Mobile Resource Guarantees (MRG)

EU-funded research project Edinburgh – Munich (2002 – 2004)

L. Beringer, K. MacKenzie, D. Sannella, D. Aspinall, S. Gilmore, I. Stark

Abstract: *Mobile* computation is characterised by the mobility of target devices (handheld PC's, phones, wearable computers) and that of code. The design of mobile applications is constrained by several factors: devices move through a network of dynamic topology and offer limited computational resources (memory, operating time, communication bandwidth). Code is dynamically downloaded from a code producer and replaces existent code modules, but may be unsafe or corrupted during the transfer. The MRG project [1] is developing the technology for endowing mobile code with independently verifiable certificates describing its resource requirements.

Security threats of mobile code

- authentication (certification of authorship)
- data integrity and access permissions to local data
- consumption of resources: memory, execution time, energy

Java Virtual Machine's approach

- program analysis at code consumer side checks basic properties of code
- code is executed in a controlled environment (sandbox, protection domain) which dynamically enforces access restrictions
- class-file loader includes security manager

Disadvantages:

- dynamic monitoring is costly
- maliciousness of code is detected late (at execution time), when resources have already been consumed or data has been overwritten
- program analysis performed at code consumer's side

Alternative: proof-carrying code

Idea: transmit program *and a proof* certifying its properties [4]

1. formalise the property of interest (security policy, memory usage,...) and encode it in an appropriate logic
2. code producer creates proof and bundles it with program
3. code consumer checks program-proof pair before execution and rejects inconsistent pairs

Advantages:

- the computationally expensive proof *construction* is shifted from the mobile device to the code producer (compiler), and needs to be performed only once. The consumer only performs proof *checking*
- neither the code producer nor the network need to be trusted
- complements authorship certification / encryption etc.

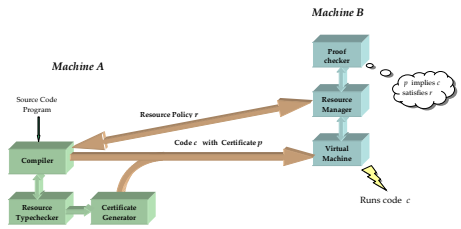
Ongoing PCC research

- to develop logics appropriate for interesting properties
- to reduce size of proofs and to automate proof search
- to develop certifying compilers for high-level languages: compilers that emit assembly programs which include the proof certificates
- foundational type-theoretical questions

So far, most research concentrates on authentication and data integrity.

MRG: PCC & resource consumption

MRG aims to certify resource consumption during execution on a virtual machine (JVM, .NET). These certificates are derived from high-level programs via *resource types* – abstract characterisations of resource consumption such as Hofmann's ◇-types for heap space [2]. Typed compilation and transparent cost models relate certificates and type systems.



Code transfer and certification

More precisely, we are tackling the following topics.

1. definition of realistic models of resource consumption (heap usage, number of system calls,...) for a JVM-like low-level language
2. development of a byte-code logic for resource consumption: Hoare-style proof rules relating low-level programs to their costs
3. design of a high-level language with a transparent mapping to byte-code
4. development of reasoning principles and type systems for the high-level language (◇-types,...)
5. implementation of a certifying compiler (proof generator)
6. extensions: objects, concurrency, negotiation of certification policy

Grail, Camelot and Galahad

Abstract reasoning about VM code is facilitated by a novel low-level language *Grail*. Grail presents a *functional* representation to the high-level compiler while also allowing programs to be expanded into *imperative* byte code in a straight forward way. Grail's main characteristics are:

- the expansion may target various VM's (JVM, .NET, JavaCard)
- program transmission uses existing VM technology, hence integration of legacy software is supported and class-file verification may be applied
- reasoning and certification may be formulated at the functional level, hence they may abstract from inessential VM-features
- transmitted programs may be converted back into functional form, hence consumer-side proof checking may apply the more abstract reasoning techniques that are available for functional languages
- VM code satisfies Leroy's conditions [3] for efficient on-card verification

While the method structure of VM code is retained, basic blocks are represented as tail-recursive functions. A novel calling convention ensures that local VM variables correspond *bijectively* to functional variables.

Camelot is a functional high-level programming language similar to core-ML and OCAML, with higher-order functions, arrays, polymorphism, algebraic datatypes and (restricted) pattern matching.

Galahad extends Camelot by programmer-friendly idioms and a module system, facilitating structured programming-in-the-large.

Conversion tools for the expansion of Grail code and its reversal have been completed, as have compilers from Camelot and Galahad to Grail. The latter tools will be extended by mechanisms for manufacturing certificates in the future, based on Camelot-level resource type systems, Grail-level resource logics and VM-level cost models.

References

[1] MRG web-page: http://www.lfcs.ed.ac.uk/mrg/.

[2] Martin Hofmann. A type system for bounded space and functional in-place update. *Nordic Journal of Computing*, 7(4), Winter 2000.

[3] Xavier Leroy. Bytecode verification for Java smart cards. *Software Practice & Experience*, 32:319–340, 2002.

[4] George C. Necula. Proof-carrying code. In *Proceedings of the 24th ACM Symposium on Principles of Programming Languages (POPL'97)*, pages 106–119, January 1997.

Division of informatics