Deep Specifications and Certified Abstraction Layers

Ronghui Gu, Jérémie Koenig, Tahina Ramananandro, Zhong Shao, Newman Wu, Shu-Chun Weng, Haozhong Zhang\(^1\), Yu Guo\(^1\)

Yale University, University of Science and Technology of China

January 17, 2015

http://flint.cs.yale.edu
Motivation

How to build reliable & secure **system software stacks**?
Motivation

Android architecture & system stack

Motivation

Visible software components of the Linux desktop stack

From http://en.wikipedia.org/wiki/Linux
Motivation

Software stack for HPC clusters
From http://www.hpcwire.com/2014/02/24/comprehensive-flexible-software-stack-hpc-clusters/

<table>
<thead>
<tr>
<th>Performance Monitoring</th>
<th>HPCC</th>
<th>Perfctr</th>
<th>IOR</th>
<th>PAPI/IPM</th>
<th>netperf</th>
</tr>
</thead>
<tbody>
<tr>
<td>Development Tools</td>
<td>Cray® Compiler Environment (CCE)</td>
<td>Intel® Cluster Studio</td>
<td>PGI (PGI CDK)</td>
<td>GNU</td>
<td></td>
</tr>
<tr>
<td>Application Libraries</td>
<td>Cray® LibSci, LibSci_ACC</td>
<td>MVAPICH2</td>
<td>OpenMPI</td>
<td>Intel® MPI (Cluster Studio)</td>
<td></td>
</tr>
</tbody>
</table>

**Essential Software and Management Tools Needed to Build a Powerful, Flexible, and Highly Available Supercomputer.**

<table>
<thead>
<tr>
<th>Resource Management / Job Scheduling</th>
<th>SLURM</th>
<th>Grid Engine</th>
<th>MOAB</th>
<th>Altair PBS Pro</th>
<th>IBM Platform LSF</th>
<th>Torque/Maui</th>
</tr>
</thead>
<tbody>
<tr>
<td>File System</td>
<td>NFS</td>
<td>Local FS (ext3, ext4, XFS)</td>
<td>PanFS</td>
<td>Lustre</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Provisioning</td>
<td>Cray® Advanced Cluster Engine (ACE) management software</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cluster Monitoring</td>
<td>Cray ACE (ISC8 and OpenIPMI)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Remote Power Mgmt</td>
<td>Cray ACE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Remote Console Mgmt</td>
<td>Cray ACE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Operating System**

- Linux (Red Hat, CentOS, SUSE)
Motivation

Cisco’s FAN (Field-Area-Network) protocol layering

From https://solutionpartner.cisco.com/web/cegd/overview
Motivation

Apollo Mobile Communication Stack
http://www.layer2connections.com/apollo_clients.html

Web Application Development Stack
From http://www.brightware.co.uk/Technology.aspx
Motivation (cont’d)

• Common themes: all system stacks are built based on abstraction, modularity, and layering

• Abstraction layers are ubiquitous!

Such use of abstraction, modularity, and layering is “the key factor that drove the computer industry toward today’s explosive levels of innovation and growth because complex products can be built from smaller subsystems that can be designed independently yet function together as a whole.”

Do We Understand Abstraction?

**In the PL community:**
(abstraction in the small)

- Mostly formal but tailored within a single programming language (ADT, objects, existential types)
- Specification only describes type or simple pre- & post condition
- Hide concrete data representation (we get the nice repr. independence property)
- Well-formed *typing* or *Hoare-style judgment* between the impl. & the spec.

**In the System world:**
(abstraction in the large)

- Mostly informal & language-neutral (APIs, sys call libraries)
- Specification describes full functionality (but in English)
- Implementation is a black box (*in theory*); an *abstraction layer* hides all things below
- *The “implements” relation* between the impl. & the spec.
Do We Understand Abstraction?

In the PL community:  
(abstraction in the small)

- Mostly formal but tailored within a single programming language (ADT, objects, existential types)
- Specification only describes type or simple pre- & post condition
- Hide concrete data representation (we get the nice repr. independence property)
- Well-formed typing or Hoare-style judgment between the impl. & the spec.

In the System world:  
(abstraction in the large)

- Mostly informal & language-neutral (APIs, sys call libraries)

Something magical going on ...  
What is it?

between the impl. & the spec.
Problems

• What is an *abstraction layer*?

• How to formally *specify* an abstraction layer?

• How to *program*, *verify*, and *compile* each layer?

• How to *compose* abstraction layers?

• How to apply *certified abstraction layers* to build *reliable* and *secure* system software?
Our Contributions

• We introduce **deep specification** and present a language-based formalization of **certified abstraction layer**

• We developed new languages & tools in Coq
  – A formal layer calculus for composing certified layers
  – ClightX for writing certified layers in a C-like language
  – LAsm for writing certified layers in assembly
  – CompCertX that compiles ClightX layers into LAsm layers

• We built multiple **certified OS kernels** in Coq
  – mCertiKOS-hyper consists of **37 layers**, took less than **one-person-year** to develop, and can boot **Linux** as a guest
What is an Abstraction Layer?

overlay $L_2$

C or Asm module implementation

underlay $L_1$
**Example: Page Tables**

**concrete C types**

```c
struct PMap {
    char * page_dir[1024];
    uint page_table[1024][1024];
};
```

**abstract Coq spec**

```coq
Inductive PTPerm: Type :=
    | PTP
    | PTU
    | PTK.

Inductive PTEInfo:=
    | PTEValid (v : Z) (p : PTPerm)
    | PTEUnPresent.

Definition PMap := ZMap.t PTEInfo.
```
Example: Page Tables

abstract state

\[ \text{PMap} := \text{ZMap.t PTEInfo} \]
\[ (* \ \text{vaddr} \rightarrow (\text{paddr, perm}) *) \]

Invariants: kernel page table is a direct map; user parts are isolated

abstract primitives

(Coq functions)

Function page_table_init = …
Function page_table_insert =…
Function page_table_rmv = …
Function page_table_read = …

memory

\[ \text{char * page_dir[1024]}; \]
\[ \text{uint page_table[1024][1024]}; \]

C functions

\[ \text{int page_table_init}(); \{ \ldots \} \]
\[ \text{int page_table_insert}(); \{ \ldots \} \]
\[ \text{int page_table_rmv}(); \{ \ldots \} \]
\[ \text{int page_table_read}(); \{ \ldots \} \]
Formalizing Abstraction Layers

What is a **certified** abstraction layer \((L_1, M, L_2)\) ?

- **spec** \(L_2\) with abstract state \(abs\)
- **module** \(M\) with concrete state: \(mem\)
- **overlay** interface
- **underlay** interface

Called abstract primitives in \(L_1\)

Simulation (implements) relation \(R(abs, mem)\)

Recorded as the **well-formed layer** judgment

\[ L_1 \vdash_R M : L_2 \]
The Simulation Relation

\[ L_1 \vdash_R M : L_2 \quad \Rightarrow \quad L_2 \leq_R \sem{M} L_1 \]

**Forward Simulation:**
- Whenever \( L_2(f) \) takes \( \text{abs1} \) to \( \text{abs2} \) in one step, and \( R(\text{abs1}, \text{mem1}) \) holds,
- then there exists \( \text{mem2} \) such that \( \sem{M}(L_1)(f) \) takes \( \text{mem1} \) to \( \text{mem2} \) in zero or more steps, and \( R(\text{abs2}, \text{mem2}) \) also holds.
Reversing the Simulation Relation

$L_1 \vdash_R M : L_2 \quad \Rightarrow \quad L_2 \leq_R \llbracket M \rrbracket L_1$

If $\llbracket M \rrbracket (L_1)$ is deterministic relative to external events (\textit{a la} CompCert),

$\llbracket M \rrbracket L_1 \leq_R L_2$

$\llbracket M \rrbracket L_1 \sim_R L_2$

$\llbracket M \rrbracket (L_1)$ and $L_2$ are bisimilar!

$L_2$ captures everything about running $M$ over $L_1$
Deep Specification

\[ M \odo L_1 \sim^R L_2 \]

\([M] (L_1) \] and \( L_2 \) are bisimilar!

\( L_2 \) captures everything about running \( M \) over \( L_1 \)

Making it “contextual” using the whole-program semantics \( \bullet \)

\( L_2 \) is a deep specification of \( M \) over \( L_1 \)

if under any valid program context \( P \) of \( L_2 \),

\[ [P \oplus M] (L_1) \] and \( [P] (L_2) \) are observationally equivalent
**Why Deep Spec is Really Cool?**

Deep spec $L$ captures all we need to know about a layer $M$

- No need to ever look at $M$ again!
- Any property about $M$ can be proved using $L$ alone.

**Impl. Independence**: any two implementations of the same deep spec are *contextually equivalent*
Is Deep Spec Too Tight?

• Not really! It still *abstracts* away:
  – the *efficient* concrete data repr & impl. algorithms & strategies

• It can still be *nondeterministic*:
  – External nondeterminism (e.g., I/O or scheduler events) modeled as a set of deterministic traces relative to external events (*a la* CompCert)
  – Internal nondeterminism (e.g., sqrt, rand, resource-limit) is also OK, but any two implementations must still be *observationally equivalent*

• It *adds* new logical info to make it *easier-to-reason-about*:
  – auxiliary *abstract states* to define the full functionality & invariants
  – accurate *precondition* under which each primitive is valid
Problem w. Shallow Specs

C or Asm module

C & Asm Module Implementation

shallow spec A

C & Asm Modules w. Shallow Spec A

shallow spec B

Want to prove another spec B?

Need to revisit & reverify all the code!
Shallow vs. Deep Specifications

- C or Asm module
- shallow spec
- deep spec

C & Asm Module Implementation

C & Asm Modules w. Shallow Specs

C & Asm Modules w. Deep Specs
How to Make Deep Spec Work?

No languages/tools today support deep spec & certified layered programming

*Challenges:*

• Implementation done in C or assembly or …
• Specification done in richer logic (e.g., Coq)
• Need to mix *both* and also simulation proofs
• Need to compile C layers into assembly layers
• Need to compose different layers
Our Contributions

• We introduce deep specification and present a language-based formalization of certified abstraction layer

• We developed new languages & tools in Coq
  – A formal layer calculus for composing certified layers
  – ClightX for writing certified layers in a C-like language
  – LAsm for writing certified layers in assembly
  – CompCertX that compiles ClightX layers into LAsm layers

• We built multiple certified OS kernels in Coq
  – mCertiKOS-hyper consists of 37 layers, took less than one-person-year to develop, and can boot Linux as a guest
What We Have Done

**Coq**
- Layer Spec \( L \)

**Clight**
- CompCert
- Asm

**Extended Asm Language**
- LAsm

- **ClightX[\( L \)]**
  - compositional compiler

- **LayerLib calculus**

**Layered prog. in LAsm**
- Layered prog. in ClightX

- Layered prog. in ClightX

**Layered prog. in ClightX**

**Link everything together**

**Parametrize it w. abstract states & primitives in \( L \)**
LayerLib: Vertical Composition

\[
\begin{align*}
L_1 \vdash_R M : L_2 & \quad L_2 \vdash_S N : L_3 \\
L_1 \vdash_{R \circ S} M \oplus N : L_3
\end{align*}
\]
Example: Thread Queues

High Abs-State

1::0::2::nil

tcbp(0) tcbp(1) tcbp(2)

Ready Ready Ready

Low Abs-State

tcbp(0)

Ready 1 2

Ready 0

Ready

Concrete Memory


Ready Ready Ready
**C Implementation**

typedef enum {
    TD_READY, TD_RUN,
    TD_SLEEP, TD_DEAD
} td_state;

struct tcb {
    td_state tds;
    struct tcb *prev, *next;
};

struct tdq {
    struct tcb *head, *tail;
};

struct tcb tcbp[64];
struct tdq tdqp[64];

struct tcb * dequeue (struct tdq *q) {
    ....... }

**Low Layer Spec in Coq**

Inductive td_state :=
| TD_READY | TD_RUN
| TD_SLEEP | TD DEAD.

Inductive tcb :=
| TCBV (tds : td_state)
 (prev next : Z)

Inductive tdq :=
| TDQV (head tail : Z)

Record abs :=
    tcbp : ZMap.t tcb;
    tdqp : ZMap.t tdq }

Function dequeue (d : abs) (i : Z) :=
Some(set_tdq d i q', h)
| nil => None
end

**High Layer Spec in Coq**

Inductive td_state :=
| TD READY | TD RUN
| TD_SLEEP | TD DEAD.

Definition tcb := td_state.

Definition tdq := List Z.

Record abs' :=
    tcbp : ZMap.t tcb;
    tdqp : ZMap.t tdq }

Function dequeue (d : abs') (i : Z) :=
match (d.tdqp i) with
| h :: q' =>
    Some(set_tdq d i q', h)
| nil => None
end
Example: Dequeue

High Abs-State

Low Abs-State

tcbp(0)

tcbp(1)

tcbp(2)

Concrete Memory

1 :: 0 :: 2 :: nil

Ready

Ready

Ready

Ready

Ready

Ready

Ready

Ready

Ready

Ready

Ready

Ready

Ready
Conflicting Abstract States?

client program $P$

$L_1$ with $\text{abs}_1$

$R_1$

module $M_1$

interface $L$ with abstract state: $\text{abs}$

$R$

module $M$ with concrete state: $\text{mem}$

$L_2$ with $\text{abs}_2$

$R_2$

$?$
LayerLib: Horizontal Composition

- $L_1$ and $L_2$ must have the same abstract state
- both layers must follow the same simulation relation $R$

\[
\frac{L \xleftarrow{\mathcal{R}} M : L_1}{L \xleftarrow{\mathcal{R}} M \oplus N : L_1 \oplus L_2} \quad \text{HCOMP}
\]
Programming & Compiling Layers

ClightX

\[ L \vdash_R M_c : L_1 \]

\[ L_1 \leq_R \llbracket M_c \rrbracket_{\text{ClightX}} (L) \]

CompCertX correctness theorem (where \textit{minj} is a special kind of memory injection)

\[ \llbracket M_c \rrbracket_{\text{ClightX}} (L) \leq_{\text{minj}} \llbracket \text{CompCertX}[L](M_c) \rrbracket_{\text{LAsm}} (L) \]

\[ L_1 \leq_{R \cdot \text{minj}} \llbracket \text{CompCertX}[L](M_c) \rrbracket_{\text{LAsm}} (L) \]

\( R \) must absorb such memory injection: \( R \cdot \text{minj} = R \) then we have:

\[ L_1 \leq_R \llbracket \text{CompCertX}[L](M_c) \rrbracket_{\text{LAsm}} (L) \]

Let \( M_a = \text{CompCertX}[L](M_c) \) then \( L \vdash_R M_a : L_1 \)

LAsm
Our Contributions

• We introduce **deep specification** and present a language-based formalization of **certified abstraction layer**

• We developed new languages & tools in Coq
  – A **formal layer calculus** for composing certified layers
  – **ClightX** for writing certified layers in a C-like language
  – **LAsm** for writing certified layers in assembly
  – **CompCertX** that compiles **ClightX** layers into **LAsm** layers

• We built multiple **certified OS kernels** in Coq
  – **mCertiKOS-hyper** consists of **37 layers**, took less than **one-person-year** to develop, and can boot **Linux** as a guest
Case Study: mCertiKOS

Single-core version of CertiKOS (developed under DARPA CRASH & HACMS programs), 3 kloc, can boot Linux

Aggressive use of abstraction over deep specs (37 layers in ClightX & LAsm)
Decomposing mCertiKOS

Based on the abstract machine provided by boot loader
Decomposing mCertiKOS (cont’d)

<table>
<thead>
<tr>
<th>PProc Layer</th>
<th>get/set_uctx</th>
<th>proclin</th>
<th>send/recv/check_chan</th>
<th>proc_create/start/exit</th>
<th>mm/thread_prim</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUCtx Layer</td>
<td>get/set/save textarea_uctx</td>
<td>proclin</td>
<td>send/recv/check_chan</td>
<td>mm/thread_prim</td>
<td></td>
</tr>
<tr>
<td>PIPC Layer</td>
<td>proclin</td>
<td>send/recv/check_chan</td>
<td>mm/thread_prim</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PIPCIntro Layer</td>
<td>schedinit</td>
<td>get/set_chan</td>
<td>thread_prim</td>
<td>mm/prim</td>
<td></td>
</tr>
<tr>
<td>PThread Layer</td>
<td>schedinit</td>
<td>cid_get</td>
<td>thread_sleep/yield/spawn/kill/wakeup</td>
<td>mm/prim</td>
<td></td>
</tr>
<tr>
<td>PSched Layer</td>
<td>schedinit</td>
<td>cid_get</td>
<td>Htcb_set</td>
<td>thread/sched/kill</td>
<td>mm/prim</td>
</tr>
<tr>
<td>PCID Layer</td>
<td>htqinit</td>
<td>cid_set</td>
<td>Htcb_get/set</td>
<td>Hen/def/ rm_queue</td>
<td>kctx_switch/new/free</td>
</tr>
<tr>
<td>PAbQueue Layer</td>
<td>htqinit</td>
<td>Htcb_get/set</td>
<td>Hen/def/ rm_queue</td>
<td>kctx_switch/new/free</td>
<td>mm/prim</td>
</tr>
<tr>
<td>PTDQInitLayer</td>
<td>tdqinit</td>
<td>Ltcb_get/set</td>
<td>Len/def/ rm_queue</td>
<td>kctx_switch/new/free</td>
<td>mm/prim</td>
</tr>
<tr>
<td>PTDQIntro Layer</td>
<td>tdqinit</td>
<td>Ltcb_get/set</td>
<td>Ldtq_get/set</td>
<td>kctx_switch/new/free</td>
<td>mm/prim</td>
</tr>
<tr>
<td>PTCBInitLayer</td>
<td>tcbinit</td>
<td>Ltcb_get/set</td>
<td>kctx_switch/new</td>
<td>mm/prim</td>
<td></td>
</tr>
<tr>
<td>PKCtxNewLayer</td>
<td>pmapinit</td>
<td>Ltcb_get/set/init</td>
<td>kctx_switch/new</td>
<td>PT_free</td>
<td>mm/prim</td>
</tr>
<tr>
<td>PKCtx Layer</td>
<td>pmapinit</td>
<td>kctx_switch</td>
<td>kctx_new</td>
<td>PT_free</td>
<td>mm/prim</td>
</tr>
<tr>
<td>MPTNewLayer</td>
<td>pmapinit</td>
<td>PT_new/free</td>
<td>PT_repl/reel/free</td>
<td>palloc/free</td>
<td>setPT</td>
</tr>
</tbody>
</table>

Thread and Process Management (14 Layers)
Decomposing mCertiKOS (cont’d)

Virtualization Support (9 Layers)

- **User-space**
  - User-space Virtual Machine Manager
  - Virtual Device 1
  - ... Virtual Device N
  - Certified App
  - Uncertified App

- **Trap**
  - Trap Handlers (interrupts, exceptions, system call handlers)

- **Virtualization**
  - AMD SVM Abstraction (primitives for VMCB & NPT)

- **Process & Thread**
  - Process & Thread Management & IPC

- **MM**
  - Memory Management (Physical Memory & Virtual Memory Management)

- **Drivers & Prinit**
  - Preinit
  - PIC Driver
  - Timer Driver
  - IDE Driver
  - SVM Driver

- **HW**
  - CPU
  - Memory
  - PIC (8259)
  - Timer (8254)
  - IDE Controller

- **Certified Kernel**

### VSVM Layer
- mm/proc.abs, npt, hctx, vmst
- vmcbinit
- svm_check/exit/sync/inject/set (16)
- vm_run/exit
- NPT_insr
- mm/proc_prim

### VMCBOp Layer
- mm/proc.abs, npt, hctx, vmst
- vmcbinit
- vmcb_check/clear/inject/set (12)
- NPT_insr
- switch_to/from_guest
- mm/proc_prim

### VSVMIntro Layer
- mm/proc.abs, npt, hctx, vmst
- vmcbinit
- vm_st_read/write
- NPT_insr
- switch_to/from_guest
- mm/proc_prim

### VVMCBInit Layer
- mm/proc.abs, npt, hctx, vmcb
- vmcbinit
- vmcb_read/write
- NPT_insr
- switch_to/from_guest
- mm/proc_prim

### VVMCBIntro Layer
- mm/proc.abs, npt, hctx, vmcb
- nptinit
- vmcb_read/write
- NPT_insr
- switch_to/from_guest
- mm/proc_prim

### VSVMOp Layer
- mm.abs, proc.abs, npt, hctx
- nptinit
- NPT_insr
- switch_to/from_guest
- mm/proc_prim

### VSVMSwitch Layer
- mm.abs, proc.abs, npt, hctx
- nptinit
- NPT_insr
- restore/save_hctx
- mm/proc_prim

### VNPTInit Layer
- mm.abd, proc.abs,npt
- nptinit
- NPT_insr
- mm/proc_prim

### VNPTIntro Layer
- mm.abd, proc.abs,npt
- procinit
- set_NPDE
- mm/proc_prim

### PProc Layer
- mm.abs, proc.abs
- procinit
- proc_create/start/exit
- get/set_utct
- send/rec/exit
- mm/thread_prim
Decomposing mCertikOS (cont’d)

Current Target: Single-Core CertikOS Syscall and Trap Handlers (3 Layers)
Variants of mCertiKOS Kernels
Example: Page Fault Handler

TSysCall
TTtrap
TTtrapArg
PProc
PUCtx
...
PCID
...
PMap
MPTOp
MPTIntro
MAT
MATOp
...
PreInit ikern_set setcr3 pf_get

pagefault_handler
pf_resv
set_err
set_uctx
proc_start
proc_exit
save_uctx
set_PTE
pt_resv
pt_insrt
set_PTE
setpmi
Hallo
Conclusions

• Great success w. today’s **system software** … but why?

• We identify, sharpen, & **formalize** two possible ingredients
  – abstraction over deep specs
  – a compositional layered methodology

• We build new lang. & tools to make **layered programming rigorous & certified** --- this leads to **huge benefits**:
  – simplified design & spec; reduced proof effort; better extensibility

• They also help **verification in the small**
  – hiding implementation details as soon as possible

• Still need better PL and tool support  
  (Coq / ClightX / LAsm)
Thank You!

Interested in working on the CertiKOS project? we are hiring & recruiting at all levels:

postdocs,
research scientists,
PhD students, and visitors
A Subtlety for LAsm

Some functions (e.g., kernel context switch) do not follow the C calling convention and must be programmed in LAsm\([L]\).

\[ L \vdash_R M_a : L_2 \quad \Rightarrow \quad L_2 \leq_R \llbracket M_a \rrbracket_{\text{LAsm}}(L) \]

**Problem:** per-module semantics \(\llbracket M_a \rrbracket_{\text{LAsm}}(L)\) is *NOT deterministic* relative to external events.

\[ \llbracket M_a \rrbracket_{\text{LAsm}}(L) \leq_R L_2 \]

Fortunately, whole-machine semantics \(\llbracket \bullet \rrbracket_{\text{LAsm}}(L)\) is deterministic relative to external events, so it can still be reversed:

\[ \forall P. \llbracket P \oplus M_a \rrbracket_{\text{LAsm}}(L) \sim_R \llbracket P \rrbracket_{\text{LAsm}}(L_2) \]
Hide concrete memory; replace it with Abstract State
Only the *getter* and *setter* primitives can access memory
Layer Pattern 2: AbsFun

Memory does not change
New implementation code does not access memory directly!