Alias Control is Crucial for Modular Verification of Object-Oriented Programs

Peter Müller and Arnd Poetzsch-Heffter
Fernuniversität Hagen, Germany
[Peter.Mueller, Arnd.Poetzsch-Heffter]@fernuni-hagen.de

Position. One of the most important motivations for alias control is to support program verification. In object-oriented settings, *modular* verification techniques are required to prove that an OO-module behaves according to its specification in all contexts it is (re-)used. In this position paper, we argue that alias protection is crucial for modular verification of OO-programs. If objects can be shared in an arbitrary way, the absence of certain kinds of side-effects cannot be guaranteed by state-of-the-art verification techniques.
Overview. Based on the formal specification and verification technique presented in [MPH99], we describe situations in which unwanted sharing of objects leads to side-effects that cannot be detected by modular verification. We sketch a type system that allows one to prevent such situations by alias control.

Specification Technique. To handle data abstraction and encapsulation, we use so-called abstract attributes to formalize the abstract values of object structures. In implementations, an abstract attribute A is represented by a set of concrete attributes C_i. We say that A *depends on* C_i. Following [Lei95], we require these dependencies to be declared explicitly.

Frame properties of a method m are specified by a modifies-clause describing the set of attributes that might be modified by m. To handle encapsulation and subtyping, the permission to modify an abstract attribute A includes the right to modify all attributes A depends on.

Modular Verification of Frame Properties. The context of a module M consists of M and all modules imported by M. Modular verification means to guarantee that properties of a module M that have been proven in the context of M carry over to all contexts in which M is used.

Modular verification is particularly difficult for frame properties since one has to ensure that a method m does not affect attributes that are not covered by its modifies-clause, even if these attributes are declared outside m's context. The following example illustrates the most prominent problem: Two classes C_1 and C_2 declared in different contexts contain attributes A_1 and A_2 which depend on an attribute A of an imported class. If two instances of A_1 and A_2 depend on the same instance of A, a method m of C_1 with the permission to modify A_1 might also modify A_2 via the shared instance of A. This side-effect cannot be detected during the verification of m since A_2 does not belong to the context of C_1.

Alias Control with Universes. To prevent undetectable side-effects, objects of imported types must not be shared among objects of types declared in different contexts. To achieve that, we equip each module with a so-called *universe*. Universes are partitions of the object store that belong exclusively to the objects of the types declared in the corresponding module (so-called owner objects). Objects inside a universe U can only be directly referenced by other objects of U or by U's owner objects.

This behavior is enforced by an extended type system: Objects in different universes are regarded to be of different types. I.e., types consist of a type identifier and a universe. A type T of universe U is only accessible in U and U's owner module. I.e., outside U, it cannot occur as parameter or return type of public methods which guarantees that references are never passed beyond U's owner objects. Therefore, universes control both static and dynamic aliases. Data can be exchanged between different universes by a clone operation which performs a deep copy of an object structure.

Alias control on the level of modules is sufficient for modular verification, since more fine-grained sharing properties can be guaranteed by verification (cf. [MPH99] for a discussion). The effort of proving such properties can be significantly reduced by refining universes to allow for alias control on the object level (cf. [CPN98] and [Alm97] for such type systems).

References

[Alm97] P. S. Almeida. Balloon types: Controlling sharing of state in data types. In M. Akşit and S. Matsuoka, editors, *ECOOP '97: Object-Oriented Programming*, volume 1241 of *Lecture Notes in Computer Science*, pages 32–59. Springer-Verlag, 1997.

[CPN98] D. G. Clarke, J. M. Potter, and J. Noble. Ownership types for flexible alias protection. In *Proceedings of Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)*, volume 33(10) of *ACM SIGPLAN Notices*, October 1998.

[Lei95] K. R. M. Leino. *Toward Reliable Modular Programs.* PhD thesis, California Institute of Technology, 1995.

[MPH99] P. Müller and A. Poetzsch-Heffter. Modular specification and verification techniques for object-oriented software components. In G. Leavens and M. Sitaraman, editors, *Foundations of Component-Based Systems.* Cambridge University Press, 1999. (to appear).