Integrating Iris into the Verified Software Toolchain, and vice versa

William Mansky, University of Illinois Chicago
Iris Workshop, May 23rd, 2023
The Verified Software Toolchain (VST)

• Concurrent separation logic verifier in Coq

• Higher-order, step-indexed, symbolic execution + entailment solving

• Specialized to C, connected to CompCert
  • End-to-end soundness theorem
Iris and VST

- Proof mode
- Custom ghost state
- Invariants
- Logical atomicity
- ...

C isn’t garbage-collected, so logic shouldn’t be affine
Ownership can’t be “just ghost state”: it’s translated to CompCert permissions and used for adequacy
Iris in VST

- Proof mode
- Custom ghost state
- Invariants
- Logical atomicity
- ...

• Keep VST’s foundations the same, import or reconstruct the features we want
Iris in VST

- Proof mode
- Custom ghost state
- Invariants
- Logical atomicity
- ...

- Keep VST's foundations the same
- the core of the model and soundness proof, import or reconstruct the features we want
VST on Iris

- Replace VST’s foundations with Iris, rebuild the rest of VST on top, get Iris features for free
Iris in VST

- Proof mode
- Custom ghost state
- Invariants
- Logical atomicity
- ...

• Keep VST’s foundations the same: the core of the model and soundness proof, import or reconstruct the features we want
VST + MoSeL

Get IPM by instantiating the BI interface with VST’s logic

Non-obvious parts of BI:
• Step-indexing and ▷ — but they’re exactly the same in VST as in Iris
• Persistence (□) modality
  • Default definition: (□P)(x) when (P)(core x)
  • VST has core too, but core is always emp!
  • Simple instantiation: (□P)(x) when (P ∧ emp)(x)

MoSeL: A General, Extensible Modal Framework for Interactive Proofs in Separation Logic, Krebbers et al., ICFP 2018
VST + MoSeL

Immediately get MoSeL tactics for VST’s separation logic

```plaintext
start_function
(** For each assignment statement, "symbolically execute" it ** using the forward tactic *)
forward. (* w = NULL; *)
forward. (* v = p; *)
(** To prove a while-loop, you must supply a loop invariant, ** in this case (EX s1 PROP(...LOCAL(...)(SEP(...)). *)
forward_while
(EX s1: list val, EX s2 : list val,
  EX w: val, EX v: val,
  PROP (sigma = rev s1 ++ s2)
  LOCAL (temp _w w; temp _v v)
  SEP (listrep s1 w; listrep s2 v)).

(** The forward_while tactic leaves four subgoals, ** which we mark with * (the Coq "bullet") *)
* (* Prove that precondition implies loop invariant *)
go_lower.
iIntros "list".
iExists nil, _, nullval, _.
iSplit; first done.
by iFrame.
```

1 goal
Espec : OracleKind
sigma : list val
p : val
Delta_specs : Maps.PTree.t funspec
PNp : is_pointer_or_null p
H : is_pointer_or_null (Vlong (Int64.repr (Int.signed (Int.repr 0)))))

listrep sigma p
|-- EX (a a0 : list val) (a1 a2 : val),
   !! (sigma = rev a ++ a0 ∧ a1 = Vlong (Int64.repr (Int.signed (Int.repr 0))) ∧ a2 = p) ∧&
   [listrep a a1 * listrep a0 a2]
VST + MoSeL

Immediately get MoSeL tactics for VST’s separation logic
VST + MoSeL

Immediately get MoSeL tactics for VST’s separation logic

• Don’t get wp tactics, but VST has its own (forward)
• Don’t yet get tactics for invariants, updates, atomic updates, etc. – those have their own classes to instantiate
• But first, we need those things to exist in VST!

MoSeL: A General, Extensible Modal Framework for Interactive Proofs in Separation Logic, Krebbers et al., ICFP 2018
Ghost State in VST

• Model of Iris: \( M \triangleq \prod_{i \in I} \mathbb{N} \rightarrow M_i \),
  where cameras \( M_i \) may include predicates

• Model of VST (“rmap”): \( R \triangleq \text{loc} \rightarrow \text{res} \),
  where res may include predicates (“predicates in the heap”)

• New model of VST: \( R \ast M \)
  • In practice, we don’t get HO ghost state, just agreement
Ghost State Updates in VST

• New model of VST: $R \ast M$ (approximately)
• Now we can define $\Rightarrow$, and add updates between steps in our semantics
  • And instantiate BUpd class, use iMod and iModIntro
• Hoare rules are unchanged, since program ops ignore ghost state
• Adequacy proof basically unchanged

• Used for simple double-increment example, external state reasoning

Verifying an HTTP Key-Value Server with Interaction Trees and VST, Zhang et al., ITP 2021
Invariants in VST

• Invariants can be built out of ghost state

• In Iris: \( W \triangleq \bullet (\triangleright I) * \bigstar (\triangleright I(i) * \text{dis}(i)) \lor \text{en}(i) \) where \( I \) is a map from names to assertions

• This is exactly what we can’t do with our restricted HO ghost state!

• Refactored construction:
  \[
  W \triangleq \bullet G * \bigstar \text{ag}_{G(i)} I(i) * ((\triangleright I(i) * \text{dis}(i)) \lor \text{en}(i))
  \]

• Satisfies all the same proof rules, and we can build namespaces, instantiate proofmode classes for invariants, etc. on top of it
Fancy Updates in VST

\[ \frac{\varepsilon \leftarrow x \mapsto \Phi(v) \mapsto \varepsilon}{\varepsilon \leftarrow x \mapsto \Phi(v) \mapsto \varepsilon} \]

\[ \text{WP-ATOMIC} \]

\[ \frac{\varepsilon_1 \leftarrow \varepsilon_2 \leftarrow \Phi(v) \mapsto \varepsilon_2}{\varepsilon_1 \leftarrow \varepsilon_2 \leftarrow \Phi(v) \mapsto \varepsilon_2} \]

\[ \{x \mapsto 3 \lor x \mapsto 4\} \]
\[ x \leftarrow 3 \parallel x \leftarrow 4 \]
\[ \{x \mapsto 3 \lor x \mapsto 4\} \]

• In C, this is undefined behavior!
Fancy Updates in VST

\[ \text{WP-VUP} \]
\[ \begin{align*}
\models_{\mathcal{E}} \text{wp}_{\mathcal{E}} \ e \ \{ v. \ \models_{\mathcal{E}} \ \Phi(v) \} & \vdash \text{wp}_{\mathcal{E}} \ e \ \{ \Phi \}
\end{align*} \]

\[ \text{WP-ATOMIC} \]
\[ \begin{align*}
\models_{\mathcal{E}_{1}} \text{wp}_{\mathcal{E}_{2}} \ e \ \{ v. \ \models_{\mathcal{E}_{2}} \ \Phi(v) \} & \vdash \text{wp}_{\mathcal{E}_{1}} \ e \ \{ \Phi \}
\end{align*} \]

\begin{align*}
\{ & x \mapsto 3 \lor x \mapsto 4 \} \\
x \leftarrow 3 & \par \par x \leftarrow 4 \\
\{ & x \mapsto 3 \lor x \mapsto 4 \} 
\end{align*}

- In C, this is undefined behavior!
- We set atomic to mean concurrency-atomic: lock acquire/release, atomic_load/store, etc., and nothing else
Fancy Updates in VST

We set atomic to mean concurrency-atomic: lock acquire/release, atomic_load/store, etc.

Unlike basic updates, this changes the semantics: “real” resources can change hands between steps.

For concurrent soundness, have to prove race-freedom, which seems true but not obvious.
Persistence in VST

• In Iris: invariants are persistent, can freely be automatically duplicated and passed between threads

• In VST: we defined $\square P$ to only hold on emp!

• Invariants really need to be affine too, but in VST nothing is affine!

• Step 1: weaken the core axiom

• Step 2: make the logic semi-linear
Persistence in VST

Step 1: weaken the core axiom

\[ \text{VST: } a \preceq b \rightarrow \text{core}(b) = \text{core}(a) \]
\[ \text{Iris: } a \preceq b \rightarrow \text{core}(b) \preceq \text{core}(a) \]

• In Iris, the core of \((\mathbb{N}, +)\) can tell us “the value is at least \(n\)”; in VST, it can only tell us “the value is at least 0”

• Simple solution: weaken VST’s core axiom
  • Heap resources still have trivial cores, but ghost state doesn’t have to
  • Now we can define useful persistence
  • And all the existing proofs still work
Persistence in VST

Step 2: make the logic *semi-linear*

- ORA idea from MoSeL: equip algebras with an *extension order* ⊑ describing which resources can be thrown away
- Define predicates to be closed under ⊑
- Surprisingly, VST’s model also has a slot for this order! Included in 2009, never mentioned in a paper or instantiated nontrivially
- We choose: \((r, m) \sqsubseteq (r', m') \triangleq r' = r \land m \leq m'\)
- Now all ghost state is affine, and all ghost state cores (including invariants) are intuitionistic!
Using Iris in VST

- We now have custom ghost state, invariants, updates, and all the relevant Iris tactics in VST
- Can import definitions like logical atomicity directly
Using Iris in VST

- We now have custom ghost state, invariants, updates, and all the relevant Iris tactics in VST
- Can import definitions like logical atomicity directly

```
- Intros i 11 keys; forward. forward.
  rewrite -> sub_repr, and_repr; simpl.
  rewrite -> Zland_two_p with (n := 14) by lia
  replace (Z ^ 14) with size by (setoid_rewrite (proj2_sig has_size); auto)
  exploit (Z_mod_lt ii size); [lia | intro H11]
  assert PROP (Zlength entries = size) as Hentries by entail!
  assert (0 <= ii mod size < Zlength entries) as H11 by lia
  match goal with [H : Forall _ _ [H' => pose proof (Forall Znth _ _ H11' H)] as Hptr end]
  destruct (Znth (ii mod size) entries) as (pki, pvi) eqn: Hpi: destruct Hptr.
  forward; setoid_rewrite Hpi.
  { entail!, !}
  assert (Zlength (rebase keys (hash k)) = size) as Hrebase
  { rewrite Zlength_rebase; replace (Zlength keys) with size; auto; apply hash_range. }
  forward_call atomic_load_int (pki, top, empty).
  fun v : Z => AS * ghost_snap v (Znth (ii mod size) lg)).
  { rewrite !sepccon_assoc; apply sepccon_derivs. ![cancel].
    intros 
    simpl.
    iDestruct ('AS') as (HT) ![hashtable Hclose]!; simpl.
    iDestruct "hashtable" as (T) ![is & excl & entries].
    rewrite -> (ite_sepccon Znth' with (d := Inhabitant Z) (i := ii mod size) by
    (try apply Cveric; rewrite Zlength_upto ZNat.id; lia).
    rewrite Znth_upto by (rewrite -> ?Zlength_upto, ZNat.id; lia).
    unfold hashtable_entry at 1.
    rewrite Hpi.
```

```
Iris in VST: Summary

• All the concurrency features of Iris, in VST
• Foundational changes: ghost state in the model, weaker core axiom, extension order for affine ghost state, fancy updates in the semantics
• Now we can prove atomic specs for concurrent C programs, using VST for C code and switching to Iris tactics for concurrency reasoning
• Could be useful for other non-Iris verifiers that want Iris features
• Can now reconstruct ghost-state-based reasoning in VST, e.g. ReLoC

• Paper on arXiv, opam package coq-vst-iris
Iris in VST: Summary

• Now we can prove atomic specs for concurrent C programs, using VST for C code and switching to Iris tactics for concurrency

But:

• Concurrent soundness is still complicated
• We’re reconstructing Iris features, and there’s always more we might want to reconstruct (transfinite step-indexes, later credits, …)
• We’re working in parallel to RefinedC and the whole Iris ecosystem

• What if VST was built on Iris instead?
Iris and VST

- Proof mode
- Custom ghost state
- Invariants
- Logical atomicity
- ...

- C isn’t garbage-collected, so logic shouldn’t be affine
- We can use ORAs!
- Ownership can’t be “just ghost state”: it’s translated to CompCert permissions and used for adequacy
- Need a fancier relationship between physical state and mapsto assertions

Verified Software Toolchain
VST on Iris

• Replace VST’s foundations with Iris, rebuild the rest of VST on top, get Iris features for free
VST on Iris: “juicy” view

• State interpretation: \( \sigma \) where \( \sigma \) is a map from locations to values
• Maps-to: \( l \mapsto v \) is defined as \( \sigma \circ \{[l := v]\} \)

\[ \sigma \ast l \mapsto v \vdash \sigma(l) = v \]

• In VST, these don’t coincide!
  • Physical memory (CompCert) maps locations to values + permissions (readable, writable, etc.)
  • Logical memory maps locations to rmap resources + shares
  • Semantics defined in terms of a “juicy mem” that includes both CompCert mem and rmap, plus proof that they are coherent
VST on Iris: “juicy” view

• General views: parameterized by a relation $R$, give:

  • $a \ast \circ b \vdash R\ a\ b$

• In VST, we can choose $R \triangleq \text{coherent}$, and get:

  • $m \ast l \mapsto_{\pi} v \vdash \text{coherent } m\ l\ \pi\ v$
VST on Iris: “juicy” view

• General views: parameterized by a relation $R$, give:

  $$a \ast \circ b \vdash R \ a \ b$$

• In VST, we can choose $R \triangleq \text{coherent}$, and get:

  $$m \ast l \mapsto \pi \ v \vdash \text{coherent} \ m \ l \ \pi \ v$$

Old VST:

$$\forall j. \ (l \mapsto v) \ (\text{rmap}_{of} \ j) \rightarrow \text{valid}_{pointer} \ l \ (\text{mem}_{of} \ j)$$

VST on Iris:

$$m \ast l \mapsto v \vdash \neg \text{valid}_{pointer} \ l \ m \ \neg$$
VST on Iris: semantics

• Iris: \( wp_e \{ \Phi \} \) when either \( e \) is terminated in a state satisfying \( \Phi \), or \( S(\sigma) \Rightarrow (e, \sigma) \rightarrow (e', \sigma') \Rightarrow S(\sigma') \ast wp_e' \{ \Phi \} \)

• VST defines safety similarly, except that there are two kinds of steps:
  • Core steps are steps by the Clight semantics
  • External calls call arbitrary external functions with provided pre- and postconditions

• Safety was originally defined as a relation on juicy mems, but we can rephrase it inside the logic analogously to \( wp \)
VST on Iris: program logic

• Proved exactly the same triples for C statements (mod. Iris notation)
• Proofs are about ½ the size of old versions
  • #1 reduction: reasoning at the logic level instead of unfolding to the model
  • #2 reduction: proof mode tactics
VST on Iris: adequacy

• Still in progress: should be the same paper proof, but in Iris terms
• Aim to prove as much as possible (probably everything!) in the logic instead of unfolding to the model

• VST has complicated armature for lifting CompCert’s soundness to concurrency; it should be easier with Iris, but basically the same
• We’re long overdue for a better approach to compiler correctness for concurrency! Happy to talk if you have ideas.
VST on Iris: user interface

• Still need to rebuild symbolic execution tactics and automation

• Interaction mode 1: VST + Iris
  • Can do anything we did before in VST in exactly the same way
  • Drop into Iris proof mode as desired for invariants, atomics, etc.

• Interaction mode 2: Iris style
  • Turn Hoare triples into WP format, stay in IPM the whole time
  • Will require retooling VST’s automation (forward, etc.) to work on IPM goals
  • More comfortable for Iris people, could adapt Diaframe
Conclusion

• Iris in VST: mostly done
  • Can prove logically atomic specs for C programs using Iris logic and tactics
  • Takes cues from Iris, reuses some of it, rebuilds a lot more

• VST on Iris: looks like it’ll work!
  • More expressive ghost state
  • Can incorporate new Iris ideas: transfinite step-indexing, later credits, ...
  • Integrate with other tools? Diaframe, RefinedC, ...
  • What would you do with a CompCert C mode for Iris?
VST on Iris: ownership

- Iris mapsto is simple: $l \mapsto_q v$, where $q$ is a positive fraction
  - Any $q$ is enough to read, 1 is required to write
- VST uses tree shares, with 4 distinct permission levels (corresponding to CompCert permission levels): nonempty, readable, writable, freeable
- Nonempty ownership gives knowledge of the location, but not its value!

\begin{verbatim}
Inductive shared :=
| YES (dq : dfrac) (rsh : readable_dfrac dq) (v : agree V)
| NO (sh : share0) (rsh : ¬readable_share' sh).
\end{verbatim}

- $l \mapsto_q v$ is \{[$l := YES q \_ v$]\}
- Also have $l \mapsto_q \bot$, which is \{[$l := NO q \_ ]$\}
VST on Iris: resources

• Model of VST: \( R \triangleq \text{loc} \rightarrow \text{res} \), where \( \text{res} \) may include predicates ("predicates in the heap")

• \( \text{res} \triangleq \text{VAL} \, v \mid \text{LK} \, R \mid \text{FUN} \, A \, P \, Q \)

• Last two use "predicates in the heap"
  • But in Iris they don’t need to! We’ll come back to this
VST on Iris: predicates in the heap

• Model of VST: $R \triangleq \text{loc} \rightarrow \text{res}$, where res may include predicates ("predicates in the heap")

• $\text{res} \triangleq \text{VAL } v \mid \text{LK } R \mid \text{FUN } A P Q$

• We can take the predicates out of the heap, and use ghost state/invariants for them instead
VST on Iris: predicates in the heap

• Model of VST: \( R \triangleq \text{loc} \rightarrow \text{res} \), where res may include predicates ("predicates in the heap")

• \( \text{res} \triangleq \text{VAL} \, v \mid \text{LK} \mid \text{FUN} \, A \, P \, Q \)

• We can take the predicates out of the heap, and use ghost state/invariants for them instead

• \( \text{isLK} \, l \, R \triangleq l \mapsto \text{LK} \ast \text{inv} \, R \)
VST on Iris: predicates in the heap

• \( \text{res} \triangleq \text{VAL} \, v \mid \text{LK} \mid \text{FUN} \, A \, P \, Q \)

\[
\text{Inductive funspec} \ := \ \text{mk_funspec: typesig} \rightarrow \ \text{calling_convention} \rightarrow \\
\quad \forall A \ (P: A \rightarrow \text{mpred}) \ (Q: A \rightarrow \text{mpred}), \ \text{funspec}.
\]

• \( l \mapsto \text{FUN} \, A \, P \, Q \) asserts that \( l \) is a function pointer w/ spec
\( \forall a: A, \{P \, a\} \, l \, \{Q \, a\} \)

• We build an OFE for funspec (roughly isomorphic to \{\( A \& (A \rightarrow \text{mpred}) \ast (A \rightarrow \text{mpred})\)}


VST on Iris: predicates in the heap

- \( \text{res} \triangleq \text{VAL} \, v \mid \text{LK} \mid \text{FUN} \)

\[
\text{Inductive funspec} := \text{mk_funspec: typesig } \rightarrow \text{calling_convention } \rightarrow \forall A (P: A \rightarrow \text{mpred}) (Q: A \rightarrow \text{mpred}), \text{funspec}.
\]

- \( l \mapsto \text{FUN} \, A \, P \, Q \) asserts that \( l \) is a function pointer w/ spec \( \forall a: A, \{P \, a\} \, l \, \{Q \, a\} \)

- We build an OFE for funspec (roughly isomorphic to \( \{A \& (A \rightarrow \text{mpred}) \star (A \rightarrow \text{mpred})\} \))

- Define \( \text{isFUN} \, l \, f \triangleq l \mapsto \text{FUN} \ast \circ \{[l := \triangleright \, f]\} \)
  - Analogous to invariant construction