Lissom, a Source Level Proof Carrying Code Platform

João Gomes, Daniel Martins, Simão Melo de Sousa
Universidade da Beira Interior

Jorge Sousa Pinto
Universidade do Minho

Traditional PCC architectures center their certificate generation mechanisms on the output of the compilation. Along the lines of recent projects, we believe that there are strong benefits in moving the certificate generation to the source code level. Because there exist good tools for source code verification and for formal verification in general, it is a feature of the Lissom platform that existing tools are used as much as possible at key points of its infrastructure.

Our vision of PCC is based on the following two underlying principles:

- Source level PCC is the way. It is our belief that the realistic formal verification of mobile code should be performed at source level. Programmers may be unaware of the target architecture details, and in general algorithmic constructions are expressed at source level.
- Reuse as much as possible. There exist plenty of powerful tools for the formal verification of source code. Such tools already have experienced user communities, and have reached an appreciable level of maturity and flexibility that make them natural choices in the context of a source level PCC architecture.

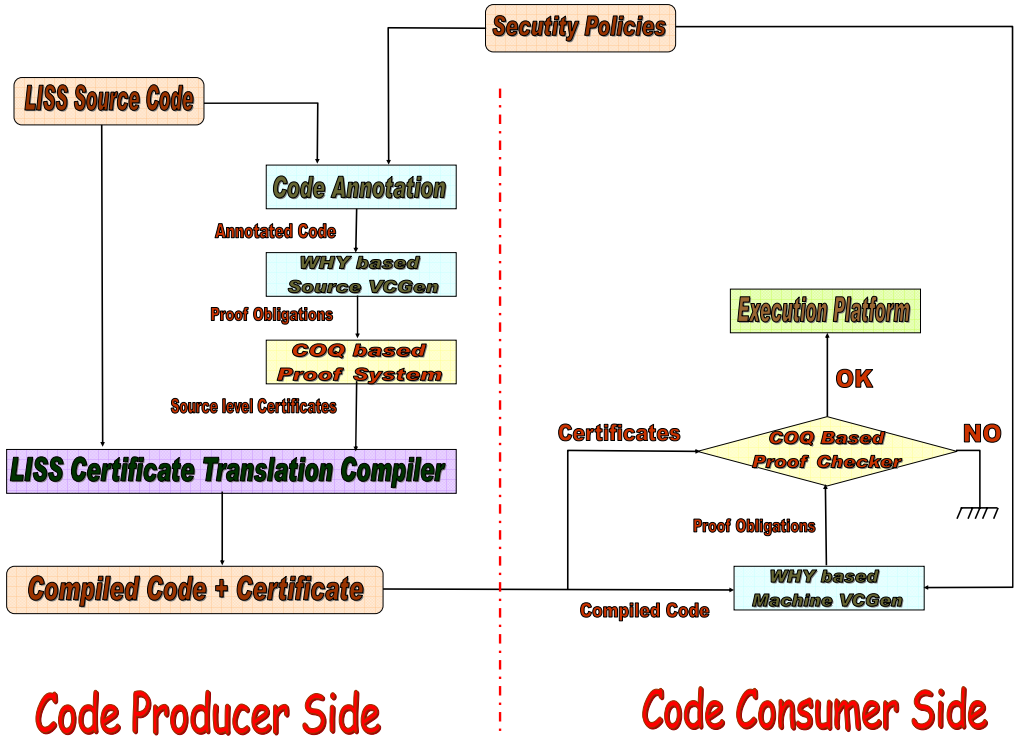
Started as a challenge for final year Computing students, Lissom was thought as a mean to prove to a skeptic community, and in particular to students, that formal verification tools can be put to practice in a realistic environment, and be used to solve complex and concrete problems. The attractiveness of the problems that PCC addresses has already brought students to show interest in this project. Aside this context, our main goal with Lissom is to get experienced with PCC and to put it into practice.

After the present prototyping phase, the platform must have proved to be adequate for mobile code security. The relevance and conceptual solidity of previous works on formal verification (e.g. [2]) on which this is based lead us to believe in the success of the enterprise.

Our road-map is the following, where the points highlighted as in progress are the modules which are currently in active development.

1. To design an annotation language for Liss (in progress);
2. To extend the why tool to contemplate this annotation language, allowing to use why as a generator of proof obligations for source code (in progress);
3. To design a proof system for the Liss language, integrated in Coq (starting phase);
4. To extend the Liss compiler with the capability to translate certificates (starting phase);
5. To design a proof system for the virtual machine and its language, integrated in Coq (in progress);
6. To design a proof obligation generator for compiled code (in progress).

We finish with a few examples of the many interesting problems that will be raised in this work, along with the classical challenges that every PCC platform must address. A first problem is the automation of the COQ proof process and its impact on the consumer effort; the tension between expressiveness and automation is a well-known problem that must be carefully studied. It also remains to see to what extent the techniques presented in [7] allow for conciseness of COQ certificates. The language Liss and the virtual machine used are non-trivial, but are still relatively simple when compared to platforms such as JAVA or .NET. The capacity of the platform to scale up to such platforms must thus be evaluated. Finally, it will be important to apply our choices in an appropriate case study that starts from the security policy specification to the certificate verification (proof of concept)



Proposed Architecture

The Source Language and the Compiler. LISS (Language for Integers Sets and Sequences) is a non-trivial toy language that also features a realistic type system (with e.g. sets, vectors a la Java, etc.) and high-level constructs. LISS is intended here as a suitable test-bed before aiming at an industrial-level language.

This language must be extended in order to provide an annotation system for the source code. In a source level PCC architecture, the compiler has to compile source code but also proofs into their machine level counterpart. A very interesting challenge is to transform a source level structural proof (proofs heavily rely on the structure of the analyzed program) into a proof that is still structurally close to the machine level code. We follow here the contributions of [1]).

The Virtual Machine. Lissom uses a sequential, stack-based virtual machine, which despite its simplicity has the capacity to support real languages (such as C or Java), and is, together with LISS, a suitable test-bed. The machine is an adaptation of Filliâtre's original virtual machine[4], used for teaching several Courses at our universities (e.g. Compiler Construction and Formal Methods).

The Proof System and the Proof Checker. As far as the Trusted Computing Base (TCB) is concerned, it is important for it to be as small and solid as possible; we believe that an adequate choice of proof system may help attaining such a goal. Also, it is important to be able to express high level polices as well as lower level ones.

These requirements have led us to consider using the COQ proof system and its higher-order specification language and underlying proof mechanisms. This system, based on the calculus of inductive constructions, has been used with success for the formal verification of critical and large-scale systems. As far as source code is concerned, integration with COQ is guaranteed by the existence of a numberof tools suited for code annotation and proof (e.g. [6]). Lissom will feature a source code verification system based on Why and the COQ system. Thus we intend to use COQ proof objects as certificates.

The Verification Condition Generator. This will be obtained using Filliatre's WHY tool [5], which is capable of producing proof obligations for various systems, including COQ. We are presently working on a WHY module for the language LISS (equivalent to Caduceus for C[6]) and for the input language of the virtual machine. The annotation language used for this will be an adaptation of JML[3] specialized for the security policy specification.

Future Work and Directions

- Short and mid term objectives: Lissom is at an early development stage we must conclude the implementation and test the Lissom platform.

  This will include: the LISS Certificate Translation Compiler, narrowing and automatizing the use of the proof system and the VCGen to a specific class of security policies.
- Long term objective:

  Deployment of a source level PCC architecture for constrained embedded systems.

  Due to high security concerns:
  - embedded code execution schemes are very constrained. For instance: no dynamic facilities, heavy resources (memory, etc...) confinement, no firmware/software updates, In general one purpose induces one dedicated architecture.
  - Embedded software development models usually rely on software components outsourcing and contractors have difficulties to perform safe integration of the outsourced software components

  In this context, Safety mechanisms based on verifiable evidences (certificates) potentially will:
  - ease the distributed development model (e.g. safe outsourcing: outsourced code comes with evidences that it does not contain flaws identified by the contractor);
  - give rise to a new embedded software paradigm. Safety certificates allow new execution scheme where, for instance, (a) a program can provide static evidences that it will not use unsafe operations or resources ;(b) two applications can safely cohabit in a embedded system.

References

[1] G. Barthe, B. Grégoire, C. Kunz, and T. Rezk. Certificate translation for optimizing compilers. In Proceedings of the 13th International Static Analysis Symposium. LNCS, Springer-Verlag, 2006.

[2] G. Barthe, P. Courtieu, G. Dufay, and S. Melo de Sousa, Tool-Assisted Specification and Verification of Typed Low-Level Languages. Journal of Automated Reasoning, Jan 2006.

[3] Lilian Burdy, Yoonsik Cheon, David Cok, Michael Ernst, Joe Kiniry, Gary T. Leavens, K. Rustan M. Leino, and Erik Poll, An overview of JML tools and applications, International Journal on Software Tools for Technology Transfer (STTT) 7 (2005), no. 3, 212-232.

[4] J.-C. Filliâtre, Resources for the compilation course - the virtual machine, from the author's webpage.

[5] J.-C. Filliâtre, Why: a multi-language multi-prover verification tool, Research Report 1366, LRI, Université Paris Sud, March 2003.

[6] J.-C. Filliâtre and Claude Marché, Multi-Prover Verification of C Programs, Sixth International Conference on Formal Engineering Methods (ICFEM) (Seattle), Lecture Notes in Computer Science, vol. 3308, Springer-Verlag, November 2004, pp. 15-29.

[7] G. C. Necula and P.Lee, Efficient representation and validation of logical proofs, Proceedings of the 13th Annual Symposium on Logic in Computer Science. IEEE Computer Society Press, 1998, pp. 93-104.