

A Program Logic to Reason About Well-Bracketed Control Flow

Amin Timany¹, Lars Birkeedal¹, Armaël Guéneau²

¹Aarhus University, Aarhus, Denmark

²Université Paris-Saclay, CNRS, ENS Paris-Saclay, INRIA, Laboratoire Méthodes Formelles

May 24th, 2023,

Third Iris workshop,

MPI-SWS

Saarbrücken, Germany

Well-bracketed control flow

Well-bracketed control flow:

In every function call, the caller resumes execution **exactly once when** the callee terminates.

In other words, function calls and returns form a stack.

This is often the case, except in the presence of features that break well-bracketedness like concurrency and continuations.

The Very Awkward Example (VAE)

Well-bracketedness is often captured via the **Very Awkward Example (VAE)** (attributed to Jacob Thamsborg):

```
let r = ref(0) in λf. r ← 0; f (); r ← 1; f (); !r
```

 (Very Awkward)

We expect that this program should always return 1 **in a well-bracketed setting** but the reasoning is rather subtle:

- Intuitively, r is not accessible from outside the closure hence, it could not be influenced from “outside”.
- However, the closure can itself be called within the callback used to call the closure.

```
let g = let r = ref(0) in λf. r ← 0; f (); r ← 1; f (); !r in  
g (λ_. g (λ_. ()); ())
```

 (VAE-self-applied)

The Very Awkward Example (VAE)

[Dreyer et al., 2010] introduced a logical relations model featuring state transition systems with **public** (solid) and **private** (dashed) transitions.

Every function can locally make private transitions but must be in a publicly reachable state upon returning.



(VAE-sts)

This approach has been successfully replicated by others, e.g., [Skorstengaard et al., 2019] and [Georges et al., 2021].

The latter uses Iris to hide all the details of the model, step-indexing, resources, *etc.*, **but still uses an explicit collection of state transition systems.**

Central question

Central question: Can we do better? Can we avoid explicit state transition systems?

Desiderata for the ideal solution (Our solution achieves all):

Conservative Extension A Hoare logic based on separation logic (Iris) which validates **verbatim** all the usual reasoning principles of separation logic for a sequential language,

Bespoke Facilities with **additional** reasoning principles for reasoning about well-bracketedness.

Minimal Extension It uses existing reasoning facilities, *e.g.*, Iris's ghost state and invariants, instead of public/private transitions.

Versatility It can be used to construct unary and binary logical relations models **in the standard way**, *i.e.*, as presented in [Krebbers et al., 2017, Timany et al., 2022], to obtain a model which enables reasoning about well-bracketedness **without explicit state transition systems**.

The Awkward Example

Let us take a look at a closely related example.

The VAE is an adaptation of [Pitts and Stark, 1999]

`let $r = \text{ref}(0)$ in $\lambda f. r \leftarrow 1; f ()$; ! r` (Awkward)

One would use the following state transition system for Awkward:



(Awkward-sts)

It has only one public transitions and no private transition.

The Akward Example

We **can** prove that Awkward returns 1 in Iris without using state transition systems explicitly.

The spec for Awkward:

$$\begin{aligned} & \{\text{True}\} \\ & \quad \text{let } r = \text{ref}(0) \text{ in } \lambda f. r \leftarrow 1; f (); !r \qquad \qquad \qquad (\text{awkward-spec}) \\ & \{g. \forall f. \{\{\text{True}\} f () \{x. x = ()\}\} g f \{x. x = 1\}\} \end{aligned}$$

The Akward Example

The idea is to use the following one-shot ghost theory:

MAKE-ONE-SHOT

$\vdash \text{F} \Rightarrow \exists \gamma. \text{pending}(\gamma)$

SHOOT

$\text{pending}(\gamma) \vdash \text{F} \Rightarrow \text{shot}(\gamma)$

PENDING-NOT-SHOT

$\text{pending}(\gamma) * \text{shot}(\gamma) \vdash \text{False}$

SHOT-PERSISTENT

$\vdash \text{persistent}(\text{shot}(\gamma))$

together with the following invariant:

$$\text{AwkwardInv} \triangleq \boxed{(\ell \mapsto 0 * \text{pending}(\gamma)) \vee (\ell \mapsto 1 * \text{shot}(\gamma))} \quad (\text{awkward-invariant})$$

In Iris, ghost resources and frame-preserving updates correspond to **public** transitions.

Counterpart for private transitions

If Iris's ghost theory and frame-preserving updates capture public transitions, what then would allow us to capture the kind of reasoning supported by private transitions?

Answer: We introduce the **well-bracketed program logic** (well-bracketed Hoare triples)


- **built on top of the existing Hoare logic of Iris**
- It satisfies all the desiderata mentioned earlier.
- Enables reason about well-bracketing in HeapLang.
- Features proof rules to allocate and manipulate **stacks** of ghost names.

The central idea: to make a private transition from s to s' we do the following:

- The **current** ghost resource instance is the one on top on stack.
- Leave the current instance (tracking s) untouched.
- Allocate a fresh instance in state s' and push it onto the stack.
- To roll back, *i.e.*, to go to a publicly reachable state, we just pop the stack.

Well-bracketed Hoare triples

The stack mask: a set of (*open*) stack names

$$\langle P \rangle e \langle x. Q \rangle^{\mathcal{O}}$$


Just like ordinary Hoare triples of Iris with an added **stack mask**: a set of *open* stack names. We omit \mathcal{O} when it is empty.

It satisfies the counterpart to all WP rules for HeapLang except for WP-FORK:

WBHOARE-CONSEQUENCE

$$\frac{Q \implies P \quad \forall v. \Phi(v) \implies \Psi(v)}{\langle P \rangle e \langle \Phi \rangle^{\mathcal{O}} \vdash \langle Q \rangle e \langle \Psi \rangle^{\mathcal{O}}}$$

WBHOARE-INV-OPEN

$$\frac{\langle I * P \rangle e \langle x. I * Q \rangle^{\mathcal{O}} \quad e \text{ is atomic}}{\langle \boxed{I} * P \rangle e \langle x. Q \rangle^{\mathcal{O}}}$$

WBHOARE-BIND

$$\frac{\langle P \rangle e \langle \Psi \rangle^{\mathcal{O}} \quad \forall v. \langle \Psi(v) \rangle K[v] \langle \Phi \rangle^{\mathcal{O}}}{\langle P \rangle K[e] \langle \Phi \rangle^{\mathcal{O}}}$$

WBHOARE-FRAME

$$\langle P \rangle e \langle \Phi \rangle^{\mathcal{O}} \vdash \langle P * R \rangle e \langle x. \Phi(x) * R \rangle^{\mathcal{O}}$$

WBHOARE-ALLOC

$$\vdash \langle \text{True} \rangle \text{ref}(v) \langle x. x \mapsto v \rangle^{\mathcal{O}}$$

WBHOARE-LOAD

$$\vdash \langle \ell \mapsto v \rangle !\ell \langle x. x = v * \ell \mapsto v \rangle^{\mathcal{O}}$$

WBHOARE-STORE

$$\vdash \langle \ell \mapsto v \rangle \ell \leftarrow w \langle x. x = () * \ell \mapsto w \rangle^{\mathcal{O}}$$

Hence, we can ignore stacks when not reasoning about well-bracketedness.

Reasoning principles for well-bracketedness (stacks)

An authoritative resource algebra of stacks ($stack_{\bullet}$ and $stack_{\circ}$).

$stack_{\bullet}$: managed by the program logic (except if the stack is open, *i.e.*, in the mask \mathcal{O}).

$stack_{\circ}$: given to the user.

STACKS-AGREE

$$stack_{\bullet}(N, s) * stack_{\circ}(N, s') \vdash s = s'$$

STACK-FULL-UNIQUE

$$stack_{\bullet}(N, s) * stack_{\bullet}(N, s') \vdash \text{False}$$

STACK-FRAGMENT-UNIQUE

$$stack_{\circ}(N, s) * stack_{\circ}(N, s') \vdash \text{False}$$

STACK-EXISTS

$$stack_{\circ}(N, s) \vdash stack_{\exists}(N)$$

STACKS-PUSH

$$stack_{\bullet}(N, s) * stack_{\circ}(N, s) \vdash \Leftrightarrow stack_{\bullet}(N, \gamma :: s) * stack_{\circ}(N, \gamma :: s)$$

STACKS-POP

$$stack_{\bullet}(N, \gamma :: s) * stack_{\circ}(N, \gamma :: s) \vdash \Leftrightarrow stack_{\bullet}(N, s) * stack_{\circ}(N, s)$$

Stack rules

Throughout the proof we can:

Allocate new stacks:

$$\begin{array}{c} \text{WBHOARE-CREATE-STACK} \\ (\forall N, \langle P * \text{stack}_\circ(N, [\gamma]) \rangle) e \langle \Phi \rangle^\circ \vdash \langle P \rangle e \langle \Phi \rangle^\circ \end{array}$$

Access, i.e., open, stacks in a delimited way: [In VAE: around the body of the closure]

WBHOARE-ACCESS-STACK

$$N \notin \mathcal{O}$$

$$\frac{}{\left(\forall s. \langle P * \text{stack}_\bullet(N, s) \rangle e \langle x. \Phi(x) * \text{stack}_\bullet(N, s) \rangle^{\mathcal{O} \cup \{N\}} \right) \vdash \langle \text{stack}_\exists(N) * P \rangle e \langle \Phi \rangle^\circ}$$

Mend, i.e., temporarily close, stacks: [In VAE: around the callback]

WBHOARE-MEND-STACK

$$\langle P \rangle e \langle \Phi \rangle^{\mathcal{O} \setminus \{N\}} \vdash \langle \text{stack}_\bullet(N, s) * P \rangle e \langle x. \Phi(x) * \text{stack}_\bullet(N, s) \rangle^\circ$$

Note how WBHOARE-MEND-STACK is the dual of WBHOARE-ACCESS-STACK

Specs and proof of VAE

We prove the following about VAE:

$$\begin{aligned} & \langle \text{True} \rangle \\ & \quad \text{let } r = \text{ref}(0) \text{ in } \lambda f. r \leftarrow 0; f (); r \leftarrow 1; f (); !r \quad (\text{VAE-spec}) \\ & \langle g. \forall f. (\langle \text{True} \rangle f ()) \langle x. x = () \rangle \rangle g f \langle x. x = 1 \rangle \end{aligned}$$

This exactly the spec for Awkward but using well-bracketed triples instead of ordinary triples.

The invariant that we use for proving VAE-spec is as follows:

$$\text{VAEInv} \triangleq \boxed{\exists \gamma, s. \text{stack}_o(N, \gamma :: s) * (\ell \mapsto 0 * \text{pending}(\gamma)) \vee (\ell \mapsto 1 * \text{shot}(\gamma))} \quad (\text{VAE-invariant})$$

exactly the invariant for Awkward

except γ is the one on top of the stack

Proof of VAE

The proof is very similar to that of Akward with some additional steps (in the following order):

- We allocate a stack N to establish the invariant and remember the name using $stack_{\exists}(N)$.
- We access stack N around inside the closure.
- When we write 0 to ℓ , we push a new instance (in pending state) on the stack.
- We mend the stack before each callback (the callback may use stacks; **its mask is also empty**).
- When we write 1 to ℓ , we pop the stack
- If necessary perform **SHOOT** resource update. [This also happens in Awkward.]

A brief look at the internals of well-bracketed triples

Well-bracketed Hoare triples are defined in terms of well-bracketed weakest preconditions:

$$\langle P \rangle e \langle \Phi \rangle^{\mathcal{O}} \triangleq \square (P \multimap \text{wbwp } e \langle \Phi \rangle^{\mathcal{O}})$$

Well-bracketed weakest preconditions are defined in terms of Iris weakest preconditions:

A finite map from stack names to stack contents

$$\text{wbwp } e \langle \Phi \rangle^{\mathcal{O}} \triangleq \forall \mathcal{S}. \text{AllStacksExcept}(\mathcal{S}, \mathcal{O}) \multimap$$

Ownership of all *stack*_• resource in \mathcal{S} except for those in \mathcal{O}

$$\text{wp } e \{x. \exists \mathcal{S}'. \mathcal{S} \subseteq \mathcal{S}' \multimap \text{AllStacksExcept}(\mathcal{S}', \mathcal{O}) \multimap \Phi(x)\} \quad (\text{wbwp-definition})$$

↑

The crux: well-bracketed programs may allocate new stacks,
but have to preserve the state of each and every stack.

The relation to ordinary Hoare triples and concurrency

Ordinary Hoare triples imply well-bracketed Hoare triples:

$$\text{HOARE-WBHOARE} \\ \{P\} e \{ \Phi \} \vdash \llbracket P \rrbracket e \llbracket \Phi \rrbracket^{\mathcal{O}}$$

Intuitively: if a program does not rely on stacks it cannot break well-bracketedness.

Upshot: We can combine reasoning and reason about innocuous non-well-bracketed concurrent programs. We can prove $\llbracket \text{True} \rrbracket h g \llbracket x. x = 1 \rrbracket$ for the following program:

```
let g = let first = ref(true) in let res = ref(None) in
  let rec wait() = if !res = None then wait () else () in
  λ_.; if !first then first ← false; fork {res ← Some(fact 1000000)} else wait ()
in let h = let r = ref(0) in λf. r ← 0; f (); r ← 1; f (); !r in
h g
```

(innocuous-concurrency)

This demonstrates that we capture the semantic essence of well-bracketedness as we can capture and reason about well-bracketedness in HeapLang including (innocuous) concurrency.

Well-bracketedness as a trace property

We use the methodology of [Birkedal et al., 2021] together with VAE-spec to prove that calls to and returns from VAE are indeed well-bracketed.

We instrument VAE (to emit events when entering and exiting the VAE closure).

$$\mathcal{L}_{seqfull} ::= \langle \tau, \text{call} \rangle \cdot \mathcal{L}_{seqfull} \cdot \langle \tau, \text{ret} \rangle \mid \mathcal{L}_{seqfull} \cdot \mathcal{L}_{seqfull} \mid \varepsilon \quad (\tau \in \text{Tag} \triangleq \text{String})$$

Here τ is a unique tag created for each call made.

We show that any trace of (partial) execution of the program always matches a prefix of $\mathcal{L}_{seqfull}$.

Logical relations models

We can add support for reasoning about well-bracketedness to the logical relations model of [Timany et al., 2022] by simply using WBWP instead of Iris's WP.

We use the obtained unary and binary models to prove well-bracketedness of VAE.

$$\begin{aligned}
 \mathcal{E}[\exists \vdash \tau]_{\Delta} &\triangleq \lambda(e, e'). \forall j, C. \text{SpecCtx} * j \mapsto C[e'] * \text{wp } e \{v. \exists v'. j \mapsto C[v'] * [\exists \vdash \tau]_{\Delta}(v, v')\} \\
 [\exists \vdash \alpha]_{\Delta} &\triangleq \Delta(\alpha) \\
 [\exists \vdash \mathbf{1}]_{\Delta} &\triangleq \lambda(v, v'). v = v' = () \\
 [\exists \vdash \mathbb{B}]_{\Delta} &\triangleq \lambda(v, v'). v = v' \in \{\text{true}, \text{false}\} \\
 [\exists \vdash \mathbf{Z}]_{\Delta} &\triangleq \lambda(v, v'). v = v' \in \mathbb{Z} \\
 [\exists \vdash \tau_1 \times \tau_2]_{\Delta} &\triangleq \lambda(v, v'). \exists v_1, v_2, v'_1, v'_2. (v = (v_1, v_2)) * (v' = (v'_1, v'_2)) * \\
 &\quad [\exists \vdash \tau_1]_{\Delta}(v_1, v'_1) * [\exists \vdash \tau_2]_{\Delta}(v_2, v'_2) \\
 [\exists \vdash \tau_1 + \tau_2]_{\Delta} &\triangleq \lambda(v, v'). \forall_{i \in \{1, 2\}} \exists w, w'. (v = \text{inj}_i w) * (v' = \text{inj}_i w') * [\exists \vdash \tau_i]_{\Delta}(w, w') \\
 [\exists \vdash \tau \rightarrow \rho]_{\Delta} &\triangleq \lambda(v, v'). \square (\forall w, w'. [\exists \vdash \tau]_{\Delta}(w, w') \rightarrow \mathcal{E}[\exists \vdash \rho]_{\Delta}(v w, v' w')) \\
 [\exists \vdash \forall \alpha. \tau]_{\Delta} &\triangleq \lambda(v, v'). \square (\forall (\Psi : \text{Val} \times \text{Val} \rightarrow \text{iProp}_{\square}). \mathcal{E}[\exists, \alpha \vdash \tau]_{\Delta, \alpha \rightarrow \Psi}(v(), v'())) \\
 [\exists \vdash \exists \alpha. \tau]_{\Delta} &\triangleq \lambda(v, v'). \exists (\Psi : \text{Val} \times \text{Val} \rightarrow \text{iProp}_{\square}). \\
 &\quad \exists w, w'. (v = \text{pack}(w)) * (v' = \text{pack}(w')) * [\exists, \alpha \vdash \tau]_{\Delta, \alpha \rightarrow \Psi}(w, w') \\
 [\exists \vdash \mu \alpha. \tau]_{\Delta} &\triangleq \mu(\Psi : \text{Val} \times \text{Val} \rightarrow \text{iProp}_{\square}). \\
 &\quad \lambda(v, v'). \exists w, w'. (v = \text{fold } w) * (v' = \text{fold } w') * \triangleright [\exists \vdash \tau]_{\Delta, \alpha \rightarrow \Psi}(w, w') \\
 [\exists \vdash \text{ref}(\tau)]_{\Delta} &\triangleq \lambda(v, v'). \exists \ell, \ell'. (v = \ell) * (v' = \ell') * \boxed{\exists w, w'. \ell \mapsto w * \ell' \mapsto w' * [\exists \vdash \tau]_{\Delta}(w, w')}^{\mathcal{N}, \ell, \ell'}
 \end{aligned}$$

$$\begin{aligned}
 \mathcal{E}[\exists \vdash \tau]_{\Delta} &\triangleq \lambda(e, e'). \forall C. \text{SpecCtx} * \mapsto C[e'] * \text{wbwp } e \{v. \exists v'. \mapsto C[v'] * [\exists \vdash \tau]_{\Delta}(v, v')\} \\
 [\exists \vdash \alpha]_{\Delta} &\triangleq \Delta(\alpha) \\
 [\exists \vdash \mathbf{1}]_{\Delta} &\triangleq \lambda(v, v'). v = v' = () \\
 [\exists \vdash \mathbb{B}]_{\Delta} &\triangleq \lambda(v, v'). v = v' \in \{\text{true}, \text{false}\} \\
 [\exists \vdash \mathbf{Z}]_{\Delta} &\triangleq \lambda(v, v'). v = v' \in \mathbb{Z} \\
 [\exists \vdash \tau_1 \times \tau_2]_{\Delta} &\triangleq \lambda(v, v'). \exists v_1, v_2, v'_1, v'_2. (v = (v_1, v_2)) * (v' = (v'_1, v'_2)) * \\
 &\quad [\exists \vdash \tau_1]_{\Delta}(v_1, v'_1) * [\exists \vdash \tau_2]_{\Delta}(v_2, v'_2) \\
 [\exists \vdash \tau_1 + \tau_2]_{\Delta} &\triangleq \lambda(v, v'). \forall_{i \in \{1, 2\}} \exists w, w'. (v = \text{inj}_i w) * (v' = \text{inj}_i w') * [\exists \vdash \tau_i]_{\Delta}(w, w') \\
 [\exists \vdash \tau \rightarrow \rho]_{\Delta} &\triangleq \lambda(v, v'). \square (\forall w, w'. [\exists \vdash \tau]_{\Delta}(w, w') \rightarrow \mathcal{E}[\exists \vdash \rho]_{\Delta}(v w, v' w')) \\
 [\exists \vdash \forall \alpha. \tau]_{\Delta} &\triangleq \lambda(v, v'). \square (\forall (\Psi : \text{Val} \times \text{Val} \rightarrow \text{iProp}_{\square}). \mathcal{E}[\exists, \alpha \vdash \tau]_{\Delta, \alpha \rightarrow \Psi}(v(), v'())) \\
 [\exists \vdash \exists \alpha. \tau]_{\Delta} &\triangleq \lambda(v, v'). \exists (\Psi : \text{Val} \times \text{Val} \rightarrow \text{iProp}_{\square}). \\
 &\quad \exists w, w'. (v = \text{pack}(w)) * (v' = \text{pack}(w')) * [\exists, \alpha \vdash \tau]_{\Delta, \alpha \rightarrow \Psi}(w, w') \\
 [\exists \vdash \mu \alpha. \tau]_{\Delta} &\triangleq \mu(\Psi : \text{Val} \times \text{Val} \rightarrow \text{iProp}_{\square}). \\
 &\quad \lambda(v, v'). \exists w, w'. (v = \text{fold } w) * (v' = \text{fold } w') * \triangleright [\exists \vdash \tau]_{\Delta, \alpha \rightarrow \Psi}(w, w') \\
 [\exists \vdash \text{ref}(\tau)]_{\Delta} &\triangleq \lambda(v, v'). \exists \ell, \ell'. (v = \ell) * (v' = \ell') * \boxed{\exists w, w'. \ell \mapsto w * \ell' \mapsto w' * [\exists \vdash \tau]_{\Delta}(w, w')}^{\mathcal{N}, \ell, \ell'}
 \end{aligned}$$

left: the binary logical relations model of [Timany et al., 2022]; right: added support for well-bracketedness and removed concurrency.

Logical relations models

We use our unary and binary logical relation models to prove correctness of VAE.

In the binary case, we show that VAE is contextually equivalent to the following program:

$$\lambda f. f (); f (); 1 \quad \text{(Straightforward)}$$

In the unary case, where logical relations implies safety, we prove safe a program that would crash if VAE returned anything except 1.

Conclusion

- We introduced a well-bracketed program logic
 - Extends and embeds Iris's program logic (for sequential programs)
 - Captures the semantic essence of well-bracketedness
 - Useful to construct logical relations to reason about well-bracketedness with minimal effort
- We established its soundness: the program trace is indeed well-bracketed

The parts of the work were not covered in this talk:

- Details of proofs
- The ghost theory of stacks, *e.g.*, definition of `AllStacksExcept`, and its construction
- Coq proofs

References I



Birkedal, L., Dinsdale-Young, T., Guéneau, A., Jaber, G., Svendsen, K., and Tzevelekos, N. (2021).

Theorems for free from separation logic specifications.

[Proc. ACM Program. Lang.](#), 5(ICFP).



Dreyer, D., Neis, G., and Birkedal, L. (2010).

The impact of higher-order state and control effects on local relational reasoning.

In [Proceedings of the 15th ACM SIGPLAN International Conference on Functional Programming, ICFP '10](#), page 143–156, New York, NY, USA. Association for Computing Machinery.



Georges, A. L., Guéneau, A., Van Strydonck, T., Timany, A., Trieu, A., Huyghebaert, S., Devriese, D., and Birkedal, L. (2021).

Efficient and provable local capability revocation using uninitialized capabilities.

[Proc. ACM Program. Lang.](#), 5(POPL).

References II



Krebbers, R., Timany, A., and Birkedal, L. (2017).

Interactive proofs in higher-order concurrent separation logic.

[In Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017, pages 205–217.](#)



Pitts, A. M. and Stark, I. D. B. (1999).

[Operational Reasoning for Functions with Local State](#), page 227–274.

[Cambridge University Press, USA.](#)



Skorstengaard, L., Devriese, D., and Birkedal, L. (2019).

Stktokens: Enforcing well-bracketed control flow and stack encapsulation using linear capabilities.

[Proc. ACM Program. Lang.](#), 3(POPL).

References III



Timany, A., Krebbers, R., Dreyer, D., and Birkedal, L. (2022).

A logical approach to type soundness.

[Reported under submission on https://iris-project.org/.](https://iris-project.org/)