Poster Presentation: Abstract Interpretation-based Mobile Code Certification

Elvira Albert[1], Germán Puebla[2], and Manuel Hermenegildo[2,3]

[1] SIP, Complutense University of Madrid, elvira@sip.ucm.es
[2] Fac. de Informática, Technical U. of Madrid, {german,herme}@fi.upm.es
[3] Depts. of Comp. Sci. and El. and Comp. Eng., U. of New Mexico, herme@unm.edu

Current approaches to mobile code safety—inspired by the technique of *Proof-Carrying Code* (PCC) [4]—associate safety information (in the form of a *certificate*) to programs. The certificate (or *proof*) is created by the code supplier at compile time, and packaged along with the untrusted code. The consumer who receives the code+certificate package can then run a *checker* which, by a straightforward inspection of the code and the certificate, is able to verify the validity of the certificate and thus compliance with the safety policy. The main practical difficulty of PCC techniques is in generating safety certificates which at the same time: i) allow expressing interesting safety properties, ii) can be generated automatically and, iii) are easy and efficient to check.

We propose an automatic approach to PCC which makes use of *abstract interpretation* [2] techniques for dealing with the above issues. While our approach is general, we develop it for concreteness in the context of (Constraint) Logic Programming, (C)LP, because this paradigm offers a good number of advantages, especially the maturity and sophistication of the analysis tools available. *Assertions* are used to define the safety policy. Such assertions are syntactic objects which allow expressing "abstract"—i.e. symbolic—properties over different *abstract domains*. The first step in our method then involves automatically inferring a set of *safety assertions* (corresponding to the analysis results), using abstract interpretation, and taking as a starting input the program, the predefined assertions available for library predicates, and any (optional) assertions provided by the user for user-defined predicates. The safety policy consists in guaranteeing that the safety assertions hold for the given program in the context of the desired abstract domain. This is automatically provided by the inference process and its correctness ensured by the proved correctness of the process.

The *certification* process—i.e., the generation of a safety certificate by the code supplier which is as small as possible—is in turn based on the idea that only a particular *subset* of the analysis results computed by abstract interpretation-based fixpoint algorithms needs to be used to play the role of certificate for

attesting program safety. In our implementation, the high-level assertion language of [5] is used and the certificate is automatically generated from the results computed by the *goal dependent* fixpoint abstract interpretation-based analyzer of [3]. These analysis results are represented by means of two data structures in the output: the *answer table* and the *arc dependency table*. We show that a particular subset of the analysis results—namely the answer table—is sufficient for mobile code certification. A verification condition generator computes from the assertions and the answer table a *verification condition* in order to attest compliance of the program with respect to the safety policy. Intuitively, the verification condition is a conjunction of boolean expressions whose validity ensures the consistency of a set of assertions. The automatic *validator* attempts to check its validity. When the verification condition is indeed checked, then the answer table is considered a valid certificate.

In order to retain the safety guarantees, the consumer, after receiving the program together with the certificate from the supplier, can trust neither the code nor the certificate. Thus, in the *validation* process, the consumer not only checks the validity of the answer table received but it also (re-)generates a trustworthy verification condition, as it is done by the supplier. The crucial observation in our approach is that the *validation* process performed by the code consumer is similar to the above certification process but replacing the fixpoint analyzer by an *analysis checker* which *does not need to compute a fixpoint*. It simply *checks* the analysis, using an algorithm which is a very simplified *one-pass* analyzer. Intuitively, since the certification process already provides the fixpoint result as certificate, an additional analysis pass over it cannot change the result. Thus, as long as the answer table is valid, a single cycle over the code validates the certificate.

We believe that our proposal can bring the expressiveness and automation which is inherent to abstract interpretation-based techniques to the area of mobile code safety. In particular, the expressiveness of existing abstract domains will be useful to define a wider range of safety properties. Furthermore, in the case of (C)LP the approach inherits the inference power and automation of the abstract interpretation engines developed for this paradigm. A complete description of the method (and related techniques) can be found in [1].

References

1. E. Albert, G. Puebla, and M. Hermenegildo. An Abstract Interpretation-based Approach to Mobile Code Safety. TR CLIP8/2003.0, T. U. of Madrid, Nov. 2003.
2. P. Cousot and R. Cousot. Abstract Interpretation: a Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. POPL'77, pages 238–252, 1977.
3. M. Hermenegildo, G. Puebla, K. Marriott, and P. Stuckey. Incremental Analysis of Constraint Logic Programs. *ACM TOPLAS*, 22(2):187–223, March 2000.
4. G. Necula. Proof-Carrying Code. *POPL'97*, pages 106–119. ACM Press, 1997.
5. G. Puebla, F. Bueno, and M. Hermenegildo. An Assertion Language for CLP. In *Analysis and Visualization Tools for Constraint Programming*, pages 23–61. Springer LNCS 1870, 2000.