Certification of Embedded Systems

F. Blanqui and J.-P. Jouannaud (INRIA)

LIAMA & Tsinghua University, Beijing, China

FORMES project

http://formes.asia

06/10/09, Osaka, Japan

Outline

Software certification

Some certification tools

Software certification

goal: make sure that an executable behaves as expected

problem: it relies on the correctness of many tools

- executable generator: parser, type-checker, compiler, pretty-printer, assembler, linker
- verification tools: static analyzer, model checker, verification condition generator, automated theorem prover, proof checker

and on the correctness of hardware...

Software certification

goal: make sure that an executable behaves as expected

problem: it relies on the correctness of many tools

- executable generator: parser, type-checker, compiler, pretty-printer, assembler, linker
- verification tools: static analyzer, model checker, verification condition generator, automated theorem prover, proof checker

and on the correctness of hardware...

proposal: try to break this vicious circle by using a proof assistant

Break the vicious circle

- logicians have shown that proofs can be carried out by using a small number of deduction rules
- proof assistants like Coq, Isabelle, HOL, Agda, etc. are based on this idea:
  - a kernel checks the correctness of proofs
    it is small enough to be checked by hand (and formalized)
  - a proof development environment provides tools for building proofs
    it does not matter too much that these tools have bugs:
    their results are checked by the kernel

proof-script → [Proof assistant] → proof → [Kernel] → yes/no

Proof assistants

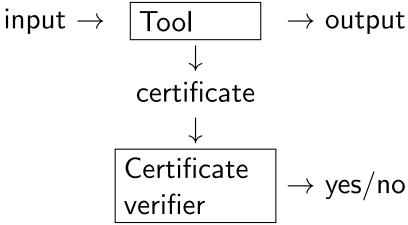
generally provide:

- a (non-imperative) programming language for defining functions that can generally be extracted and compiled as usual programs
- a specification language for defining predicates
- a script language or interface for building proofs
- libraries with usual functions, predicates and tactics
- a module system
- ...

How to certify a software ?

possible approaches:

- prove once and for all that the software has no bug
  - hard/tedious for complex/big software
  - static analysers and automated theorem provers can help a lot
  - must be redone every time the code is changed
- make the software provide, each time it is run, some certificate that can be used to check the correctness of its output
  - does not depend on the way the tool is implemented
  - require to develop and prove a certificate verifier
  - not adapted for compilers, simulators, etc.
  - well adapted for decision procedures (SMT, termination, etc.)



examples:

- certification of termination proofs (CoLoR)
- certification of CoqMT, an extension of Coq with SMT (Pierre-Yves Strub in FORMES)

Outline

Software certification

Some certification tools

CompCert: http://compcert.inria.fr

- Coq-certified optimized compiler from CLight to ARM/PowerPC
- CLight is a large subset of C with a completely defined semantics

not certified yet:

- parser and type-checker for C
- pretty-printer for ARM/PowerPC
- assembler and linker

see picture. . .

Why: `http://why.lri.fr`

(partially certified) verification condition generator:

- takes as input a C/Java/ML program with, for each function, annotations describing pre- and post-conditions and, for each loop, an invariant and a measure for termination (variant)
- for each instruction of the program, generates all the conditions that must be satisfied for the instruction to execute correctly and the pre- and post-conditions of the program to be satisfied
- try to check these conditions automatically by sending them to various automated theorem provers
- send them to some proof assistant otherwise

see picture. . .

Example of annotated C code

```
int index(int t[], int n, int v)
  /*@ array_length(t) = n */ {
  int i = 0;
  while (i < n)
    /*@ invariant 0 <= i variant n - i */
    if (t[i] == v) break;
    i++;
  }
  return i;
}
/*@ 0 <= result < n -> t[result] = v */
```

Frama-C: `http://frama-c.cea.fr`

(uncertified) suite of tools dedicated to the analysis of C code

- gathers several static analysis techniques (as plugins) in a single collaborative framework: slicer, dependency analysis, etc.
- allows static analyzers to build upon the results already computed by other analyzers in the framework
- verification of functional specifications based on Why

see picture. . .

Application to PLC programs

- formalization in Coq of the various languages used for programming PLCs together with their semantics (SFC, IL, ST, FBD, LD, Basic, C)
- development of a Coq-certified compiler for these languages
- extension of Why/Frama-C with time/temporal logic annotations

Conclusion

- many tools already exist for (partially) certifying programs
- some of these tools need to be themselves certified
- kernel-based proof assistants can help break this vicious circle
- instead of certifying a program, one can use certificates

Thank you!