Programming a Microkernel Specification in Separation Logic

Paolo G. Giarrusso, Gregory Malecha, David Swasey, Yoichi Hirai

BedRock Systems, Inc.
Formal Verification @ Bedrock


- Operational semantics "at the boundaries" — HW & unverified guests.

TCB:
- C++ compiler correctness
- C++ axiomatic semantics in Iris
- HW models
Formal Verification @ Bedrock

This talk

NOVA API (outside of bare-metal property statement)

Highly concurrent L4-family capability-based microkernel

The future is built on BedRock.
Challenges with kernel specs

Verified host processes

Disciplined NOVA specs — in Iris

Kernel API

NOVA microkernel
Challenges with kernel specs

Kernel API

NOVA microkernel

Verified host processes

Disciplined NOVA specs — in Iris

Unverified, undisciplined processes
Challenges with kernel specs

Undisciplined NOVA specs

Kernel API

NOVA microkernel

Verified host processes

Disciplined NOVA specs — in Iris

Robustness proof

Unverified, undisciplined processes
Challenges with kernel specs

- Undisciplined NOVA specs
- Kernel API
- NOVA microkernel
- Verified host processes

- Robustness proof
- Disciplined NOVA specs — in Iris
- Unverified, undisciplined processes

*directly in Iris*
Undisciplined specs in Iris: Advantages

- Single proof for NOVA (NOVA's pretty complex)
- Small footprint without detours through big footprint and associated overhead
- We lose adequacy for NOVA in isolation; but appropriate for us since NOVA's internal

Subjectively:
- Easy to evolve
- Two specs, but little duplication (undisciplined specs are mostly about error handling and atomicity)
Undisciplined NOVA specs as axiomatic semantics
An undisciplined WP for the NOVA machine

NOVA machine = NOVA + CPU:

Predicates:
- nova.wp : ∀ (ec : ec_nameT), mpred
- ec.reg : ec_nameT → Qp → regsT → mpred

Types:

ec_nameT: an identifier for a "thread" (Execution Context)

regsT: the type of the "register file" (CPU internal state)

regular_machine_step :
  ∀ (old new : regsT), Prop

atomic CPU steps (no assumptions on guest discipline)

HW, caches, memory modeled as external components
An undisciplined WP for the NOVA machine

\[
\text{nova.wp}_\text{step_intro}: \\
|={}^{\top},^\uparrow\text{nova}\_\text{ns}\Rightarrow \triangleright (\exists \text{regs, ec.regs ec 1 regs} \ast \\
\text{if syscall\_trap}\ \text{regs then wp\_hypercall}\ \text{ec}\ \text{regs} \ \text{else} \\
(\forall \text{regs}', [\mid \text{regular\_machine\_step}\ \text{regs}\ \text{regs'} \mid] -{*} \\
\text{ec.regs ec 1 regs'} ={^\uparrow\text{nova}\_\text{ns},^{\top}}={^*} \ \text{wp}\ \text{ec}) \\
\land \ \text{wp\_traps}\ \text{ec}\ \text{regs})
\]
\[\vdash \text{nova.wp}\ \text{ec}.
\]

Elimination rule: syscall for spawning threads
An undisciplined WP for the NOVA machine

wp_hypercall ec regs :=
  match decode_syscall regs with
  | ipc_call => wp_ipc_call ec regs
  | ipc_reply => wp_ipc_reply ec regs
  | ...
  end.
Robustness

Robustness statement:

\[ \text{inv } \text{invName process_resources } * \text{ persistent_process_props } \vdash \text{ nova.wp } \text{ ec} \]

Proof sketch: by Löb induction and case analysis on the step; each obligation must be satisfied via the invariant.

- For memory, for each physically accessible page (via page tables) we need ownership in invariants.
- For syscalls, we must satisfy all syscall preconditions from invariants.
An example syscall: IPC call
The future is built on BedRock.
The future is built on BedRock.
The future is built on BedRock.
EC 1 (NOVA)

ipcall(handle)

lookup(handle) = EC 2
is EC 2 free? No? Pause!
lookup(handle) = EC 2
is EC 2 free? No? Pause!

EC 2

EC 3

ipc_reply()
The future is built on BedRock.

EC 1

ipc_call(handle)

EC 1 (NOVA)

lookup(handle) = EC 2

is EC 2 free? No? Pause!

lookup(handle) = EC 2

is EC 2 free? No? Pause!

lookup(handle) = EC 2

is EC 2 free? Yes, so rendezvous!

EC 2

EC 3

ipc_reply()
Rendezvous in Iris

**Definition** resolve_handle_chan_rendezvous
(caller_ec : ec_nameT) handle Q :=

\[ \text{AU} \leftrightarrow \forall \text{ chan rights q callee_state,} \]
\[ \text{cap_at caller_ec handle q (channel, rights)} \]
\[ \square \text{ channel_ec channel callee_ec} \]
\[ \text{ec.kstate callee_ec callee_state} \rightarrow @ \text{ novaM} \]
\[ \leftrightarrow \exists \text{ result, cap_at handle q (chan, rights)} \]
\[ \text{if insufficient rights then } \| \text{ result = EPERM } \| \]
\[ \text{ec.kstate callee_ec callee_state} \]
\[ \text{else } \| \text{ callee_state = AVAILABLE } \land \text{ result = SUCCESS } \| \]
\[ \text{ec.kstate callee_ec RUNNING),} \]
\[ \text{COMM Q result callee_ec} \rightarrow. \]
Definition \( \text{ipc\_spec\_raw\ caller\_ec\ handle} := \) 
\[
\text{resolve\_handle\_chan\_rendezvous\ caller\_ec\ handle}\ 
(\lambda\ \text{result\ callee\_ec}, \ 
\forall\ \text{src\ dst}, \ 
\text{buf\_addr\ caller\_ec\ src} -\ast\ (*\ \text{Persistent}\ *) \ 
\text{buf\_addr\ callee\_ec\ dst} -\ast \ 
\text{do\_buf\_copy\ caller\_ec\ callee\_ec} \ 
(\text{do\_set\_regs\ callee\_ec} \ 
(\text{nova.wp\ callee\_ec})))
\]
Example: inter-process message send, simplified

\{
\text{nova\_src\_buf} \rightarrow \text{msg\_bytes0} * P \text{msg\_bytes0} * \text{channel\_spec channel\_handle} P Q \}
\text{ipc\_call(channel\_handle)}
\{
\text{nova\_src\_buf} \rightarrow \text{msg\_bytes1} * Q \text{msg\_bytes1} * \text{channel\_spec channel\_handle} P Q \}
Example: inter-process message send, simplified

\{
  nova_src_buf |-> msg_bytes0
  \* P msg_bytes0 \* channel_spec channel_handle P Q \}

ipc_call(channel_handle)

\{
  nova_src_buf |-> msg_bytes1
  \* Q msg_bytes1 \* channel_spec channel_handle P Q \}

- Sufficient for undisciplined clients: no \(\times\), assumes sequential ownership (not satisfiable from invariants)!
- Other threads can write to the buffer during the call
Buffer copy with atomic triples

\{ \text{nova\_src\_buf} \rightarrow \text{msg\_bytes} \times (\exists \text{xs}, \text{nova\_dst\_buf} \rightarrow \text{xs}) \} \\
\text{ipc\_call\_copy()} \\
\{ \text{nova\_src\_buf} \rightarrow \text{msg\_bytes} \times \text{nova\_dst\_buf} \rightarrow \text{msg\_bytes} \}

\\
\\
\lll \forall \text{msg\_bytes}, \text{nova\_src\_buf} \rightarrow \text{msg\_bytes} \times (\exists \text{xs}, \text{nova\_dst\_buf} \rightarrow \text{xs}) \\rrr \\
\text{ipc\_call\_copy()} \\
\lll \text{nova\_src\_buf} \rightarrow \text{msg\_bytes} \times \text{nova\_dst\_buf} \rightarrow \text{msg\_bytes} \\rrr

► Sufficient for unverified clients: ✔️ — Sequential ownership not required!
► Implies disciplined spec: ✔️ (atomic triples imply sequential triples)
► Implementable (efficiently): ❌
  ► normal buffer read is not atomic
  ► a big kernel lock would not suffice; only stopping all other threads
  ► performance requires unsynchronized reads
  ► multiple atomic steps!
Byte copy via sequential composition

\[
\forall x, \; P \implies e \implies \exists y, \; Q \implies \text{RET } f \times y \implies :=
\forall R, \; \text{AU } \forall x, \; P x \implies \exists y, \; Q x y, \; \text{COMM } R (f \times y) \implies \text{WP } e \{\{ R \}\}
\]

do_byte_read src Q := AR \forall v, \; \text{src } \mapsto v \implies \exists Q v \implies
AR \forall x, \; P x \implies \forall R x \implies :=
\text{AU } \forall x, \; P x \implies \exists P x, \; \text{COMM } R x \implies
do_byte_write dst v Q := AC \forall w, \; \text{dst } \mapsto w \implies \exists dst \mapsto v, \; \text{COMM } Q v \implies
do_byte_copy src dst Q :=
do_byte_read src (\lambda v, \; \text{do_byte_write dst } v Q)

- Sufficient for unverified clients: ✅
- Implies disciplined spec: ✅ (sequential ownership suffices to prove AUs)
- Implementable (efficiently): ¬✅ (atomics suffice)
Non-deterministic parallel composition

For performance, NOVA does not order reads/writes to different bytes. So our final spec is:

\[
\text{do_buf_copy} \ src \ dst \ Q := \\
\quad \exists \ (Q\text{copy} : N -> \text{mpred}), \\
\quad \quad \quad \quad \quad (\forall i \in [0, 512[ \ do_{\text{byte_copy}} (src + i) (dst + i) (Q\text{copy} i)) \cdot \\
\quad \quad \quad \quad \quad (\forall i \in [0, 512[ \ Q\text{copy} i) \cdot Q)
\]

Final spec: \text{do_buf_copy} \ src \ dst \ R \cdot WP \ \text{ipc_call_copy()} \ {{ R }}

Sufficient for unverified clients: ✅
Implementable (efficiently): ✅ (relaxed atomics suffice!)
Some metrics: Approximate spec size

Specs for 12 syscalls (out of ~15): 39 commits
- ipc_call requires 7 steps + UTCB copy
- ctrl_sm: 6 steps
- ctrl_pd (selector manipulation): 2 + 2 for each selector
- 24 steps across the other 10 syscalls

We derived sequential specs for most of those.
Conclusions

Undisciplined specs simplify maintenance of kernel specs:

- Single verification of NOVA against undisciplined spec
- Derive disciplined spec
- Conjectured: robustness (robust safety?)
- Less overhead than operational semantics
- Enable end-to-end verification