The Hyper-V Verification Experiment

verifying 10s of 1000s of lines of C code

optimized concurrent system-level industrial

Stephan Tobies
European Microsoft Innovation Center

with Michał Moskal, Ernie Cohen, and many more
why verification?

- the **only** way we have to establish functional correctness for most interesting software
- reduce functional test cost
- take QA out of the TCB
- cope with missing components/environment
- provide more accurate documentation
- reduce system brittleness
The reason to verify C is because it is the most common language used... To say that a little better you will be providing more overall verification to the universe of current software by verifying C than by doing so for any other language.

That's the good part. Here's the bad part. By trying to verify C, you are starting something that you will likely never finish. You will almost certainly experience a sense of utter dismay at the number of flaws you find,...

The third thing to realize is that you will probably learn absolutely nothing about the underlying issues of proving properties of programs through your effort - unless of course you are not already an expert. So, let's see...
if so, why not?

- “it requires logicians”
  - users have to be experts in **proof**, not just **truth**
- “it’s boring”
  - few people enjoy dragging an interactive proof assistant through messy proof states
- “it’s too much work”
  - L4.verified: **annotation/code ratio of 20/1** - for sequential code
- “it only works for toy programs”
  - real code is **written for speed**, not for verification
- “if programmers can’t do it, it won’t scale”
that’s why
our vision

- programmers should be able to verify the functional correctness of their code
- cost should be comparable to the cost of thorough functional testing
- verification should be program-centric, not proof-centric
unsigned binary_search(int val, int *arr, unsigned len)
_(requires \thread_local_array(arr, len))
_(requires \forall UINT i,j; i < j && j < len ==> arr[i] <= arr[j])
_(ensures \result != UINT_MAX ==> arr[\result] == val)
_(ensures \result == UINT_MAX ==> \forall UINT i; i < len ==> arr[i] != val)
{
    unsigned low, high, mid;
    low = 0; high = len;
    while (low < high)
        _(invariant high <= len)
        _(invariant \forall unsigned i; i < low ==> arr[i] < val)
        _(invariant \forall unsigned i; high <= i && i < len ==> arr[i] >= val)
        {
            mid = low + (high - low) / 2;
            if (arr[mid] < val) low = mid + 1;
            else high = mid;
        }
    if (low < len && arr[low] == val) return low;
    else return UINT_MAX;
}
what’s a hypervisor?

Microsoft Hyper-V™ Server 2008

x64 HW

virtual x64 HW

virtual x64 HW

virtual x64 HW
why Hyper-V?

- industrial software with tractable code size (ca. 100K lines of C, 5K lines of x64 asm)
- self-contained
- correctness is important
- testing and debugging are difficult
- prototypical for OS verification
- complex implementation, concise spec
correctness

- A guest OS cannot distinguish between whether it is executed:
  - directly on the hardware
  - or through the HV

Diagram:
- Direct execution
  - OS
  - Hardware
- Virtualization
  - OS
  - Virtual HW
  - Hypervisor
  - Hardware
correctness

- correct virtualization wrt. a realistic x64 model given as C code
  - caches, TLBs, store buffers
  - weak memory model
- memory safety, concurrency safety
- on the level of full-blown, unmodified C
  - verify implementation, not algorithms
VCC

- an industrial strength verifier for concurrent C
- user input: assertions, ghost code
- feedback: error messages, counterexamples
- fully automatic via Z3 SMT solver
- verification is function- and thread-modular
- integrated into VS 2008 and VS 2010
- annotation/code ratio of Hyper-V: 1:1 to 3:1
verisoft xt

- EMIC, DFKI, Uni Saarbrücken, Windows
- co-funded by German Ministry of Education and Research (BMBF)
- numerous “stand-alone” results

verisoftxt.de
results

- annotated and verified ~30KLOC in 1 year
  - synch objects: locks, rundownhs, etc.
  - data structures, objects protocols
  - Shadow Address Space (SAS) implementation
  - procedural abstraction for cross-thread calls
- formal x64 model in C (incl. caches)
- practical store buffer reduction for x64
- and much more ...
where are we?

- verification should be **program-centric**, not proof-centric

- **programmers** should be able to verify the functional correctness of their code

- **cost** should be comparable to the cost of thorough functional testing
get it!

vcc.codeplex.com

rise4fun.com/vcc