Behavioural types to make object-oriented programs go right

Mario Bravetti* Adrian Francalanza† Hans Hüttel‡ António Ravara§

12 September 2016

Stateful objects typically have a non-uniform behaviour, as the availability of their methods depend on their internal state. For instance, in an object that implements file access the method to read a file should not be invoked before caling the method to open that file. Similarly, in an iterator object, calls to the next method should be preceeded by calls to the hasNext method.

Behavioural types are particularly well-suited to object-oriented programming, as they make it possible to statically guarantee that method calls happen when an object has an internal state that is safe for the method's execution. Following the *typestates* approach [5], one may declare for each possible state of the object the set of methods that can be safely executed in that state.

Several languages, like Bica [3], Mool [1, 2], or Mungo [4], associate with a class a dynamic description of objects' behaviour declaring the admissible sequences of method calls. These descriptions, herein called *usages*, can be used to ensure at compile time that, in a given program, all the sequences of method calls to each object follow the order declared by its usage. To ensure usages to be followed, objects are linear to prevent interferences unexpectedly changing their state.

However, the typing systems referred to above have two shortcomings. First, type checking is typically inefficient, as a method's body is checked each time that method appears in a usage. Second, said typing systems limit themselves to just verifying that method calls follow the usage, and do not necessarily prevent the typed program from "going wrong" (*e.g.*, getting stuck or producing a null pointer exception).

Our work addresses these weaknesses. We attain a stronger type-safety result to the sequential subset of Mool (the simplest of the three languages referred), by including de-referencing a null reference in the definition of errors and by including in type-checking a form of null pointer analysis. Thus, type-safety in our setting means no run-time errors and complete execution of objects' usages. We attain more efficient type-checking by analysing methods' bodies only once. Instead of checking the code following the usage, we introduce *client usages*, behavioural descriptions of how a methods' code changes the state of objects in fields (and variables/parameters), type the method bodies following that information and check the consistency of the usages independently.

Client usages have actually three advantages: (1) they improve the efficiency of type-checking; (2) they facilitate null-pointer analysis for shared objects; and (3) they can, not only be inferred from the code, but also be used to produce pre- and post-conditions to methods that then allow to infer usages (as done by Vasconcelos and Ravara [6]).

We are developing this work in stages. First we define a type system only with usages and prove type-safety (our enhanced version). Subsequently, we extend the type system with client usages and get a more efficient type-checking. Afterwards we infer the client usages from the code, and finally we infer pre- and post-conditions from client usages. Our aim is to provide an approach that takes a program in a Java-like language and automatically infers class usages that describe safe orders of method calls, but also type-checks (client) code against usages (either inferred or user-defined) so as to guarantee that the whole program does not go wrong.

*Department of Computer Science and Engineering, University of Bologna, Italy/ Focus, INRIA, France
†Department of Computer Science, University of Malta, Malta
‡Department of Computer Science, Aalborg University, Denmark
§Department of Computer Science, FCT, New University of Lisbon, Portugal

References

[1] Joana Campos and Vasco Thudichum Vasconcelos. Channels as objects in concurrent object-oriented programming. In *Proceedings of the Third Workshop on Programming Language Approaches to Concurrency and communication-cEntric Software*, volume 69 of *EPTCS*, pages 12–28, 2011.

[2] Joana Correia Campos. Linear and shared objects in concurrent programming. Master's thesis, University of Lisbon, 2010.

[3] Simon J. Gay, Nils Gesbert, António Ravara, and Vasco Thudichum Vasconcelos. Modular session types for objects. *Logical Methods in Computer Science*, 11(4), 2015.

[4] Dimitrios Kouzapas, Ornela Dardha, Roly Perera, and Simon J.Gay. Typechecking protocols with Mungo and StMungo. In *Proceedings of the 18th International Symposium on Principles and Practice of Declarative Programming*. ACM, 2016.

[5] R E Strom and S Yemini. Typestate: A programming language concept for enhancing software reliability. *IEEE Trans. Softw. Eng.*, 12(1):157–171, January 1986.

[6] Cláudio Vasconcelos and António Ravara. From object-oriented code with assertions to behavioural types. In *Proceedings of the Portuguese Simposium in Informatics*. INFORUM, 2016. URL: http://usinfer.sourceforge.net/articles/inforum2016.pdf.