LOCAL STATES

More hidden states, more modalities, and generalized invariants and ghost ownership

Hai Dang, Gregory Malecha, David Swasey
Local states?

Usually, $l \mapsto v$ means the same everywhere,

But everyone has *secrets*!

<table>
<thead>
<tr>
<th>$l \mapsto \heartsuit$</th>
<th>$l \mapsto \dagger$</th>
<th>$l \mapsto \roses$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>Bob</td>
<td>Charles</td>
</tr>
</tbody>
</table>

differently depending on one’s local states!
Interpretation depends on local states

- **not uncommon**
  - in weak memory
  - in distributed systems
  - in virtual address spaces

- a solution: indexing resources by the local state $l \mapsto \{s\} v$

$$
\begin{align*}
\text{Alice} & \quad \mapsto \{A\} v :\approx \quad l \mapsto \heartsuit \\
\text{Bob} & \quad \mapsto \{B\} v :\approx l \\
\text{Charles} & \quad \mapsto \{C\} v :\approx l
\end{align*}
$$
Indexing resources by local state: P s

- but not everything depends on some local states—local states should be *ambient*
  - hide local states with Iris’ *monPred*
  - only work with local states explicitly when needed
    - using *modalities*, inspired by Iris-based weak-mem works
- benefits:
  - cleaner for things that don’t care about local states
  - more idiomatic reasoning with explicit local states

*examples coming in a moment* ...
But what about *composing* local states?

- weak-mem + virtual address + ???

- how to compose a logic that knows about weak-mem and a logic that knows about virtual address?
- open world problem, similar to `inG`
WIP: Monotone Lenses

**solution:**

1. generalizing `monPred` with *monotone lenses* to encode that the local state type embeds some concrete local state type
2. generalizing `monPred` modalities to lens-induced *families* of modalities
3. generalizing invariants and ghost ownership
EXAMPLE:
VIRTUAL ADDRESS SPACES
Virtualization

Verified Services

VM

vCPU vCPU vMem

NOVA Hypervisor

VM local state

page tables

unverified clients

verified clients

The Future is Built on BedRock

8
Example: virtual address spaces

-goal: building a points-to ownership for a VM’s memory

va $\mapsto$ $v$ $\trianglerighteq$ $\exists$ pa, va $\mapsto_{AS}$ pa $\ast$ pa $\mapsto$ $v$

- va : virtual machine address
- pa : physical machine address

va $\mapsto_{AS}$ pa $\trianglerighteq$ in the current virtual address space $AS$,
va is mapped to pa

Page tables are local states of an address space
Building virtual points-to

- $aProp := \text{monPred AddrSpace } iProp$
  - $\approx \text{AddrSpace } \rightarrow iProp$, but “monotone”
- $va \mapsto v : aProp \approx \lambda AS, \exists pa, va \mapsto_{AS} pa \ast pa \mapsto v$
- for those without interesting interaction with $AS$, same rules:
  - $\{ va \mapsto v \} \ast va \{ \text{w. } w = v \ast va \mapsto v \}$
- with interesting interactions with page tables, use $\exists AS$ and $@AS P$
  - state-explicit modalities inspired by Iris-based weak-mem works
Example: send memory across VMs

size(L) = 512

va1 ↦ L

va2 ↦ L

The Future is Built on NOVA Hypervisor
Logically,

\[ \text{size}(L) = 512 \]

\[ \begin{align*}
\text{sender} & \quad \text{VM1} \\
\text{receiver} & \quad \text{VM2}
\end{align*} \]

\[ \begin{align*}
\text{va1} & \leftrightarrow L \\
\text{va2} & \leftrightarrow L
\end{align*} \]

\[ @AS1 \quad \text{va1} \leftrightarrow L \quad @AS2 \quad \text{va2} \leftrightarrow L \]

\[ \text{local-state explicit mode} \]

\[ \text{AS1} \quad \text{NOVA Hypervisor} \quad \text{AS2} \]

The Future is Built on BedRock
Quick summary: a **recipe** for local states in Iris

- tProp := monPred I iProp, where I is the type of the local state
- for those *without* interesting interactions with the local state,
  - lift rules for tProp
  - pro: same rules as before
- for those *with* interesting (non-local) interactions,
  - explicit reasoning with local-state modalities (eg., \( \exists i \), @i P)
  - *spoiler*: adjustments for invariants and ghost ownership
MONOTONE LENSES

Composing local states
Composing local states?

weak-mem + virtual address + ???

vProp aProp

logic that knows about weak-mem states logic that knows about address translation

→ idea: work with some general tProp and, as needed, assume that tProp “embeds” vProp and/or aProp and/or others.
Generalizing the local state

➔ idea: work with some general \( tProp \) and, as needed, assume that \( tProp \) “embeds” \( vProp \) and/or \( aProp \) and/or others.

- idea in “typeclass” style:
  - have \( tProp \) as the assertion type
  - need weak-mem? assume \( \text{HasVProp} \ tProp \)
  - need address space? assume \( \text{HasAProp} \ tProp \)

- lightweight implementation with \( \text{monPred} \)
  - quantify over arbitrary local state type \( I: \forall I, \text{monPred} I \ iProp \ (\approx tProp) \)
  - when needed, assume \( I \) “embeds” \( \text{View} \) (weak-mem states) and/or \( \text{AddrSpace} \) (address-translation states)
Monotone Lenses

Context \{I : biIndex\} \{PROP: bi\}.
Notation \( t_{\text{Prop}} := \text{monPred} \ I \ \text{PROP} \).

"I embeds the address-translation states AddrSpace"  
\( \approx \) the existence of a \textit{monotone lens} from I into AddrSpace

Context \{L_{\text{AS}} : I \rightarrow \text{AddrSpace}\}.

💡 Monotonicity to fit monPred, crucial for stability of the frame.
Monotone Lenses

Context \{I : \text{biIndex}\} \{\text{PROP} : \text{bi}\}.

Notation \text{tProp} := \text{monPred } I \text{ PROP}.

Context \{J : \text{biIndex}\} \{L : I \text{-} \text{ml} \triangleright J\}.

Structure \text{MLens} (I J : \text{biIndex}) : \text{Type} := \text{MLensMake}\{\begin{align*}
\text{mlens}_{\text{get}} & : I \rightarrow J ; \\
\text{mlens}_{\text{set}} & : J \rightarrow I ightarrow I ; \\
\text{mlens}_{\text{get}}\text{set} & : \forall \ i \ j, \text{mlens}_{\text{get}} (\text{mlens}_{\text{set}} j i) = j ; \\
\text{mlens}_{\text{set}}\text{get} & : \forall \ i, \text{mlens}_{\text{set}} (\text{mlens}_{\text{get}} i) i = i ; \\
\text{mlens}_{\text{set}}\text{set} & : \forall \ i \ j1 \ j2, \text{mlens}_{\text{set}} j1 (\text{mlens}_{\text{set}} j2 i) = \text{mlens}_{\text{set}} j1 i ; \\
\text{mlens}_{\text{get}}\text{mono} & : \text{Proper} ((\subseteq) \Rightarrow (\subseteq)) \text{mlens}_{\text{get}} ; \\
\text{mlens}_{\text{set}}\text{mono} & : \text{Proper} ((\subseteq) \Rightarrow (\subseteq) \Rightarrow (\subseteq)) \text{mlens}_{\text{set}} ; \\
\end{align*}\}.

Operations on lenses:
- product/projection
- compose
- equivalence
- disjointness
- inclusion
- \text{L}_{\text{id}} as a unit
Families of modalities

Context \{I : bilIndex\} \{PROP : bi\}.

Notation \(t\text{Prop} := \text{monPred} I \text{PROP}\).

Context \{J : bilIndex\} \{L : I \triangleright J\}.

- Resources that are local-state independent simply ignore I
- Resources that depend on some local-state J use J’s family of modalities

\[(L,j) P : t\text{Prop} \cong P \text{ holds at a local state whose } J \text{ component is } j\]

\[\exists\{L\} j : t\text{Prop} \cong \text{The current local state’s } J \text{ component is at least } j\]

- + more modalities
- interactions between families
INVARIANTS and GHOST OWN

with lenses
Invariants and ghost ownership?

weak-mem + virtual address + ???

vProp
logic that knows about weak-mem states

aProp
logic that knows about address translation

future logics

tProp
Problem: BI-general invariants and ghost own

- Each lens-dependent `monPred` benefits from invariants and ghost own
  - Iris invariants and ghost ownership are tied to iProp

- Generalization: BI with own

  ```pauline
  Class HasOwn {PROP : bi} {A : cmra} : Type := {
  own          : gname → A → PROP ;
  own_op       : ∀γ (a b : A), own γ (a ⋅ b) ⊣⊢ own γ a * own γ b ;
  own_mono     : ∀γ, Proper (flip (≼) ==> (⊢)) (own γ) ;
  own_ne       : ∀γ, NonExpansive (own γ) ;
  own_timeless : ∀γ (a : A), Discrete a -> Timeless (own γ a) ;
  own_core_persistent : ∀γ a, CoreId a -> Persistent (own γ a) .
  }
  ```

- Generalization: with modalities, invariants are general except for allocation

  ```pauline
  Definition inv_def N (P : PROP) : PROP :=
  (∀ E : coPset, ↑N ⊆ E ⊣⊢ □ P ={E \ ↑N,E}=: emp))%I.
  ```

⇒ more use of siProp
Invariants for monPred’s

- Iris invariants are lifted into **objective invariants** for monPred
  - resources stored in invariants must be independent of the local state \( l \)
  - Objective \( P := \forall \ i1 \ i2, \ P \ i1 \vdash \ P \ i2 \)
  - Objective \( \@i \ P \)
  - ALLOC: Objective \( P \rightarrow \ \triangleright \ P \vdash \mid\{E\}=\Rightarrow \ inv \ N \ P \)
  - INTRO: \( P \vdash \exists \ s, \ \exists i \ast @i \ P \)
  - ELIM: \( @i \ P \vdash \exists i \ast P \)

- generalization for lenses:
  - ObjectiveWith \( L \ P := \forall \ (i : I) \ (j : J), \ P \ i \vdash P \ i[L := j] \)
  - \( P \) is independent of the \( J \) component of the local state
  - Components whose local states only differ in \( J \) can communicate \( P \) freely
    - \( @(L_{\text{cpu}}, c) l \mapsto v \) can be shared across CPUs that are in the same address space
  - Derived notion of local invariants
    - \( @(L_{\text{cpu}}, c) l \mapsto v \) can be put in AS-local invariants
Problems

- BI-general invariants and ghost ownership
- Algebra of lenses ?
  - interactions of families of modalities
- Adequacy of wp ?
- bilIndex that depends on the logic ?
  - an abstraction from one lens to another that depends on the logic
  - similar to higher-order ghost state
- Cross-BI modalities ?
  - lenses generalize I in monPred I PROP
  - what about PROP? from monPred I PROP1 to monPred I PROP2?
- Proofmode/Automation ?
CONCLUSION

- monPred to hide local states and local-state modalities to expose them when needed
- monotone lenses to abstract over and compose local states
- generalized invariants and ghost ownership as useful features of BIs
- work-in-progress, with quite a few TODOs

THANK YOU
Local states?

{ isLock(s, P) }
  s.lock();
{ P }
...
...
...
{ isLock(s, P) * P }
  s.unlock();
{ emp }

× not the kind of “local” we focus on here
Local states?

```cpp
#foo.hpp
static int x = 0;
class foo {
    int f() { ... ; x = 1; ... }
};
```

compilation-unit local statics

```cpp
#bar.cpp
int bar() {
    int arr[N];
    ...
};
```

thread-local stack variables

➡️ often modeled as *explicit* resources (points-to)
**Implicit local states**

Local states are ambient:

1. they are always around/readily available
   - by threading through weakest-pre
   - more abstractly, by using non-atomic invariants

2. they should be unobtrusive
   - by hiding them with Iris’ `monPred`
Example: weak memory in Iris

\{ \text{tid} \mapsto V \ast P \ V \} \ e \ on \ \text{tid} \ \{ (v, V'). \ \text{tid} \mapsto V' \ast Q \ V' \} \\

- Most things don’t care about the view V, only memory accesses do
- In many cases, memory accesses do not have interesting interactions with the view V

⇒ Motivation for hiding views, and only let them bubble up when it’s “interesting”.

a View—presenting the local cache of a CPU

explicit ownership of the local state
Implicit weak memory states in Iris

resources may depend on the local state

\{ \text{tid} \mapsto V \ast P V \} \text{ e on tid } \{ (v, V'). \text{tid} \mapsto V' \ast Q V' \}

\{ P \} \text{ e } \{ v. Q \}

\text{P Q : monPred View iProp } \equiv \text{ View } \rightarrow \text{iProp, but "monotone"}

\text{wp e } \{ v. Q \} : \text{monPred View iProp } \equiv \\
\lambda V, \forall \text{tid, tid} \mapsto V -\ast \text{wp e on tid } \{ (v, V'). \text{tid} \mapsto V' \ast Q V' \}
**Implicit weak memory states in Iris**

\[ P \enspace Q : \text{monPred View iProp} \approx \text{View} \rightarrow \text{iProp}, \text{but "monotone"} \]

**completely local**

\[
\{ \text{emp} \} \; \; \{ v. \; v = z_1 + z_2 \} \]

\[
\{ l \mapsto _\_ \} \; l := \text{na} \; v \{ l \mapsto v \}
\]

**unobtrusive**

**non-local cross-core communication?**

\[
\{ l \mapsto \text{at } \_ \} \; l := \text{at} \; v \{ \exists V \in l \mapsto \text{at} (v,V) \}
\]

\[
\{ l \mapsto (v,V) \} \ast \text{at} \; l \{ v. \; l \mapsto (v,V) \ast \exists V \}
\]

**the local state temporarily explicit with modalities**
Communicating local-state dependent resources

\{ l \mapsto _{at} \_ \} \mid := \mid _{at} v \{ \exists V \ast \mid \mapsto _{at} (v,V) \} \\

⇒ releasing resources:
\{ P \ast \mid \mapsto _{at} \_ \} \mid := \mid _{at} v \{ \exists V \ast @V P \ast \mid \mapsto _{at} (v,V) \}

implicitly local-state dependent

explicitly local-state dependent

\{ l \mapsto _{at} (v,V) \} \ast_{at} \{ v. \mid \mapsto _{at} (v,V) \ast \exists V \}

⇒ acquiring resources:
\{ @V P \ast \mid \mapsto _{at} (v,V) \} \ast_{at} \{ v. \mid \mapsto _{at} (v,V) \ast \exists V \ast P \}
Some properties

INTRO: \( P \vdash \exists j, \{ L \} j \ast @(L, j) \ P \)

ELIM: \( @(L, j) \ P \vdash \{ L \} j - \ast P \)

COMM: \( Lj \#\# Lk \rightarrow @(Lj, j) @(Lk, k) \ P \vdash @(Lk, k) @(Lj, j) \ P \)