Safe, expressive language interoperability

Peter-Michael Osera
posera@cis.upenn.edu
University of Pennsylvania

November 14, 2011

Language interoperability is not a new research area for the programming languages community. In the early 2000s, considerable effort went into understanding how to efficiently marshal data between two interoperating languages. The "last word" in this area was the birth of the .NET framework which solved the problem by unifying the data model that interoperating languages utilize. Since then, both the .NET framework and the JVM have evolved to handle increasingly larger classes of programming languages, most recently dynamic languages with .NET's DLR and Java's Da Vinci machine project.

One of the biggest meta-problems that programming language researchers face when developing new languages with fancy type systems is that of *relevance*. More often than not, the advanced types of these languages make it difficult to understand how to extend these works into more mainstream languages so that they can be enjoyed by the masses. Because of this, it is difficult for a language designer to make the critical argument that their creation is *relevant and useful*.

Language interoperability makes this situation much more palatable. Imagine a world where your fancy linearly-typed or dependently-typed language could interoperate with the mainstream programming language you use for day-to-day work. As an end-user, this is great because the barrier to adopting the fancy language has been greatly lowered. But as a researcher, now we can leverage a mainstream programming language's libraries and general usability in order to generate a wealth of examples that demonstrate the utility of our language. The only catch is that the interoperability layer that we create between these two languages preserves the guarantees made by our fancy type system.

The key thing to note is that this concern we just described is not an implementation concern in the vein of the previous work into interoperability. Instead it is a concern of the *semantics* of our interoperability layers. As Matthew's and Findler's [1] pioneering work in this area showed, enforcing these semantics can be casted as a problem of enforcing contracts [2] between languages.

However, there remains much more work to be done in this area so that both programmer and PL researcher can benefit from language interoperability for advanced programming languages. In particular, understanding the checks that must be made in these interoperability layers for particular sorts of type systems is critical. The critical assumption we make in language interoperability is that the two languages are independent and one cannot affect the other except via crossing an interoperability boundary. Because of this restriction, we may need to sacrifice some expressiveness of our advanced programming language in order to preserve safety. This in turn can inform more fine-grained approaches to the merging of programming styles.

References

[1] J. Matthews and R. B. Findler. *Operational semantics for multi-language programs.* ACM Trans. Program. Lang. Syst., 31(3):144, 2009. ISSN 0164-0925.

[2] R. B. Findler and M. Felleisen. *Contracts for higher-order functions.* SIGPLAN Not., 37(9):4859, 2002. ISSN 0362-1340.