The Accompanying Technical Appendix of the paper: The Logical Essence of Well-Bracketed Control Flow

AMIN TIMANY, Aarhus University, Denmark
ARMAËL GUÉNEAU, Université Paris-Saclay, CNRS, ENS Paris-Saclay, Inria, LMF, France
LARS BIRKEDAL, Aarhus University, Denmark

1 CONSTRUCTION OF THE PRESENTED GHOST THEORIES

So far we have seen the essential building blocks of the well-bracketed program logic in terms of predicates which are defined using so-called *ghost resources*. In this section we will discuss what ghost resources are in general in Iris and outline how they can be used to define these predicates. Readers who are not interested in all the details of the construction of the presented predicates in terms of ghost resources of Iris could simply ignore this section as understanding it is not instrumental for understanding the main contributions of the paper which are presented in other sections. All the constructions we present in Section 1.1 are standard in Iris [Jung et al. 2018]; in Section 1.2 we show how to use the existing constructions to define the new stack resources.

In Iris ghost resources are modeled using an algebraic structure called resource algebras which are essentially partial commutative semi-groups where the operation represents composition of resources. Formally, the resource algebras are step-indexed in Iris, but here we omit discussion of step-indexing for the sake of brevity and clarity. We also simplify the description of persistence. We remark that our simplifications are immaterial for what we discuss in this paper.

Being a partial commutative semi-group, a resource algebra consists of a carrier set M together with a *partial* binary operation $\cdot : M \times M \to M$ that is associative and commutative. The partiality here is captured by working with a *valid* subset of the elements of the resource algebra. We write $\checkmark_M(m)$ to say that m is valid; we drop the subscript M whenever it is clear what is meant. The only requirement of the valid set is that it respects resource compositions in the sense that constituents of valid resources must also be valid:

$$\checkmark(m \cdot m') \implies \checkmark(m) \wedge \checkmark(m')$$

The way resource algebras are used in Iris reasoning is using the following predicate called ownership: $\boxed{m : M}^{\gamma}$; we drop $: M$ part whenever it is clear what resource algebra M is from the context. The basic rules for working with resources are as follows:

OWN-ALLOC
$$\checkmark_M(m) * \mathsf{infinite}(A) \vdash \Rrightarrow \exists \gamma \in A.\ \boxed{m : M}^{\gamma}$$

OWN-OP
$$\boxed{m \cdot m' : M}^{\gamma} \dashv\vdash \boxed{m : M}^{\gamma} * \boxed{m' : M}^{\gamma}$$

OWN-UPDATE
$$m \rightsquigarrow_M m' * \boxed{m : M}^{\gamma} \vdash \boxed{m' : M}^{\gamma}$$

OWN-VALID
$$\boxed{m : M}^{\gamma} \vdash \checkmark_M(m)$$

OWN-PERSISTENCE
$$m \cdot m = m \vdash \mathsf{persistent}(\boxed{m : M}^{\gamma})$$

Note how, as formalized by the rule OWN-OP and OWN-VALID, the partiality of the operation of the resource algebra is reflecting a notion of separation in the resource algebra. That is, for two elements m and m', $\boxed{m : M}^{\gamma}$ and $\boxed{m' : M}^{\gamma}$ are suitably disjoint, *i.e.*, $\boxed{m : M}^{\gamma} * \boxed{m' : M}^{\gamma} \vdash$ False if and only if $\not\checkmark_M (m \cdot m')$. The rule OWN-OP also tells us how duplicable resources, *i.e.*, elements for which we

Authors' addresses: Amin Timany, timany@cs.au.dk, Aarhus University, Aarhus, Denmark; Armaël Guéneau, armael.gueneau@inria.fr, Université Paris-Saclay, CNRS, ENS Paris-Saclay, Inria, LMF, Paris, France; Lars Birkedal, Birkedal@cs.au.dk, Aarhus University, Aarhus, Denmark.

have $m = m \cdot m$, are precisely those resources whose ownership is persistent: $\mathsf{persistent}(\boxed{m}^\gamma)$.[1] The rule own-alloc allows us to allocate new ghost resources. In fact, as also evidenced by own-op and own-valid, Iris maintains the global invariant that the product of all resources owned at all times is always valid. The name of the freshly allocated resource is drawn from an infinite set A of ghost names. The allocated resource must crucially be a valid element. The rule own-persistence allows us to derive persistence of propositions defined in terms of resources; ownership is persistent if the resource m is *duplicable*, *i.e.*, if $m \cdot m = m$. According to the rule own-update, a resource m can be updated to a resource m' if m can be *frame preservingly updated to* m', written $m \rightsquigarrow_M m'$. The frame preserving update relation is defined as follows:

$$m \rightsquigarrow_M m' \triangleq (\checkmark_M(m) \implies \checkmark_M(m')) \land \forall f.\ \checkmark_M(f \cdot m) \implies \checkmark_M(f \cdot m') \qquad \text{(frame-preserving-update)}$$

This definition is essentially saying that we should show that what we update to is valid element under the assumption that the element we are updating from is valid, and furthermore that whenever an element f is a valid frame for m, *i.e.*, the element f could potentially be the resources owned separately in the system, then f must also be a valid frame for m'. This means that the frame preserving update being the requirement for updating resources in the system does indeed preserve the global invariant of validity maintained at all times that we discussed earlier. Note that if the resource algebra has an identity element, *i.e.*, an element e for which we have $e \cdot m = m$ for all m, then the left conjunct in frame-preserving-update is subsumed by the right conjunct.

The Extension Order. A resource algebra M induce a so-called *extension order*, $\preceq_M$, among their its elements which we will use below in resource algebra constructions.

$$m \preceq_M m' \triangleq m = m' \lor \exists m''.\ m' = m'' \cdot m$$

Whenever $m \preceq_M m'$ holds $\checkmark_M(m')$ implies $\checkmark_M(m)$.

1.1 General Resource Algebra Constructions

The Agreement Resource Algebra. Given a set A, we construct the agreement resource algebra over A, $\textsc{Ag}(A)$, as the resource algebra with the following carrier set:

$$\textsc{Ag}(A) \triangleq \{\mathsf{ag}(a) | a \in A\} \cup \{\bot_{\textsc{Ag}}\}$$

Here all elements are valid except for $\bot_{\textsc{Ag}}$, $\not\checkmark(\bot_{\textsc{Ag}})$. The operation of the agreement resource algebra is defined as follows:

$$m \cdot m' \triangleq \begin{cases} m & \text{if } m = \mathsf{ag}(a) \text{ and } m' = \mathsf{ag}(a') \text{ and } a = a' \\ \bot_{\textsc{Ag}} & \text{otherwise} \end{cases}$$

This resource algebra has no non-trivial frame preserving update.

The Exclusive Resource Algebra. The exclusive resource algebra, $\textsc{Ex}(A)$ where A is a set. The carrier set of this resource algebra is as follows:

$$\textsc{Ex}(A) \triangleq \{\mathsf{ex}(a) | a \in A\} \cup \{\bot_{\textsc{Ex}}\}$$

and we have

$$\checkmark(m) \iff m \neq \bot_{\textsc{Ex}}$$

[1] Our notion of persistence is slightly simplified compared to the official definition in Iris (where all persistent resources are duplicable, but not the other way round); but this simplification is immaterial for what we discuss in this paper.

The operation of the exclusive resource algebra is essentially never defined which means there can never be two separate elements of the resource in the system, hence the name *exclusive*.

$$m \cdot m' \triangleq \bot_{\text{Ex}}$$

Therefore, we have any valid exclusive element ca be updated to any other valid element:

$$\mathsf{ex}(a) \rightsquigarrow \mathsf{ex}(b)$$

In the exclusive resource algebra the extension order is simply the reflexivity relation as the operation always results in $\bot_{\text{Ex}}$.

The Finite Set Resource Algebra. Given a set A, the resource algebra of finite sets, $\textsc{FinSet}(A)$, has as carrier the set of all finite subsets of A. All elements are valid and the operation of this resource is the simply the union of the two sets. Since all elements of this resource algebra are valid any element can be frame preservingly updated to any other element. In the FinSet resource algebra the extension order coincides with the subset relation: $C \preceq_{\textsc{FinSet}(A)} D$ if and only if $C \subseteq D$. This resource algebra is not very interesting on its own but will prove useful in combination with other resource algebra constructions.

The Sum Resource Algebra. Given two resource algebras M_1 and M_2, their sum, $\textsc{Sum}(M_1, M_2)$. The carrier set of this resource algebra is as follows:

$$\textsc{Sum}(M_1, M_2) \triangleq \{\mathsf{injl}(m_1) | m_1 \in M_1\} \cup \{\mathsf{injr}(m_2) | m_2 \in M_2\} \cup \{\bot_{\textsc{Sum}}\}$$

The validity is defined as the validity of the underlying resource algebras:

$$\begin{aligned} \checkmark(\mathsf{injl}(m_1)) &\iff \checkmark(m_1) \\ \checkmark(\mathsf{injr}(m_1)) &\iff \checkmark(m_2) \end{aligned}$$

with $\bot_{\textsc{Sum}}$ being invalid, $\not\checkmark(\bot_{\textsc{Sum}})$. The operation of the sum resource algebra is defined as follows:

$$m \cdot m' \triangleq \begin{cases} \mathsf{injl}(m_1 \cdot m_1') & \text{if } m = \mathsf{injl}(m_1) \text{ and } m' = \mathsf{injl}(m_1') \\ \mathsf{injr}(m_2 \cdot m_2') & \text{if } m = \mathsf{injr}(m_2) \text{ and } m' = \mathsf{injr}(m_2') \\ \bot_{\textsc{Sum}} & \text{otherwise} \end{cases}$$

The frame preserving updates of the sum resource algebra depends on the resource algebras it is instantiated with; we will discuss these when we use this resource algebra later on.

The Authoritative Resource Algebra. The authoritative resource algebra is one of the most important and versatile resource algebras among all resource algebras. Given the resource algebra M we define the authoritative resource algebra $\textsc{Auth}(M)$. The carrier set of $\textsc{Auth}(M)$ is as follows:

$$\textsc{Auth}(M) \triangleq \{\bullet\, m | m \in M\} \cup \{\circ\, m | m \in M\} \cup \{\bullet\!\circ(m, m') | m, m' \in M\} \cup \{\bot_{\textsc{Auth}}\}$$

The idea is the so-called *full* element $\bullet$ is the central authority in the sense that all *fragmental* elements, $\circ$, are included in it — see the definition validity below. Elements of the form $\bullet\!\circ$ are simply there to represent resources elements that include both the central authority together with some fragment of it — see the definition of validity and the operation of resource algebra below.

The operation of the authoritative resource algebra is defined as follows:

$$m_1 \cdot m_2 \triangleq \begin{cases} \circ\,(m_1' \cdot m_2') & \text{if } m_1 = \circ\, m_1' \text{ and } m_2 = \circ\, m_2' \\ \bullet\!\circ(m_1', m_2') & \text{if } m_1 = \bullet\, m_1' \text{ and } m_2 = \circ\, m_2' \\ \bullet\!\circ(m_2', m_1') & \text{if } m_1 = \circ\, m_1' \text{ and } m_2 = \bullet\, m_2' \\ \bullet\!\circ(m_{11}', m_{12}' \cdot m_2') & \text{if } m_1 = \bullet\!\circ(m_{11}', m_{12}') \text{ and } m_2 = \circ\, m_2' \\ \bullet\!\circ(m_{21}', m_1' \cdot m_{22}') & \text{if } m_1 = \circ\, m_1' \text{ and } m_2 = \bullet\!\circ(m_{21}', m_{22}') \\ \bot_{\textsc{Auth}} & \text{otherwise} \end{cases}$$

The validity of elements of the authoritative resource algebra are as follows:

$$\begin{aligned} \checkmark_{\textsc{Auth}(M)}(\bullet\, m') &\iff \checkmark(m') \\ \checkmark_{\textsc{Auth}(M)}(\circ\, m') &\iff \checkmark(m') \\ \checkmark_{\textsc{Auth}(M)}(\bullet\!\circ(m_1, m_2)) &\iff \checkmark(m_1) \land m_2 \preceq_M m_1 \\ \checkmark_{\textsc{Auth}(M)}(\bot_{\textsc{Auth}}) &\iff \mathsf{False} \end{aligned}$$

Note how the operation and the validity relation enforce that there is at most one full part, $\bullet$, for each instance of the authoritative resource algebra. The frame preserving updates of the authoritative resource algebra depends on the resource algebra it is instantiated with; we will discuss these when we use this resource algebra later on.

The One-Shot Resource Algebra. We define the one-shot resource using the resource algebra $\textsc{Sum}(\textsc{Ex}(\{*\}), \textsc{Ag}(\{*\}))$ where $\{*\}$ is an arbitrary but fixed singleton set. The propositions $\mathsf{pending}(\gamma)$ and $\mathsf{shot}(\gamma)$ are defined as follows:

$$\begin{aligned} \mathsf{pending}(\gamma) &\triangleq \boxed{\mathsf{injl}(\mathsf{ex}(*))}^{\gamma} \\ \mathsf{shot}(\gamma) &\triangleq \boxed{\mathsf{injr}(\mathsf{ag}(*))}^{\gamma} \end{aligned}$$

With this definition we can immediately see that the rule PENDING-NOT-SHOT holds as $\mathsf{injl}(\mathsf{ex}(*)) \cdot \mathsf{injr}(\mathsf{ag}(*)) = \bot_{\textsc{Sum}}$ by the definition of the operation of the sum resource algebra. On the other hand, $\mathsf{injl}(\mathsf{ex}(*))$ has no valid frame: the frame can neither be of the form injr nor of the form $\mathsf{injl}(x)$ for any x as $\mathsf{ex}(*)$ is exclusive. Hence, to show the frame preserving update of SHOOT we only need to show $\checkmark(\mathsf{injr}(\mathsf{ag}(*)))$ which trivially holds by definition. The rule OWN-PERSISTENCE allows us to prove SHOT-PERSISTENT as we have $\mathsf{injr}(\mathsf{ag}(*)) \cdot \mathsf{injr}(\mathsf{ag}(*)) = \mathsf{injr}(\mathsf{ag}(*))$. To show MAKE-ONE-SHOT we only need to show that $\checkmark(\mathsf{injl}(\mathsf{ex}(*)))$ which does hold by definition.

The Monotone Resource Algebra. The monotone resource algebra [Timany and Birkedal 2021] is designed for reasoning about monotonicity in separation logic with respect to arbitrary preorder (reflexive and transitive) relations. We will not detail the construction of this resource algebra here. The interested reader is referred to Timany and Birkedal [2021] for the full details. What is relevant here is the following: given a preorder relation (A, R), the monotone resource algebra for R, $\mathit{Monotone}(R)$, can embed the elements of the preorder using the function $\mathit{principal}_R : A \to \mathit{Monotone}(R)$ such that:

$$\mathit{principal}_R(a) \preceq_{\mathit{Monotone}(R)} \mathit{principal}_R(b) \iff R(a, b)$$

This means that in combination with the authoritative resource algebras we obtain:

$$\boxed{\bullet\, \mathit{principal}_R(b)}^{\gamma} * \boxed{\circ\, \mathit{principal}_R(a)}^{\gamma} \vdash R(a, b)$$

and the following frame preserving update whenever $R(a, b)$:

$$\bullet\, \mathit{principal}_R(a) \rightsquigarrow \bullet\, \mathit{principal}_R(b) \cdot \circ\, \mathit{principal}_R(b)$$

1.2 The Definition of the Building Blocks of the Well-Bracketed Program Logic

Each stack resource is constructed as an instance of the resource algebra $\textsc{Auth}(\textsc{Ex}(\mathit{stacks}))$. The predicates $\mathit{stack}^{\mathsf{IN}}_{\bullet}$ and $\mathit{stack}^{\mathsf{IN}}_{\circ}$ are defined as follows (we will write N for the (ghost) name of the stack instead of γ just to keep a familiar notation; stack names are simply ghost names):

$$\mathit{stack}^{\mathsf{IN}}_{\bullet}(N, s) \triangleq \boxed{\bullet \mathsf{ex}(s)}^{N}$$
$$\mathit{stack}^{\mathsf{IN}}_{\circ}(N, s) \triangleq \boxed{\circ \mathsf{ex}(s)}^{N}$$

Since the extension order of the exclusive resource algebra is the reflexivity relation, by using the rules own-op and own-valid, we can derive that

$$\boxed{\bullet \mathsf{ex}(s)}^{N} * \boxed{\circ \mathsf{ex}(s')}^{N} \vdash s = s'$$

which is precisely the rule stacks$^{\mathsf{IN}}$-agree. To prove the rule stacks$^{\mathsf{IN}}$-update, we need to show the following frame preserving update:

$$\bullet\!\circ(\mathsf{ex}(s), \mathsf{ex}(s')) \rightsquigarrow \bullet\!\circ(\mathsf{ex}(s''), \mathsf{ex}(s''))$$

However, we can immediately see that the left hand side of the update above cannot possibly have any valid frames. The frame cannot be a full part. On the other hand, by the definition of the operation of the authoritative and exclusive resource algebras, the frame composed with the left and side would be $\bullet\!\circ(\mathsf{ex}(s), \bot_{\mathrm{Ex}})$ which is not valid as $\bot_{\mathrm{Ex}} \not\preceq \mathsf{ex}(s)$. Hence, we have to show that $\checkmark(\bullet\!\circ(\mathsf{ex}(s''), \mathsf{ex}(s'')))$ which is equivalent to showing that $\checkmark(\mathsf{ex}(s''))$ and that $\mathsf{ex}(s'') \preceq \mathsf{ex}(s'')$ which are both trivial.

What remains of the basic building blocks is the propositions *EntireDom* and *SubsetOfDom*. We define these propositions using the resource algebra $\textsc{Auth}(\textsc{FinSet}(\mathit{stacknames}))$. In the following, we will assume we have a globally fixed ghost name γ_{dom}. The propositions *EntireDom* and *SubsetOfDom* are defined as follows:

$$\mathit{EntireDom}(A) \triangleq \boxed{\bullet A}^{\gamma_{dom}}$$
$$\mathit{SubsetOfDom}(A) \triangleq \boxed{\circ A}^{\gamma_{dom}}$$

These definitions immediately satisfy the rules dom-subset and dom-distributes. The former holds as the left hand side of the rule basically amounts to $\checkmark(\bullet\!\circ(B, A))$ which by definition implies $A \preceq_{\textsc{FinSet}(\mathit{stacknames})} B$, or equivalently, $A \subseteq B$. The latter, on the other hand, holds by the definition of the operation of the authoritative resource algebra for two fragments and a simple application of the rule own-op — recall that the operation of the finite sets resource algebra is set union. To prove the rule dom-grow we only need to show the following frame preserving update:

$$\bullet\!\circ(B, \emptyset) \rightsquigarrow \bullet\!\circ(B \cup A, A)$$

In this case, we can see that the frame cannot be of the form $\bullet\, C$ or $\bullet\!\circ(C, D)$ for any sets C and D. However, it could be of the form $\circ\, C$ for a C such that $\checkmark(\bullet\!\circ(B, \emptyset \cup C))$ which is equivalent to saying that in this case we must have $C \subseteq B$. To conclude the proof we only need to show that $\checkmark(\bullet\!\circ(B \cup A, A \cup C))$. This is indeed the case as $A \cup C \subseteq B \cup A$ and we have that $\checkmark(B \cup A)$ holds trivially as all finite sets are valid in the resource algebra of finite sets and A is required to be finite.

2 ENCODING STATE TRANSITION SYSTEMS USING GHOST STACKS

We define a state transition system with public and private transition relations (STS for short) to be a tuple $(S, \mathit{pub}, \mathit{pri})$ where *pub* is the public transition relation and *pri* is the private transition relation. We stipulate that both transition relations are reflexive and transitive and that the public relation is included in the private relation.

Below, we will construct a theory allowing us to embed state transition systems into our logic based on ghost stacks. The idea is that when the user asks to embed a new STS, we create a ghost stack and an invariant over managing the stack. We use the name of the freshly allocated stack as the name for the STS instance.

We first define configurations which consist of a stack (of ghost names) together with a STS state. We write *configurations* for the set of configurations. The idea is that our logical predicates for embedding the STS into our logic are defined as relations over configurations. We write $StateOf(c)$ for the state of the configuration c and $StackOf(c)$ for its stack. We define the following predicates to embed the STS into our logic:

$$\begin{aligned}
PubRel(c, c') &\triangleq StackOf(c) = StackOf(c') \land pub(StateOf(c), StateOf(c')) \\
PriRel(c, c') &\triangleq \exists \gamma, \gamma', s.\ StackOf(c') = \gamma' :: StackOf(c) * StackOf(c) = \gamma :: s * \\
&\qquad \boxed{\bullet\ principal_{pub}(StateOf(c))}^{\gamma} * pri(StateOf(c), StateOf(c')) \\
OwnConf_{\bullet}(N, c) &\triangleq \exists \gamma, s.\ StackOf(c) = \gamma :: s \land stack_{\circ}(N, StackOf(c)) * \boxed{\bullet\ principal_{pub}(StateOf(c))}^{\gamma} \\
OwnConf_{\circ}(N, c) &\triangleq \exists \gamma, s.\ StackOf(c) = \gamma :: s \land stack_{\bullet}(N, StackOf(c)) * \boxed{\circ\ principal_{pub}(StateOf(c))}^{\gamma}
\end{aligned}$$

The idea is that we use the monotone resource algebra over the public relation (together with the authoritative resource algebra) to construct a resource algebra that can only be updated according to the public relation. Notice how the *PubRel* captures our intuitive idea of a public transition: the stack has not changed and the states are related publicly. The *PriRel* predicate on the other hand requires the stack to have grown (we have pushed the new name on top of the stack) but crucially, we own the full part of the resource algebra tracking the state of the STS before the private transition. It is necessary to won the full part of the old instance so we can revert the private transition by popping the stack.

to enable reasoning about well-bracketedness, we use the following invariant $STSInv(N, \Phi)$, parameterized by the stack name and a predicate Φ over the states of the STS. This is the invariant that is established when we allocate the stack corresponding to the STS.

$$STSInv(N, \Phi) \triangleq \boxed{\exists c.\ OwnConf_{\bullet}(N, c) * \Phi(StateOf(c))}^{\mathcal{N}_{STS}.N}$$

The core rules for using the encoding of STSs are as follows:

wbHoare-create-STS
$$\Phi(a) * stack_{\bullet}(N, []) * stack_{\circ}(N, []) \vdash \Rrightarrow STSInv(N, \Phi) * \exists s.\ stack_{\bullet}(N, s)$$

wbHoare-access-STS
$$\frac{N \notin O \qquad \left(\forall c.\ \left(\!\left| P * OwnConf_{\circ}(N, c) \right|\!\right)\ e\ \left(\!\left| x.\ \Phi(x) * \exists c'.\ OwnConf_{\circ}(N, c') * PubRel(c, c') \right|\!\right)^{O \cup \{N\}}\right)}{\left(\!\left| STSInv(N, \Phi) * P \right|\!\right)\ e\ \left(\!\left| \Phi \right|\!\right)^{O}}$$

wbHoare-mend-STS
$$\left(\!\left| P \right|\!\right)\ e\ \left(\!\left| \Phi \right|\!\right)^{O \setminus \{N\}} \vdash \left(\!\left| OwnConf_{\circ}(N, c) * P \right|\!\right)\ e\ \left(\!\left| x.\ \Phi(x) * OwnConf_{\circ}(N, c) \right|\!\right)^{O}$$

Note how the rule wbHoare-create-STS precisely matches what is required in the rule wbHoare-create-stack. That is, wbHoare-create-STS can be used in conjunction with wbHoare-create-stack to create a stack and immediately establish *STSInv*. The rules wbHoare-access-STS and wbHoare-mend-STS directly correspond to wbHoare-access-stack and wbHoare-mend-stack.

The predicates for embedding STSs satisfy the following rules:

PUB-RELATED-TRANSITIVE
$$PubRel(c, c') * PubRel(c', c'') \vdash PubRel(c, c'')$$

PUB-PRI-RELATED-TRANSITIVE
$$PriRel(c, c') * PubRel(c', c'') \vdash PriRel(c, c'')$$

PUB-RELATED-SOUND
$$PubRel(c, c') \vdash pub(StateOf(c), StateOf(c'))$$

MAKE-PUB-TRANSITION
$$\frac{pub(StateOf(c), a')}{OwnConf_\bullet(N, c) * OwnConf_\circ(N, c) \vdash \Rrightarrow \exists c'.\ OwnConf_\bullet(N, c') * OwnConf_\circ(N, c') * PubRel(c, c') * StateOf(c') = a'}$$

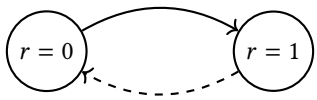
MAKE-PRI-TRANSITION
$$\frac{pri(StateOf(c), a')}{OwnConf_\bullet(N, c) * OwnConf_\circ(N, c) \vdash \Rrightarrow \exists c'.\ OwnConf_\bullet(N, c') * OwnConf_\circ(N, c') * PriRel(c, c') * StateOf(c') = a'}$$

UNDO-PRI-TRANSITION
$$PriRel(c, c') * OwnConf_\bullet(N, c') * OwnConf_\circ(N, c') \vdash \Rrightarrow OwnConf_\bullet(N, c) * OwnConf_\circ(N, c)$$

CONFIS-UPDATE-FRAG
$$OwnConf_\bullet(N, c) * OwnConf_\circ(N, c') \vdash \Rrightarrow OwnConf_\bullet(N, c) * OwnConf_\circ(N, c)$$

CONFIS-PUB-RELATED
$$OwnConf_\bullet(N, c) * OwnConf_\circ(N, c') \vdash PubRel(c', c)$$

We use the theory presented in this section to prove correctness of VAE. We use the STS that we pick for this purpose is the following:



(VAE-sts)

The predicate Φ that we pick for the invariant *STSInv* is the following:

$$\Phi(a) \triangleq \ell \mapsto a$$

where ℓ is the location allocated by VAE. See our Coq development for the details of the proof.

3 SOME RULES AND DEFINITIONS FOR REFERENCE

All of the rules and definitions below are taken verbatim from the main paper.

3.1 Rules connecting well-bracketed Hoare triples and ghost stacks

WBHOARE-CREATE-STACK
$$\frac{\forall N.\ stack_\bullet(N, []) * stack_\circ(N, []) \equiv\!\!* R(N) * \exists s.\ stack_\bullet(N, s) \qquad \forall N.\ \left(\!\left|P * R(N)\right|\!\right) e \left(\!\left|\Phi\right|\!\right)^O}{\left(\!\left|P\right|\!\right) e \left(\!\left|\Phi\right|\!\right)^O}$$

WBHOARE-ACCESS-STACK
$$\frac{N \notin O}{\left(\forall s.\ \left(\!\left|P * stack_\bullet(N, s)\right|\!\right) e \left(\!\left|x.\ \Phi(x) * stack_\bullet(N, s)\right|\!\right)^{O \cup \{N\}}\right) \vdash \left(\!\left|stack_\exists(N) * P\right|\!\right) e \left(\!\left|\Phi\right|\!\right)^O}$$

WBHOARE-MEND-STACK
$$\left(\!\left|P\right|\!\right) e \left(\!\left|\Phi\right|\!\right)^{O \setminus \{N\}} \vdash \left(\!\left|stack_\bullet(N, s) * P\right|\!\right) e \left(\!\left|x.\ \Phi(x) * stack_\bullet(N, s)\right|\!\right)^O$$

HOARE-WBHOARE
$$\left\{P\right\} e \left\{\Phi\right\} \vdash \left(\!\left|P\right|\!\right) e \left(\!\left|\Phi\right|\!\right)^O$$

3.2 Rules of the One-Shot Ghost Theory

MAKE-ONE-SHOT
$\vdash \Rrightarrow \exists \gamma.\ \mathsf{pending}(\gamma)$

SHOOT
$\mathsf{pending}(\gamma) \vdash \Rrightarrow \mathsf{shot}(\gamma)$

PENDING-NOT-SHOT
$\mathsf{pending}(\gamma) * \mathsf{shot}(\gamma) \vdash \mathsf{False}$

SHOT-PERSISTENT
$\vdash \mathsf{persistent}(\mathsf{shot}(\gamma))$

3.3 Rules Governing Stacks

STACKS-AGREE
$stack_\bullet(N, s) * stack_\circ(N, s') \vdash s = s'$

STACK-FULL-UNIQUE
$stack_\bullet(N, s) * stack_\bullet(N, s') \vdash \mathsf{False}$

STACK-FRAGMENT-UNIQUE
$stack_\circ(N, s) * stack_\circ(N, s') \vdash \mathsf{False}$

STACK-EXISTS
$stack_\circ(N, s) \vdash stack_\exists(N)$

STACKS-PUSH
$stack_\bullet(N, s) * stack_\circ(N, s) \vdash \Rrightarrow stack_\bullet(N, \gamma :: s) * stack_\circ(N, \gamma :: s)$

STACKS-POP
$stack_\bullet(N, \gamma :: s) * stack_\circ(N, \gamma :: s) \vdash \Rrightarrow stack_\bullet(N, s) * stack_\circ(N, s)$

3.4 Rules Governing AllStacksExcept

MASK-SUBSET-DOM
$\mathsf{AllStacksExcept}(\mathcal{S}, \mathcal{O}) \vdash \mathcal{O} \subseteq \mathrm{dom}(\mathcal{S})$

STACK-EXISTS-IN
$stack_\exists(N) * \mathsf{AllStacksExcept}(\mathcal{S}, \mathcal{O}) \vdash N \in \mathrm{dom}(\mathcal{S})$

STACK-FRAG-NOT-OUT
$$\frac{N \notin \mathcal{O}}{stack_\circ(N, s) * \mathsf{AllStacksExcept}(\mathcal{S}, \mathcal{O}) \vdash \mathcal{S}(N) = s}$$

STACK-FULL-IS-OUT
$stack_\bullet(N, s) * \mathsf{AllStacksExcept}(\mathcal{S}, \mathcal{O}) \vdash N \in \mathcal{O}$

STACK-TAKE-OUT
$$\frac{N \in \mathrm{dom}(\mathcal{S}) \setminus \mathcal{O}}{\mathsf{AllStacksExcept}(\mathcal{S}, \mathcal{O}) \vdash \exists s.\ \mathsf{AllStacksExcept}(\mathcal{S}, \mathcal{O} \cup \{N\}) * stack_\bullet(N, s)}$$

STACK-PUT-BACK
$$\frac{\mathcal{S}(N) = s}{stack_\bullet(N, s) * \mathsf{AllStacksExcept}(\mathcal{S}, \mathcal{O}) \vdash \mathsf{AllStacksExcept}(\mathcal{S}, \mathcal{O} \setminus \{N\})}$$

CHANGE-OUT-STACK
$$\frac{N \in \mathcal{O}}{\mathsf{AllStacksExcept}(\mathcal{S}, \mathcal{O}) \vdash \mathsf{AllStacksExcept}(\mathcal{S}[N \mapsto s'], \mathcal{O})}$$

CREATE-STACK
$\mathsf{AllStacksExcept}(\mathcal{S}, \mathcal{O}) \vdash \Rrightarrow \exists N.\ N \notin \mathrm{dom}(\mathcal{S}) * \mathsf{AllStacksExcept}(\mathcal{S}[N \mapsto [\gamma]], \mathcal{O}) * stack_\circ(N, [\gamma])$

3.5 The Definition the Predicates of the Theory of Stack Collections

$$\mathit{stack}_\bullet(N,s) \triangleq \mathit{stack}_\bullet^{\mathsf{IN}}(N,s) * \mathit{SubsetOfDom}(\{N\})$$
$$\mathit{stack}_\circ(N,s) \triangleq \mathit{stack}_\circ^{\mathsf{IN}}(N,s) * \mathit{SubsetOfDom}(\{N\})$$
$$\mathit{stack}_\exists(N) \triangleq \mathit{SubsetOfDom}(\{N\})$$
$$\mathsf{AllStacksExcept}(\mathcal{S},O) \triangleq O \subseteq \mathrm{dom}(\mathcal{S}) * \mathit{EntireDom}(\mathrm{dom}(\mathcal{S})) * \mathop{\Large *}_{N \in \mathrm{dom}(\mathcal{S}) \setminus O} \mathit{stack}_\bullet(N, \mathcal{S}(N))$$

3.6 Rules Governing the Predicates of the Theory of Stack Collections

$$\mathit{SubsetOfDom}(A) * \mathit{EntireDom}(B) \vdash A \subseteq B \qquad \text{(dom-subset)}$$
$$\mathit{SubsetOfDom}(A) * \mathit{SubsetOfDom}(B) \dashv\vdash \mathit{SubsetOfDom}(A \cup B) \qquad \text{(dom-distributes)}$$
$$\mathit{EntireDom}(B) * \mathsf{finite}(A) \vdash \Rrightarrow \mathit{EntireDom}(A \cup B) * \mathit{SubsetOfDom}(A) \qquad \text{(dom-grow)}$$

$$\mathit{stack}_\bullet^{\mathsf{IN}}(N,s) * \mathit{stack}_\circ^{\mathsf{IN}}(N,s') \vdash s = s' \qquad (\mathsf{stacks}^{\mathsf{IN}}\text{-agree})$$
$$\mathit{stack}_\bullet^{\mathsf{IN}}(N,s) * \mathit{stack}_\bullet^{\mathsf{IN}}(N,s') \vdash \mathsf{False} \qquad (\mathsf{stacks}_\bullet{}^{\mathsf{IN}}\text{-unique})$$
$$\mathit{stack}_\circ^{\mathsf{IN}}(N,s) * \mathit{stack}_\circ^{\mathsf{IN}}(N,s') \vdash \mathsf{False} \qquad (\mathsf{stacks}_\circ{}^{\mathsf{IN}}\text{-unique})$$
$$\mathit{stack}_\bullet^{\mathsf{IN}}(N,s) * \mathit{stack}_\circ^{\mathsf{IN}}(N,s') \vdash \Rrightarrow \mathit{stack}_\bullet^{\mathsf{IN}}(N,s'') * \mathit{stack}_\circ^{\mathsf{IN}}(N,s'') \qquad (\mathsf{stacks}^{\mathsf{IN}}\text{-update})$$
$$\mathsf{infinite}(A) \vdash \Rrightarrow \exists N \in A.\ \mathit{stack}_\bullet^{\mathsf{IN}}(N,s) * \mathit{stack}_\circ^{\mathsf{IN}}(N,s) \qquad (\mathsf{stacks}^{\mathsf{IN}}\text{-create})$$

REFERENCES

Ralf Jung, Robbert Krebbers, Jacques-Henri Jourdan, Aleš Bizjak, Lars Birkedal, and Derek Dreyer. 2018. Iris from the ground up: A modular foundation for higher-order concurrent separation logic. *Journal of Functional Programming* 28 (2018), e20. https://doi.org/10.1017/S0956796818000151

Amin Timany and Lars Birkedal. 2021. Reasoning about monotonicity in separation logic. In *CPP '21: 10th ACM SIGPLAN International Conference on Certified Programs and Proofs, Virtual Event, Denmark, January 17-19, 2021*, Catalin Hritcu and Andrei Popescu (Eds.). ACM, 91–104. https://doi.org/10.1145/3437992.3439931