional type declarations for the language; or devise a system for integrating a typed and untyped language (for instance, integrate C# and Iron Python under .net).

An optional type declaration

$$t : A$$

in the dynamically typed language, where t is a term and A is a type, should correspond to a cast

$$t : \star \Rightarrow^p A \Rightarrow^p \star$$

in the blame calculus, where p is a blame label reflecting the location in the program (for instance, the line number containing the optional declaration). You may implement optional type declarations by either a preprocessor or a library. Apply the notion of wrapping from [4] to provide dynamic checking of function types, and the notion of sealing from [6] to provide dynamic checking of polymorphic types. Apply the notion of blame so that when an optional declaration is violated it is indicated whether fault lies with the term contained in the declaration or the context containing the declaration.

A minimal solution will support function and polymorphic types. For additional credit, extend your system to support other types as appropriate to your choice of language, such as records or objects; or to support assigment, by permitting assignable variables and fields to have optional type declarations. You may find results in [1] and [2] useful. State what restriction of the original language you support; for instance, you may forbid

records and objects, forbid assignment, or only permit assignment to fields of an object within methods of that object.

Submit a short description of your system, examples demonstrating that it works, and the running code. Marks will be based on clarity, completeness, and correctness of your presentation.

Exercise 2 (more theoretical)

The principal theorem from [5] asserts observational equivalence between one cast and a pair of casts:

$$t : A \Rightarrow^p B \quad = \quad t : A \Rightarrow^p C \Rightarrow^p B \quad \text{when} \quad A \,\&\, B <:_n C$$

where t is a term, A, B, C are types, p is a blame label, $A <:_n B$ means A is a naive subtype of B, and $A \,\&\, B$ is the greatest lower bound of A and B with regard to naive subtyping. This theorem is asserted for a language with dynamic type $\star$ and function types $A \to B$, as described in [4] and [5]. Does the result extend to a language that also includes polymorphic types $\forall X.\, A$, as described in [6]? Either give a counter-example showing that the result does not hold, or give a proof that the result does hold. Marks will be based on clarity, completeness, and correctness of your presentation.

References

[1] Jeremy G. Siek and Walid Taha, Gradual typing for functional languages. *Scheme and Functional Programming Workshop*, 2006.

[2] Jeremy G. Siek and Walid Taha, Gradual Typing for Objects. *European Conference on Object-Oriented Programming* (ECOOP), pp. 2–27, 2007. (Springer Verlang LNCS 4609.)

[3] Philip Wadler, The Girard-Reynolds isomorphism (second edition). *Theoretical Computer Science*, 375(1-3): 201–226, 2007.

[4] Philip Wadler and Robert Bruce Findler, Well-typed programs can't be blamed. *European Symposium on Programming* (ESOP), pp. 1–16, 2009. (Springer Verlang LNCS 5502.)

[5] Jeremy G. Siek and Philip Wadler, Threesomes, with and without blame. *ACM Symposium on Priciples of Programming Languages* (POPL), pp. 365–376, 2010.

[6] Amal Ahmed, Robert Bruce Findler, Jeremy G. Siek, and Philip Wadler, Blame for all. *ACM Symposium on Priciples of Programming Languages* (POPL), pp. 201–214, 2011.