HOL-Boogie —
An Interactive Prover-Backend for the
Verifying C Compiler

Burkhart Wolff

With help by: Wolfram Schulte, Rustan Leino,
Mike Barnett, Jan Smuts, Herman Venter, Michal Moskal, ...
Context (1)

- The VeriSoft Xt Project
  - started 2007, 24 mio € budget, 3 years, ca. 100 men-year work.
  - several larger verification sub-projects
    - Avionics, Car-Electronics
    - Pike-OS Kernel (a real-time OS)
    - Microsofts Hyper-V (a virtualization OS)
Context (2)

- Microsofts Hyper-V (a virtualization OS)
What is the Hyper-V Hypervisor?

- an operating system
  - manages processes ("guests","partitions"),
  - memory,
  - events and IPC's
- (but no real devices, that is handled by the root partition)
What is special with Hyper-V?

in contrast to a standard OS, which emulates linear ("logical") memory for its processes, it emulates physical memory

i.e. an MMU

for its guests (using X86 - V Chipset)
Context (5)

• The Hyper-V Verification Project

  • Motivation:
    Tremendously complex, difficult to test.

  • Relatively small:
    50000 line of code in ANSI C (X86 - V) and Assembler

  • There have been formal models of processors and virtual machines for a while
    (INTEL's X86 (Forte), AMD's X86 (ACL 2) JVM (Isabelle/HOL), VAMP (Isabelle/HOL), ...)

The Hyper-V Verification Project

Target: Correctness Proof. Prove that an emulated X86 processor (running one one core of X86-V) behaves like a standard X86 processor (modulo time).
The Hyper-V Verification Project

obviously, a lot of new verification technology is needed.
Motivation (1)

- **Automated Theorem Proving (ATP)** has found its "Killer-Application": Static Program-Analysis
  - SAL-Annotations in MS Vista and MS Word!
  - Boogie: Data-Invariant Checking

- **Interactive Theorem Proving (ITP)**: No Killer-App in sight (people still hate to see proofs ...), but
  - Verifications of complex algorithms, or even mathematically challenging theorems, is S-o-t-A.
  - Lots of Technology exists to get calculi right and to get provers safely work together.
Motivation(2)

• Boogie:

... is a program-oriented specification method aiming at "deeper" algorithmic verification (as, e.g., SAL).

... offers an extremely attractive "Analyze&Fix" cycle.

Still, failures of proof attempts can be difficult to understand: Is it the prover? The program? The spec?
Plan of the Talk

- Scenario I: HOL-Boogie as Interactive Prover of Boogie VC's, with an “Analyse&Fix” based on ITP. (%70)

- Challenges and Answers for ITP in a static program analysis application (%20)

- Scenario II: HOL-Boogie in C Verification (%10)
Scenario I

- Workflow:
Scenario I

• The Problem: Dijkstra's Shortest Path Algorithm

Data:

type Vertex;
const Graph: [Vertex, Vertex] int;
const AllVertices: [Vertex] bool;
axiom (forall x: Vertex :: AllVertices[x]);
axiom (forall x: Vertex, y: Vertex:: x != y ==> 0 < Graph[x,y]);
axiom (forall x: Vertex, y: Vertex:: x == y ==> Graph[x,y] == 0);
const Infinity: int;
axiom 0 < Infinity;
var Shortest: [Vertex, Vertex] int;
Scenario I

- The Problem: Dijkstra's Shortest Path Algorithm

Toplevel-Specification:

```plaintext
procedure Dijkstra();
modifies Shortest
ensures (forall x:Vertex::AllVertices[x]==>Shortest[x,x] == 0);
ensures (forall x: Vertex, y: Vertex, z: Vertex ::
  AllVertices[x] && AllVertices[y] && AllVertices[z] ==> 
  Shortest[x,z] <= Shortest[x,y] + Graph[y,z]);
ensures (forall x: Vertex, z: Vertex ::
  AllVertices[x] && AllVertices[z] ==> 
  Shortest[x,z] <= Graph[x,z]);
...
```
Scenario I

- The Problem: Dijkstra's Shortest Path Algorithm

Toplevel-Specification:

```plaintext
procedure Dijkstra();
modifies Shortest
ensures (forall x:Vertex::AllVertices[x] ==> Shortest[x,x] == 0);
ensures (forall x: Vertex, y: Vertex, z: Vertex ::
    AllVertices[x] && AllVertices[y] && AllVertices[z] ==> 
    Shortest[x,z] <= Shortest[x,y] + Graph[y,z]);
ensures (forall x: Vertex, z: Vertex ::
    AllVertices[x] && AllVertices[z] ==> 
    Shortest[x,z] <= Graph[x,z]);

...
Scenario I

- The Problem: Dijkstra's Shortest Path Algorithm Implementation:

```plaintext
havoc Shortest;
assume (forall x: Vertex, y: Vertex : AllVertices[x] && AllVertices[y] ==> x==y ==> Shortest[x,y] ==0);
assume (forall x: Vertex, y: Vertex : AllVertices[x] && AllVertices[y] ==> x != y ==> Shortest[x,y] == Infinity);
SourceNotVisited := AllVertices;
```
Scenario I

- The Problem: Dijkstra's Shortest Path Algorithm Implementation:

```plaintext
assert (forall x: Vertex ::
    SourceNotVisited[x] ==> AllVertices[x]);

assert (forall x: Vertex ::
    AllVertices[x] ==> Shortest[x,x] == 0);

assert (forall x: Vertex, y: Vertex, z: Vertex ::
    AllVertices[x] && AllVertices[y] .... ==> 
    SourceNotVisited[x] ||
    Shortest[x,z] <=
    Shortest[x,y] + Graph[y,z]);
...
```
Scenario I

- The Problem: Dijkstra's Shortest Path Algorithm Implementation:
Scenario I

- Verification with HOL-Boogie (Attempt I)

Generating .b2i-file:

/cygdrive/c/boogie/Binaries/Boogie /prover:isabelle Dijkstra.bpl

and get it under /cygdrive/c/Dijkstra.1.b2i.

And then start Isabelle under ProofGeneral:

DEMO
Scenario I

- Verification with HOL-Boogie (Attempt I)

Attempt 1 stuck at:

\[
[| \ldots ; \\
   \ldots ; \\
   |] \Rightarrow 0 \leq \text{Shortest}@3(x,y) + \text{Graph}(y,z)
\]

The Problem occurs when establishing the entry-condition from DoneInner to Loophead.

- Solution: Strengthen the Invariants to \(0 \leq \text{Shortest}(x,y)\)
Scenario I

- The Problem: Dijkstra's Shortest Path Algorithm Implementation:

entry

Loophead

Loopbody

InnerLoopHead

InnerLoopBody

DoneInner

Done

assert (forall x: Vertex :: SourceNotVisited[x] ==> AllVertices[x]);

assert (forall x: Vertex :: AllVertices[x] ==> Shortest[x,x] == 0);

assert (forall x: Vertex, y: Vertex, z: Vertex :: AllVertices[x] && AllVertices[y] .... ==> SourceNotVisited[x] || Shortest[x,z] <= Shortest[x,y] + Graph[y,z]);

...
Scenario I

- The Problem: Dijkstra's Shortest Path Algorithm Implementation:

```
entry

Loophead

Loopbody

InnerLoopHead

InnerLoopBody

DoneInner

Done

assert (forall x: Vertex ::
    SourceNotVisited[x] ==> AllVertices[x]);
assert (forall x: Vertex, y: Vertex::
    AllVertices[x] && AllVertices[y] ==> 0 <= Shortest[x,y]);
assert (forall x: Vertex :: AllVertices[x] =>
    Shortest[x,x] == 0);
assert (forall x: Vertex, y: Vertex, z: Vertex ::
    AllVertices[x] && AllVertices[y] .... ==> 
    SourceNotVisited[x] ||
    Shortest[x,z] <=
    Shortest[x,y] + Graph[y,z]);
...```
Scenario I

- Results I:
  - Attempt II (with strengthened Invariant) succeeds
  - Proof takes 5 min. in interactive mode.
  - Proof deliberately low-level; anyone with medium expertise in ITP should be able to do this!
  - Z3 does still not find the proof.
  - Proof development took 1,5 working days
  - An alternative “classic” ATP verification by improvement of DijkstraN was abandoned by [Leino&al] after 1,5 days.
Challenges: ITP for PA

- Techniques specific to ITP in Program Analysis
  - Tactics taking the structure of wp-generated formulas into account
  - Positional and Structural Labelling Techniques
  - Integration of SMT solvers
  - Integration of techniques to make prover instrumentations transparent through different provers ...
Scenario I: Tactics

- Observation of wp-generated formulas:
  Why? ... The “skeleton” is a deterministic proof.

- Algorithm induced skeleton
- Interfacing interactive proofs
- Automated Proofs
Scenario I: Labelling

• Positional labels “this assertion is from line 55 …”
  
  block_at Line_25_Col_3 True
  assert_at Line_55_Col_4 (...)

  (Technique described in Leino, Millstein, and Saxe: Generating error traces from verificationcondition counterexamples. SCP, 55-1-3, 2005)

• Structural Labels “this assertion holds at entry of loop A”

  ...

  (not much used so far, but better for repeated Analyse&Fix.)
Scenario I: Instrumentation

- Any prover has a life of its own. Rules must be massaged and instrumented to tell an automated prover HOW a ruleset has to be used.

- Attributation of Signature elements:
  
  axiom \{prover:{isabelle:builtin”add_commute”}\} ( ... )

- Prover instrumentation:
  
  axiom \{prover:{isabelle:intro!}\} ( ... )
  axiom {:ignore "bvDefSem"} (forall x:int ::
  
  \{ $sign\_extend.1.32($\_int.to.bv32(x)[1:0]) \}
  
  -$\_bv64.to.int(1bv64) \leq x \&\& x < -$\_bv64.to.int(1bv64)
  
  ==> $sign\_extend.1.32($\_int.to.bv32(x)[1:0]) ==
  
  $\_int.to.bv32(x)\};
Scenario II: Verifying C Programs

- Workflow: One further redirection step. And a complex memory/machine model.

```
C compiler → .bpl → Boogie → .b2i → HOL-Boogie

axiomatization of the "c virtual machine" (cvm)
```

VCC

```
HOL-Boogie → .thy → Z3

Z3
```

HOL-Boogie
Scenario II

- Example:

```c
longint i = 0;

void incr()
  requires i < maxint
  ensures i <= maxint
{
    (i++);
}
```
Scenario II

- Example:

```plaintext
const i_ptr :: ptr

procedure incr();
modifies mem
requires ($clt.u8($ld.u2(mem, i_ptr), maxint))
ensures ($cle.u8($ld.u2(mem, i_ptr), maxint) &&
    modifiesOnly(mkSet(i_ptr)))

implementation incr(){
assumes($clt.u8($ld.u2(mem, i_ptr), maxint))

mem := $st.i8(mem, $add.i8($ld.i8(mem, i_ptr),1))

assert($cle.u8($ld.u2(mem, i_ptr), maxint) &&
    modifiesOnly(mkSet(i_ptr)))
}
```
Scenario II

- VCC or Spec# require:
  
  considerably large,

  axiomatic background theories on

  - memory models
  - machine operations (X86 VT)
  - specialized instrumentations on
    the prover side for each memory/machine model (actually, there is VCC1 and VCC2)
Scenario II

• Task:

  • HOL-Boogie as a generator of a consistent prelude, the “C-Virtual Machine”.

• Motivation: Providing a comprehensive Axiomatization of logics and its environment (State, Bitvectors, CVM)

  • for checking the consistency

  • for prover integration
Conclusion

- ITP techniques can provide an effective means to algorithmic verification in Boogie although the “Analyze&Fix”-cycle is substantially slower.

- ITP techniques can provide explicit, comprehensive and consistent preludes for complex logical contexts. This helps to increase confidence into the approach.

- ITP's are still unavoidable in “real” Code-Analysis if algorithms, recursive data-structures, or deep arithmetic reasoning is involved.

⇒ Lots of Potential !!!
We proudly announce ...

- Journal Paper on the nitty-gritty details:


  see: http://www.lri.fr/~wolff/publications_year.html
Scenario II

- Let's do it: (it will take some time !!!)

DEMO