

PROOF-CARRYING-CODE

APPLYING FORMAL METHODS IN A DISTRIBUTED WORLD

Hans-Wolfgang Loidl

LFE Theoretische Informatik, Institut für Informatik,
Ludwig-Maximilians Universität, München

July 1, 2005

① SUMMARY AND CURRENT TRENDS

② OTHER RELEVANT PROJECTS

ConCert/Hemlock
More Projects

③ HANDS-ON SESSION

The Touchstone Demo
The MRG infrastructure

④ THE CCURED CERTIFYING COMPILER

SUMMARY

PCC is a powerful, general mechanism for providing safety guarantees for mobile code.

It provides these guarantees without resorting to a trust relationship.

It uses techniques from the areas of type-systems, program verification and logics.

It is a very active research area at the moment.

CURRENT TRENDS

Using formal methods to check specific program properties.

- Program logics as the basic language for doing these checks attract renewed interest in PCC.
- A lot of work on program logics for low-level languages.
- Immediate applications for smart cards and embedded systems.

FUTURE DIRECTIONS

Embedded Systems as a domain for formal methods.

- Some of these systems have strong security requirements.
- Formal methods are used to check these requirements.
- Model checking is a very active area for automatically checking properties.

LINKS TO OTHER AREAS

Checking program properties is closely related to inferring quantitative information.

- **Static analyses** deal with extracting quantitative information (e.g. resource consumption)
- A lot of research has gone into making these techniques efficient.
- **Model checking** can deal with a larger class of problems (e.g. specifying safety conditions in a system)
- Just recently these have become efficient enough to be used for main stream programming.

CONCERT/HEMLOCK

An infrastructure for using **PCC on Grids**, developed by the group of Peter Lee at CMU [1].

Based on earlier joint work with Necula on the concept of PCC.

Goal: use the computational power of a Grid architecture to exploit parallelism and gain speedups.

GRID ARCHITECTURES

What is a Grid?

Grid is a type of parallel and distributed system that enables the sharing, selection, and aggregation of geographically distributed "autonomous" resources dynamically at runtime depending on their availability, capability, performance, cost, and users' quality-of-service requirements.

KEY FEATURES OF GRID ARCHITECTURES

- **Transparent, decentralised use of resources.**

The application doesn't care where something is executed, as long as the machine has the capabilities need to run the job.

- Usage of open, standardised communication protocols
- Non-trivial quality of service

But, the donator of a node in a Grid network may want to limit the amount of compute cycles, memory etc, provided to the Grid infrastructure.

⇒ use PCC to check the resource usage!

HEMLOCK

Hemlock is a **certifying compiler for Grid/ML** with threads running on a Grid architecture.

Grid/ML is a functional subset of ML plus reference types.

As part of the ConCert system, compiled down to typed-assembly language.

Thread primitives implement simple **fork-and-join parallelism**:

```
type 'a task
val spawn : (unit -> 'a) -> 'a task
val sync  : 'a task vec -> 'a vec
```

SUMMARY

- So far only basic ideas are worked out, no full infrastructure, yet.
- Demo of Hemlock/ConCert exists: fork-and-join parallelism, but no certification
- Current focus on foundational work on logics and fault tolerance



For a flashy presentation of Hemlock see

<http://www-2.cs.cmu.edu/concert/talks/Murphy2003Hemlock/hemlock.swf>

OTHER RELEVANT PROJECTS

- Tinman: <http://www.cs.utexas.edu/users/wjyu/tinman/>
- High-assurance JVM:
<http://www.kestrel.edu/HTML/projects/java/>
- Verificard: <http://www.verificard.org/>
- Hume: <http://www.hume-lang.org/>

FURTHER READING

-  Bor-Yuh Evan Chang, Karl Crary, Margaret DeLap, Robert Harper, Jason Liszka, Tom Murphy VII, and Frank Pfenning, *Trustless Grid Computing in ConCert*, in GRID 2002 — Third International Workshop on Grid Computing, November 2002. LNCS 2536.
-  Aloysius K. Mok and Weijiang Yu, *TINMAN: A Resource Bound Security Checking System for Mobile Code* in European Symposium on Research in Computer Security, Zurich, Switzerland, October 14–16, LNCS 2502, Springer-Verlag, 2002.

HANDS-ON SESSION

The final part of the lecture is a hands-on session for some of the systems discussed earlier.

Goal: Get a deeper understanding about the systems by actually working with them (continue off-line!).

Check of the state-of-the-art in terms of implemented systems.

SETUP FOR RUNNING THE CODE

Execute this script:

```
/zvol/formal/hwloidl/demosrc
```

or

```
export PATH=/zvol/formal/hwloidl/gnu/bin:\
           /zvol/formal/hwloidl/j2sdk1.4.1_07/bin:$PATH
export LD_LIBRARY_PATH=/zvol/formal/hwloidl/gnu/lib:\
                    /zvol/formal/hwloidl/j2sdk1.4.1_07/lib:$LD_LIBRA
```

Index of demos available here:

<http://www.tcs.ifi.lmu.de/~hwloidl/tmp/demos.html>

TOUCHSTONE OVERVIEW

A PCC infrastructure for proving type safety of assembler code.

Based on the original work by Necula on PCC.

Demonstrates the most complete PCC infrastructure available.

TOUCHSTONE OVERVIEW

A PCC infrastructure for proving type safety of assembler code.

Based on the original work by Necula on PCC.

Demonstrates the most complete PCC infrastructure available.

Main features;

- High-level language: Java
- Encoding of proofs: LF Terms or Oracle strings
- Hand written VCG
- Twelf as proof checker

THE TOUCHSTONE DEMO

Start from this url:

<http://raw.cs.berkeley.edu/Ginseng/Images/pccdemo.html>

The demo itself gives a good discussion of what's going on.

Click on components of the picture to get explanations.

THE MRG INFRASTRUCTURE

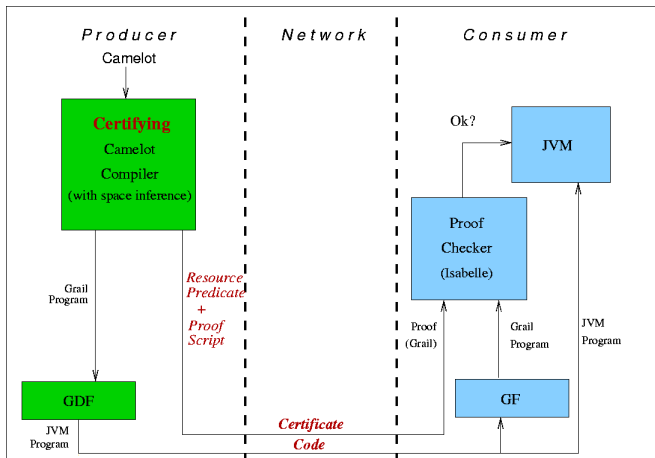
This part is structured as follows (all on-line):

- Background on inferring heap consumption for Camelot
- Basics on operational semantics and program logic for Grail
- Running the full demo on some examples

Goal:

Provide an infrastructure for **independent and automatic verification** of resource certificates (for space), sent with mobile code.

A PROOF-CARRYING-CODE INFRASTRUCTURE FOR MRG



COMPONENTS IN THE INFRASTRUCTURE

On the **producer** side we have:

Camelot compiler:

- Input: Camelot program
- Output: JAR file with executables as class files and a certificate
- Automatic inference of space consumption
- Automatic generation of a certificate (mainly loop invariants)

COMPONENTS IN THE INFRASTRUCTURE

On the **producer** side we have:

Camelot compiler:

- Input: Camelot program
- Output: JAR file with executables as class files and a certificate
- Automatic inference of space consumption
- Automatic generation of a certificate (mainly loop invariants)

Note: It's also possible to use Isabelle/HOL and “hack your proofs”.

COMPONENTS IN THE INFRASTRUCTURE

On the **consumer** side we have:

- Isabelle running in batch mode as a proof checker
- JVM for executing the code if it's safe

EXAMPLE: INSERTION SORT (CAMELOT)

Camelot program:

```
let ins a l = match l with
  Nil -> Cons(a,Nil)
  | Cons(x,t)@_ -> if a < x then Cons(a,Cons(x,t))
                  else Cons(x, ins a t)

let sort l = match l with Nil -> Nil
                       | Cons(a,t)@_ -> ins a (sort t)
```


EXAMPLE: INSERTION SORT (TYPES)

Inferred space consumption:

```

ins      : 1, int -> iList[0|int,#,0] -> iList[0|int,#,0], 0;
sort     : 0, iList[0|int,#,0] -> iList[0|int,#,0], 0;
start    : 0, list_1[string,#,1|0] -> unit, 0;
  
```

Certificate as Isabelle proof script for this specification:

$$\Gamma \triangleright \text{snd} (\text{methhtable } \text{InsSortM } \text{ins}) : \\ \llbracket \{a, l\}, 1, [a \mapsto \mathbf{I}, l \mapsto \mathbf{L}(0)] \blacktriangleright \mathbf{L}(0), 0 \rrbracket$$

$$\Gamma \triangleright \text{snd} (\text{methhtable } \text{InsSortM } \text{sort}) : \\ \llbracket \{l\}, 0, [l \mapsto \mathbf{L}(0)] \blacktriangleright \mathbf{L}(0), 0 \rrbracket$$

$$\Gamma \triangleright \text{snd} (\text{methhtable } \text{InsSortM } \text{start}) : \\ \llbracket \{l\}, 0, [l \mapsto \mathbf{L}(1)] \blacktriangleright (), 0 \rrbracket$$

EXECUTING CERTIFIED CODE ON SMALL DEVICES

Using a “delegated PCC” model, the certificate check can be done by a powerful, trusted host, before code is executed on a small device eg. PDA.

Goal:

Demonstrate usability of the infrastructure by **executing** the code on a small, memory-constrained device (PDA).

CHARACTERISTICS OF SMALL DEVICES

Characteristics of Small Devices:

- Uses MIDP standard for small devices is MIDP with a reduced Java class hierarchy.
- All applications must interact with the user via a GUI.
- For resource constrained devices: 16 or 32-bit at >16MHz CPUs; >192kb for JVM; >32k for heap.
- Verification is partially performed off-device.
- Example: Sun's KVM

CCURED: THE IDEA

A system for checking **pointer-safety** of C programs, developed by the group of George Necula at Berkeley.

System achieved **pointer safety** statically, where possible, and minimise required run-time checks.

CCURED: THE IDEA

A system for checking **pointer-safety** of C programs, developed by the group of George Necula at Berkeley.

System achieved **pointer safety** statically, where possible, and minimise required run-time checks.

The type system distinguishes between 3 kinds of pointers:

- **Safe pointers**: no arithmetic or casts
- **Sequence pointers**: arithmetic but **no casts**
- **Dynamic pointers**: **casts**, all bets are off!

THE CCURED CERTIFYING COMPILER

We will go through examples from the CCured tutorial:

<http://manju.cs.berkeley.edu/ccured/tutorial.html>

EXAMPLE: ARRAY-SUM

This computes the sum over an array of integers:

```
int arrSum(int * p, int len) {
    int sum = 0, i;
    for(; len >= 0; len --, p ++ ) {
        sum += *p;
    }
    return sum;
}
```

EXAMPLE: ARRAY-SUM (INFERENCE)

Ccured infers `p` as an FSEQ pointer: a sequence pointer with only positive pointer arithmetic:

```
int addAll(int *$$_9$p    &$_11$, int len ) &$_10$;
```

```
*** Node 9.
```

```
Kind FSEQ : Positive Arithmetic : syntax on node 9 at ccuredcode.tmp/ex  
Flags:
```

- * Positive Arithmetic : syntax on node 9 at ccuredcode.tmp/ex22.c:4
- * Cannot be SAFE : Positive Arithmetic

EXAMPLE: ARRAY-SUM (CCURED CODE)

This is the “ccured” code for this function:

```
int addAll_f(fseqp_int p, int len)
{ int sum;
  int i;
  int __retres;

  {
  sum = 0;
  i = 0;
  while (i < len) {
    CHECK_FSEQARITH2SAFE((void *)p._p, p._ms._e, (void *) (p._p + i), sizeof(int), 1, 0);
    sum = sum + *(p._p + i);
    i = i + 1;
  }
  __retres = sum;
  return (__retres);
}
}
```