Relational Models and Program Logics: Logical Relations in Iris

Lars Birkedal
Aaarhus University

Joint work with Amin Timany and Robbert Krebbers

October 2017, Shonan, Japan

Relational Models and Program Logics

- Wish to reason both about unary and relational properties.
  - relational properties include: contextual refinement, non-interference, compiler correctness
  - unary properties include ordinary specifications (Hoare triples)
- Wish to reason about properties of individual programs and also language properties
  - language properties include type safety or type-based program equivalences
- In this talk: we will see how the Iris program logic can also be used to reason about unary and relational language properties.

Logical Relations

A powerful technique to prove language properties

- Unary: **type safety**, (strong) normalization, ...
- Binary: **contextual refinement**, contextual equivalence, non-interference, ...

In this talk

- Formalization of a unary and binary logical relations
- In the Iris program logic which in turn is implemented in Coq
- For a programming language ($\mathrm{F}_{\mu,\mathit{ref},\mathit{conc}}$) with a very rich type system
- Use it to prove type safety and verify contextual refinement of concurrent algorithms

Unary logical relation

Proving type safety

Define *semantics* of types (by recursion on τ): $[\![\tau]\!]^{\mathcal{E}} : \mathit{Expr} \to \mathrm{Prop}$

Unary logical relation

Proving type safety

Define *semantics* of types (by recursion on τ): $[\![\tau]\!]^{\mathcal{E}} : \mathit{Expr} \to \mathrm{Prop}$

1. Prove *adequacy*: $[\![\tau]\!]^{\mathcal{E}}(e) \Rightarrow \mathit{Safe}_{\tau}(e)$

Unary logical relation

Proving type safety

Define *semantics* of types (by recursion on τ): $[\![\tau]\!]^{\mathcal{E}} : \mathit{Expr} \to \mathrm{Prop}$

1. Prove *adequacy*: $[\![\tau]\!]^{\mathcal{E}}(e) \Rightarrow \mathit{Safe}_{\tau}(e)$
2. Prove *compatibility lemmas* for typing rules, e.g.:

$$\frac{[\![\tau_1]\!]^{\mathcal{E}}(e_1) \qquad [\![\tau]\!]^{\mathcal{E}}(e_2)}{[\![\tau_1 \times \tau_2]\!]^{\mathcal{E}}(e_1, e_1)}$$

Unary logical relation

Proving type safety

Define *semantics* of types (by recursion on τ): $[\![\tau]\!]^{\mathcal{E}} : \mathit{Expr} \to \mathrm{Prop}$

1. Prove *adequacy*: $[\![\tau]\!]^{\mathcal{E}}(e) \Rightarrow \mathit{Safe}_{\tau}(e)$
2. Prove *compatibility lemmas* for typing rules, e.g.:

$$\frac{[\![\tau_1]\!]^{\mathcal{E}}(e_1) \qquad [\![\tau]\!]^{\mathcal{E}}(e_2)}{[\![\tau_1 \times \tau_2]\!]^{\mathcal{E}}(e_1, e_1)}$$

3. Corollary (*soundness*): $\cdot \vdash e : \tau \Rightarrow \mathit{Safe}_{\tau}(e)$

Unary logical relation

Proving type safety

Remember adequacy: $[\![\tau]\!]^{\mathcal{E}}(e) \Rightarrow \mathit{Safe}_{\tau}(e)$

- e need not syntactically be well-typed!

Unary logical relation

Proving type safety

Remember adequacy: $[\![\tau]\!]^{\mathcal{E}}(e) \Rightarrow \mathit{Safe}_{\tau}(e)$

- e need not syntactically be well-typed!
- *Compatibility lemmas* say nothing about well-typedness:

$$\frac{[\![\tau_1]\!]^{\mathcal{E}}(e_1) \qquad [\![\tau]\!]^{\mathcal{E}}(e_2)}{[\![\tau_1 \times \tau_2]\!]^{\mathcal{E}}(e_1, e_1)}$$

Unary logical relation

Proving type safety

Remember adequacy: $[\![\tau]\!]^{\mathcal{E}}(e) \Rightarrow \mathit{Safe}_{\tau}(e)$

- e need not syntactically be well-typed!
- *Compatibility lemmas* say nothing about well-typedness:

$$\frac{[\![\tau_1]\!]^{\mathcal{E}}(e_1) \qquad [\![\tau]\!]^{\mathcal{E}}(e_2)}{[\![\tau_1 \times \tau_2]\!]^{\mathcal{E}}(e_1, e_1)}$$

Logical relation allows one to prove safety modularly in the presence of untyped code, i.e., when linking with *untyped* but *verified code*, e.g., the *unsafe blocks* of Rust

Motivation for using a high-level logic (Iris)

Recursive types and higher order references

- **Recursive types:** the logical relation usually involves step-indexing

  The *crux* of the matter: semantics of a recursive type $\mu X.\,\tau$ is the fixed point of the semantics of τ
- **References:** the logical relation usually involves step-indexing and possible worlds

  The *crux* of the matter: a memory location is of type $\texttt{ref}(\tau)$ if the value stored in it is in the semantics of type τ at *all times* (it is an invariant!)
- **Iris** provides support for invariants and guarded fixed points

Unary logical relation (for type safety) for $F_{\mu,\mathrm{ref},\mathrm{conc}}$

Definition in Iris

$$[\![\tau]\!]^{\mathcal{E}}_{\Delta}(e) \triangleq \{\mathsf{True}\}\, e\, \{w.\, [\![\tau]\!]_{\Delta}(w)\}$$

Unary logical relation (for type safety) for $F_{\mu,\mathrm{ref},\mathrm{conc}}$

Definition in Iris

$$[\![\tau]\!]^{\mathcal{E}}_{\Delta}(e) \triangleq \{\mathsf{True}\}\, e\, \{w.\, [\![\tau]\!]_{\Delta}(w)\}$$

$$[\![\mathbb{N}]\!]_{\Delta}(v) \triangleq v \in \mathbb{N}$$

Unary logical relation (for type safety) for $F_{\mu,\mathrm{ref},\mathrm{conc}}$

Definition in Iris

$$[\![\tau]\!]^{\mathcal{E}}_{\Delta}(e) \triangleq \{\mathsf{True}\}\, e\, \{w.\, [\![\tau]\!]_{\Delta}(w)\}$$

$$[\![\mathbb{N}]\!]_{\Delta}(v) \triangleq v \in \mathbb{N}$$

$$[\![\tau_1 \times \tau_2]\!]_{\Delta}(v) \triangleq \exists v_1, v_2.\, v = (v_1, v_2) \wedge [\![\tau_1]\!]_{\Delta}(v_1) \wedge [\![\tau_2]\!]_{\Delta}(v_2)$$

Unary logical relation (for type safety) for $F_{\mu,\mathrm{ref},\mathrm{conc}}$

Definition in Iris

$$\llbracket \tau \rrbracket^{\mathcal{E}}_{\Delta}(e) \triangleq \{\mathsf{True}\}\, e\, \{w.\, \llbracket \tau \rrbracket_{\Delta}(w)\}$$

$$\llbracket \mathbb{N} \rrbracket_{\Delta}(v) \triangleq v \in \mathbb{N}$$

$$\llbracket \tau_1 \times \tau_2 \rrbracket_{\Delta}(v) \triangleq \exists v_1, v_2.\, v = (v_1, v_2) \wedge \llbracket \tau_1 \rrbracket_{\Delta}(v_1) \wedge \llbracket \tau_2 \rrbracket_{\Delta}(v_2)$$

$$\llbracket \tau_1 \to \tau_2 \rrbracket_{\Delta}(v) \triangleq \forall v'.\, \{\llbracket \tau_1 \rrbracket_{\Delta}(v')\}\, v\ v'\, \{w.\, \llbracket \tau_2 \rrbracket_{\Delta}(w)\}$$

Unary logical relation (for type safety) for $F_{\mu,\mathrm{ref},\mathrm{conc}}$

Definition in Iris

$$\llbracket \tau \rrbracket^{\mathcal{E}}_{\Delta}(e) \triangleq \{\mathsf{True}\}\, e\, \{w.\, \llbracket \tau \rrbracket_{\Delta}(w)\}$$

$$\llbracket \mathbb{N} \rrbracket_{\Delta}(v) \triangleq v \in \mathbb{N}$$

$$\llbracket \tau_1 \times \tau_2 \rrbracket_{\Delta}(v) \triangleq \exists v_1, v_2.\, v = (v_1, v_2) \wedge \llbracket \tau_1 \rrbracket_{\Delta}(v_1) \wedge \llbracket \tau_2 \rrbracket_{\Delta}(v_2)$$

$$\llbracket \tau_1 \to \tau_2 \rrbracket_{\Delta}(v) \triangleq \forall v'.\, \{\llbracket \tau_1 \rrbracket_{\Delta}(v')\}\, v\ v'\, \{w.\, \llbracket \tau_2 \rrbracket_{\Delta}(w)\}$$

$$\llbracket \mu X.\, \tau \rrbracket_{\Delta}(v) \triangleq \mu f.\, \exists w.\, v = \mathtt{fold}\, w \wedge \triangleright \llbracket \tau \rrbracket_{\Delta[X \mapsto f]}(w)$$

$$\llbracket X \rrbracket_{\Delta}(v) \triangleq \Delta(X)(v)$$

Unary logical relation (for type safety) for $F_{\mu,\mathrm{ref},\mathrm{conc}}$

Definition in Iris

$$[\![\tau]\!]^{\mathcal{E}}_{\Delta}(e) \triangleq \{\mathsf{True}\}\, e\, \{w.\, [\![\tau]\!]_{\Delta}(w)\}$$

$$[\![\mathbb{N}]\!]_{\Delta}(v) \triangleq v \in \mathbb{N}$$

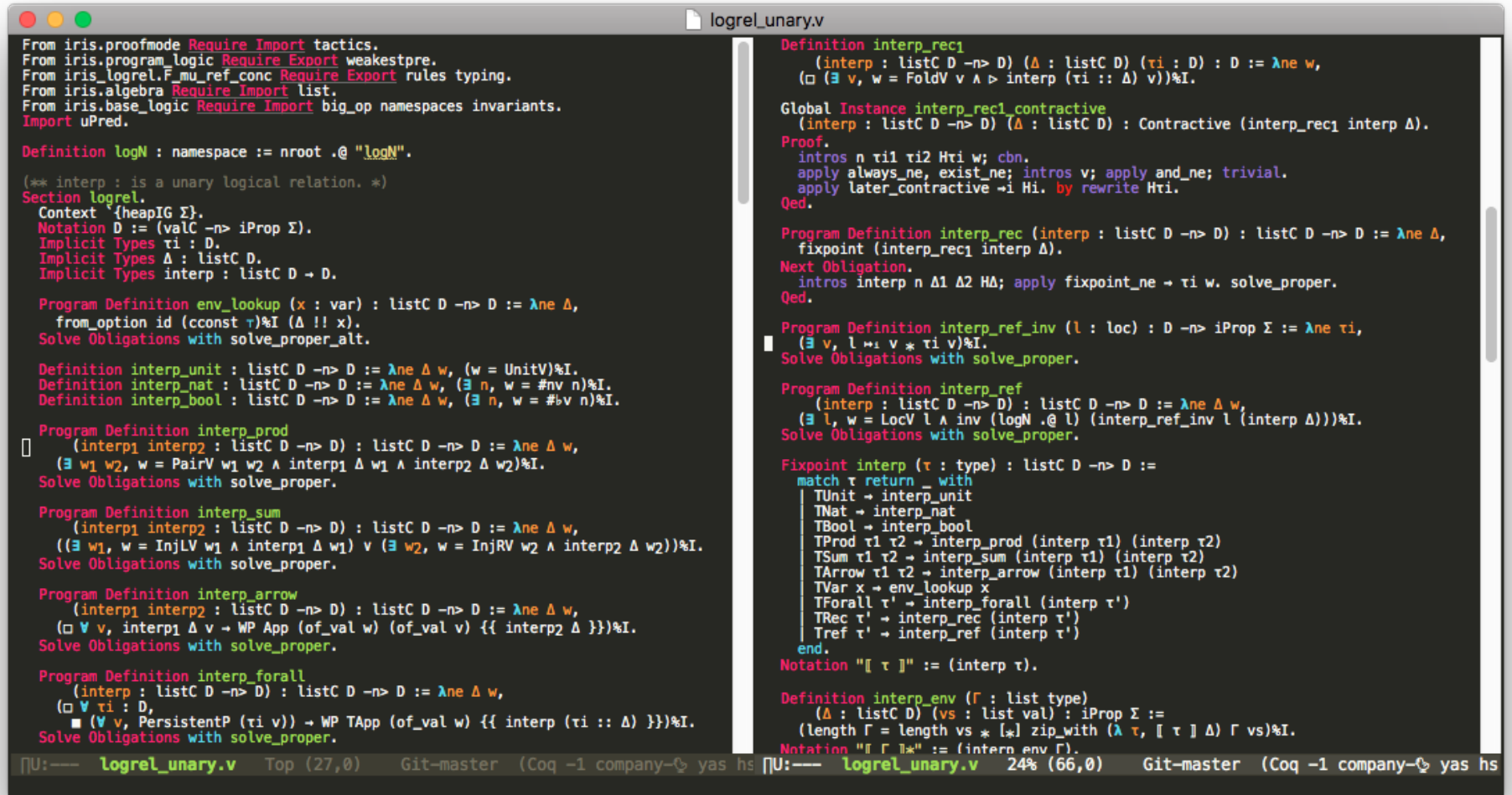
$$[\![\tau_1 \times \tau_2]\!]_{\Delta}(v) \triangleq \exists v_1, v_2.\, v = (v_1, v_2) \land [\![\tau_1]\!]_{\Delta}(v_1) \land [\![\tau_2]\!]_{\Delta}(v_2)$$

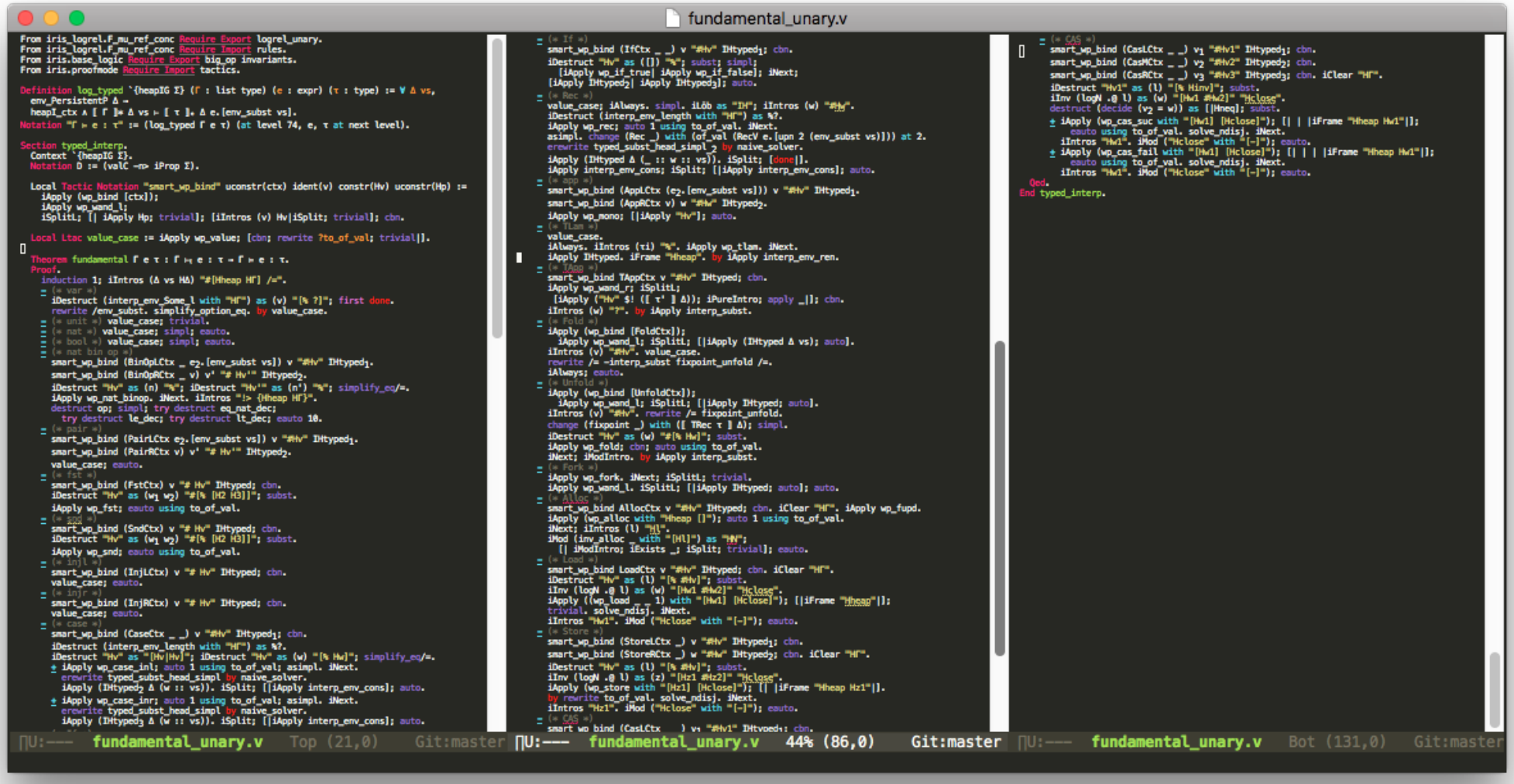
$$[\![\tau_1 \to \tau_2]\!]_{\Delta}(v) \triangleq \forall v'.\, \{[\![\tau_1]\!]_{\Delta}(v')\}\, v\ v'\, \{w.\, [\![\tau_2]\!]_{\Delta}(w)\}$$

$$[\![\mu X.\, \tau]\!]_{\Delta}(v) \triangleq \mu f.\, \exists w.\, v = \mathtt{fold}\, w \land \triangleright [\![\tau]\!]_{\Delta[X \mapsto f]}(w)$$

$$[\![X]\!]_{\Delta}(v) \triangleq \Delta(X)(v)$$

$$[\![\mathtt{ref}(\tau)]\!]_{\Delta}(v) \triangleq \exists \ell.\, v = \ell \land \boxed{\exists w.\, \ell \mapsto w * [\![\tau]\!]_{\Delta}(w)}^{\mathcal{N}.\ell}$$


logrel_unary.v
From iris.proofmode Require Import tactics.
From iris.program_logic Require Export weakestpre.
From iris_logrel.F_mu_ref_conc Require Export rules typing.
From iris.algebra Require Import list.
From iris.base_logic Require Import big_op namespaces invariants.
Import uPred.
Definition logN : namespace := nroot .@ "logN".
(** interp : is a unary logical relation. *)
Section logrel.
Context `{heapIG Σ}.
Notation D := (valC -n> iPropΣ).
Implicit Types τi : D.
Implicit Types Δ : listC D.
Implicit Types interp : listC D → D.
Program Definition env_lookup (x : var) : listC D -n> D := λne Δ,
from_option id (cconst τ)%I (Δ !! x).
Solve Obligations with solve_proper_alt.
Definition interp_unit : listC D -n> D := λne Δ w, (w = UnitV)%I.
Definition interp_nat : listC D -n> D := λne Δ w, (∃ n, w = #nv n)%I.
Definition interp_bool : listC D -n> D := λne Δ w, (∃ n, w = #♭v n)%I.
Program Definition interp_prod
(interp1 interp2 : listC D -n> D) : listC D -n> D := λne Δ w,
(∃ w1 w2, w = PairV w1 w2 ∧ interp1 Δ w1 ∧ interp2 Δ w2)%I.
Solve Obligations with solve_proper.
Program Definition interp_sum
(interp1 interp2 : listC D -n> D) : listC D -n> D := λne Δ w,
((∃ w1, w = InjLV w1 ∧ interp1 Δ w1) ∨ (∃ w2, w = InjRV w2 ∧ interp2 Δ w2))%I.
Solve Obligations with solve_proper.
Program Definition interp_arrow
(interp1 interp2 : listC D -n> D) : listC D -n> D := λne Δ w,
(□ ∀ v, interp1 Δ v → WP App (of_val w) (of_val v) {{ interp2 Δ }})%I.
Solve Obligations with solve_proper.
Program Definition interp_forall
(interp : listC D -n> D) : listC D -n> D := λne Δ w,
(□ ∀ τi : D,
■ (∀ v, PersistentP (τi v)) → WP TApp (of_val w) {{ interp (τi :: Δ) }})%I.
Solve Obligations with solve_proper.
-U:--- logrel_unary.v Top (27,0) Git-master (Coq -1 company-◔ yas hs
Definition interp_rec1
(interp : listC D -n> D) (Δ : listC D) (τi : D) : D := λne w,
(□ (∃ v, w = FoldV v ∧ ▷ interp (τi :: Δ) v))%I.
Global Instance interp_rec1_contractive
(interp : listC D -n> D) (Δ : listC D) : Contractive (interp_rec1 interp Δ).
Proof.
intros n τi1 τi2 Hτi w; cbn.
apply always_ne, exist_ne; intros v; apply and_ne; trivial.
apply later_contractive →i Hi. by rewrite Hτi.
Qed.
Program Definition interp_rec (interp : listC D -n> D) : listC D -n> D := λne Δ,
fixpoint (interp_rec1 interp Δ).
Next Obligation.
intros interp n Δ1 Δ2 HΔ; apply fixpoint_ne → τi w. solve_proper.
Qed.
Program Definition interp_ref_inv (l : loc) : D -n> iPropΣ := λne τi,
(∃ v, l ↦ᵢ v ∗ τi v)%I.
Solve Obligations with solve_proper.
Program Definition interp_ref
(interp : listC D -n> D) : listC D -n> D := λne Δ w,
(∃ l, w = LocV l ∧ inv (logN .@ l) (interp_ref_inv l (interp Δ)))%I.
Solve Obligations with solve_proper.
Fixpoint interp (τ : type) : listC D -n> D :=
match τ return _ with
| TUnit → interp_unit
| TNat → interp_nat
| TBool → interp_bool
| TProd τ1 τ2 → interp_prod (interp τ1) (interp τ2)
| TSum τ1 τ2 → interp_sum (interp τ1) (interp τ2)
| TArrow τ1 τ2 → interp_arrow (interp τ1) (interp τ2)
| TVar x → env_lookup x
| TForall τ' → interp_forall (interp τ')
| TRec τ' → interp_rec (interp τ')
| Tref τ' → interp_ref (interp τ')
end.
Notation "⟦ τ ⟧" := (interp τ).
Definition interp_env (Γ : list type)
(Δ : listC D) (vs : list val) : iPropΣ :=
(length Γ = length vs ∗ [∗] zip_with (λ τ, ⟦ τ ⟧ Δ) Γ vs)%I.
Notation "⟦ Γ ⟧*" := (interp_env Γ).
-U:--- logrel_unary.v 24% (66,0) Git-master (Coq -1 company-◔ yas hs


fundamental_unary.v
From iris_logrel.F_mu_ref_conc Require Export logrel_unary.
From iris_logrel.F_mu_ref_conc Require Import rules.
From iris.base_logic Require Export big_op invariants.
From iris.proofmode Require Import tactics.
Definition log_typed `{heapIG Σ} (Γ : list type) (e : expr) (τ : type) := ∀ Δ vs,
env_Persistent⁇ Δ →
heapI_ctx ∧ ⟦ Γ ⟧* Δ vs ⊢ ⟦ τ ⟧ₑ Δ e.[env_subst vs].
Notation "Γ ⊨ e : τ" := (log_typed Γ e τ) (at level 74, e, τ at next level).
Section typed_interp.
Context `{heapIG Σ}.
Notation D := (valC -n> iProp Σ).
Local Tactic Notation "smart_wp_bind" uconstr(ctx) ident(v) constr(Hv) uconstr(Hp) :=
iApply (wp_bind [ctx]);
iApply wp_wand_l;
iSplitL; [| iApply Hp; trivial]; [iIntros (v) Hv|iSplit; trivial]; cbn.
Local Ltac value_case := iApply wp_value; [cbn; rewrite ?to_of_val; trivial|].
Theorem fundamental Γ e τ : Γ ⊨ e : τ → Γ ⊨ e : τ.
Proof.
induction 1; iIntros (Δ vs HΔ) "#[Hheap HΓ] /=".
(* var *)
iDestruct (interp_env_Some_l with "HΓ") as (v) "[% ?]"; first done.
rewrite /env_subst. simplify_option_eq. by value_case.
(* unit *) value_case; trivial.
(* nat *) value_case; simpl; eauto.
(* bool *) value_case; simpl; eauto.
(* nat bin op *)
smart_wp_bind (BinOpLCtx _ e2.[env_subst vs]) v "#Hv" IHtyped1.
smart_wp_bind (BinOpRCtx _ v) v' "# Hv'" IHtyped2.
iDestruct "Hv" as (n) "%"; iDestruct "Hv'" as (n') "%"; simplify_eq/=.
iApply wp_nat_binop. iNext. iIntros "!> [Hheap HΓ]".
destruct op; simpl; try destruct eq_nat_dec;
try destruct le_dec; try destruct lt_dec; eauto 10.
(* pair *)
smart_wp_bind (PairLCtx e2.[env_subst vs]) v "#Hv" IHtyped1.
smart_wp_bind (PairRCtx v) v' "# Hv'" IHtyped2.
value_case; eauto.
(* fst *)
smart_wp_bind (FstCtx) v "# Hv" IHtyped; cbn.
iDestruct "Hv" as (w1 w2) "#[% [H2 H3]]"; subst.
iApply wp_fst; eauto using to_of_val.
(* snd *)
smart_wp_bind (SndCtx) v "# Hv" IHtyped; cbn.
iDestruct "Hv" as (w1 w2) "#[% [H2 H3]]"; subst.
iApply wp_snd; eauto using to_of_val.
(* inj1 *)
smart_wp_bind (InjLCtx) v "# Hv" IHtyped; cbn.
value_case; eauto.
(* inj2 *)
smart_wp_bind (InjRCtx) v "# Hv" IHtyped; cbn.
value_case; eauto.
(* case *)
smart_wp_bind (CaseCtx _ _) v "#Hv" IHtyped1; cbn.
iDestruct (interp_env_length with "HΓ") as %?.
iDestruct "Hv" as "[Hv|Hv]"; iDestruct "Hv" as (w) "[% Hw]"; simplify_eq/=.
+ iApply wp_case_inl; auto 1 using to_of_val; asimpl. iNext.
erewrite typed_subst_head_simpl by naive_solver.
iApply (IHtyped2 Δ (w :: vs)). iSplit; [|iApply interp_env_cons]; auto.
+ iApply wp_case_inr; auto 1 using to_of_val; asimpl. iNext.
erewrite typed_subst_head_simpl by naive_solver.
iApply (IHtyped3 Δ (w :: vs)). iSplit; [|iApply interp_env_cons]; auto.
(* If *)
smart_wp_bind (IfCtx _ _) v "#Hv" IHtyped1; cbn.
iDestruct "Hv" as ([]) "%"; subst; simpl;
[iApply wp_if_true| iApply wp_if_false]; iNext;
[iApply IHtyped2| iApply IHtyped3]; auto.
(* Rec *)
value_case; iAlways. simpl. iLöb as "IH"; iIntros (w) "#Hw".
iDestruct (interp_env_length with "HΓ") as %?.
iApply wp_rec; auto 1 using to_of_val. iNext.
asimpl. change (Rec _) with (of_val (RecV e.[upn 2 (env_subst vs)])) at 2.
erewrite typed_subst_head_simpl_2 by naive_solver.
iApply (IHtyped Δ (_ :: w :: vs)). iSplit; [done|].
iApply interp_env_cons; iSplit; [|iApply interp_env_cons]; auto.
(* app *)
smart_wp_bind (AppLCtx (e2.[env_subst vs])) v "#Hv" IHtyped1.
smart_wp_bind (AppRCtx v) w "#Hw" IHtyped2.
iApply wp_mono; [|iApply "Hv"]; auto.
(* TLam *)
value_case.
iAlways. iIntros (τi) "%". iApply wp_tlam. iNext.
iApply IHtyped. iFrame "Hheap". by iApply interp_env_ren.
(* TApp *)
smart_wp_bind TAppCtx v "#Hv" IHtyped; cbn.
iApply wp_wand_r; iSplitL;
[iApply ("Hv" $! ([τ'] Δ)); iPureIntro; apply _|]; cbn.
iIntros (w) "?". by iApply interp_subst.
(* Fold *)
iApply (wp_bind [FoldCtx]);
iApply wp_wand_l; iSplitL; [|iApply (IHtyped Δ vs); auto].
iIntros (v) "#Hv". value_case.
rewrite /= -interp_subst fixpoint_unfold /=.
iAlways; eauto.
(* Unfold *)
iApply (wp_bind [UnfoldCtx]);
iApply wp_wand_l; iSplitL; [|iApply IHtyped; auto].
iIntros (v) "#Hv". rewrite /= fixpoint_unfold.
change (fixpoint _) with ([TRec τ] Δ); simpl.
iDestruct "Hv" as (w) "#[% Hw]"; subst.
iApply wp_fold; cbn; auto using to_of_val.
iNext; iModIntro. by iApply interp_subst.
(* Fork *)
iApply wp_fork. iNext; iSplitL; trivial.
iApply wp_wand_l. iSplitL; [|iApply IHtyped; auto]; auto.
(* Alloc *)
smart_wp_bind AllocCtx v "#Hv" IHtyped; cbn. iClear "HΓ". iApply wp_fupd.
iApply (wp_alloc with "Hheap []"); auto 1 using to_of_val.
iNext; iIntros (l) "Hl".
iMod (inv_alloc _ with "[Hl]") as "#N";
[| iModIntro; iExists _; iSplit; trivial]; eauto.
(* Load *)
smart_wp_bind LoadCtx v "#Hv" IHtyped; cbn. iClear "HΓ".
iDestruct "Hv" as (l) "[% #Hv]"; subst.
iInv (logN .@ l) as (w) "[Hw1 #Hw2]" "Hclose".
iApply ((wp_load _ _ 1) with "[Hw1] [Hclose]"); [|iFrame "Hheap"|];
trivial. solve_ndisj. iNext.
iIntros "Hw1". iMod ("Hclose" with "[-]"); eauto.
(* Store *)
smart_wp_bind (StoreLCtx _) v "#Hv" IHtyped1; cbn.
smart_wp_bind (StoreRCtx _) v "#Hw" IHtyped2; cbn. iClear "HΓ".
iDestruct "Hv" as (l) "[% #Hv]"; subst.
iInv (logN .@ l) as (z) "[Hz1 #Hz2]" "Hclose".
iApply (wp_store with "[Hz1] [Hclose]"); [| |iFrame "Hheap Hz1"|].
by rewrite to_of_val. solve_ndisj. iNext.
iIntros "Hz1". iMod ("Hclose" with "[-]"); eauto.
(* CAS *)
smart_wp_bind (CasLCtx _ _) v1 "#Hv1" IHtyped1; cbn.
smart_wp_bind (CasMCtx _ _) v2 "#Hv2" IHtyped2; cbn.
smart_wp_bind (CasRCtx _ _) v3 "#Hv3" IHtyped3; cbn. iClear "HΓ".
iDestruct "Hv1" as (l) "[% Hinv]"; subst.
iInv (logN .@ l) as (w) "[Hw1 #Hw2]" "Hclose".
destruct (decide (v2 = w)) as [|Hneq]; subst.
+ iApply (wp_cas_suc with "[Hw1] [Hclose]"); [| | |iFrame "Hheap Hw1"|];
eauto using to_of_val. solve_ndisj. iNext.
iIntros "Hw1". iMod ("Hclose" with "[-]"); eauto.
+ iApply (wp_cas_fail with "[Hw1] [Hclose]"); [| | | |iFrame "Hheap Hw1"|];
eauto using to_of_val. solve_ndisj. iNext.
iIntros "Hw1". iMod ("Hclose" with "[-]"); eauto.
Qed.
End typed_interp.
-U:---- fundamental_unary.v Top (21,0) Git:master
-U:---- fundamental_unary.v 44% (86,0) Git:master
-U:---- fundamental_unary.v Bot (131,0) Git:master

Contextual refinement

e_i contextually refines e_s ($\Gamma \vdash e_i \preceq_{ctx} e_s : \tau$):

$$\begin{aligned}\Gamma \vdash e_i \preceq_{ctx} e_s : \tau \triangleq\ & \Gamma \vdash e_i : \tau \wedge \\ & \Gamma \vdash e_s : \tau \wedge \\ & \forall \mathcal{C} : (\Gamma \vdash \tau \rightsquigarrow \cdot \vdash 1).\ \mathcal{C}[e_i]\downarrow \Rightarrow \mathcal{C}[e_s]\downarrow\end{aligned}$$

Useful when: e_i is a more efficient version of e_s (e.g., optimized by the compiler) or when e_s is easier to verify than e_i

The idea: Use a binary logical relation such that being related implies contextual refinement

Binary logical relation

To establish contextual refinement

Define *semantics* of types (by recursion on τ):
$[\![\tau]\!]^{\mathcal{E}} : \mathit{Expr} \to \mathit{Expr} \to \mathit{iProp}$

Binary logical relation

To establish contextual refinement

Define *semantics* of types (by recursion on τ):
$[\![\tau]\!]^{\mathcal{E}} : \mathit{Expr} \to \mathit{Expr} \to \mathit{iProp}$

1. Prove *Adequacy*: $[\![\tau]\!]^{\mathcal{E}}(e, e') \Rightarrow e \downarrow \Rightarrow e' \downarrow$

Binary logical relation

To establish contextual refinement

Define *semantics* of types (by recursion on τ):
$[\![\tau]\!]^{\mathcal{E}} : \mathit{Expr} \to \mathit{Expr} \to \mathit{iProp}$

1. Prove *Adequacy*: $[\![\tau]\!]^{\mathcal{E}}(e, e') \Rightarrow e \downarrow \Rightarrow e' \downarrow$
2. Prove *compatibility* lemmas for typing rules, e.g.:

$$\frac{[\![\tau_1]\!]^{\mathcal{E}}(e_1, e_1') \qquad [\![\tau]\!]^{\mathcal{E}}(e_2, e_2')}{[\![\tau_1 \times \tau_2]\!]^{\mathcal{E}}((e_1, e_2), (e_1', e_2'))}$$

Binary logical relation

To establish contextual refinement

Define *semantics* of types (by recursion on τ):
$[\![\tau]\!]^{\mathcal{E}} : \mathit{Expr} \to \mathit{Expr} \to \mathit{iProp}$

1. Prove *Adequacy*: $[\![\tau]\!]^{\mathcal{E}}(e, e') \Rightarrow e \downarrow \Rightarrow e' \downarrow$
2. Prove *compatibility* lemmas for typing rules, e.g.:

$$\frac{[\![\tau_1]\!]^{\mathcal{E}}(e_1, e_1') \qquad [\![\tau]\!]^{\mathcal{E}}(e_2, e_2')}{[\![\tau_1 \times \tau_2]\!]^{\mathcal{E}}((e_1, e_2), (e_1', e_2'))}$$

3. Corollary (*soundness*): $\Gamma \models e \preceq_{log} e' : \tau \Rightarrow \Gamma \vdash e \preceq_{ctx} e' : \tau$

Binary logical relation (for contextual refinement)

Definition in Iris: value relations

$$\llbracket \mathbb{N} \rrbracket_\Delta(v, v') \triangleq v = v' \in \mathbb{N}$$

$$\llbracket \tau_1 \times \tau_2 \rrbracket_\Delta(v, v') \triangleq \exists v_1, v_2, v_1', v_2'.\, v = (v_1, v_2) \land v' = (v_1', v_2') \land \llbracket \tau_1 \rrbracket_\Delta(v_1, v_1') \land \llbracket \tau_2 \rrbracket_\Delta(v_2, v_2')$$

$$\llbracket \mu X.\, \tau \rrbracket_\Delta(v, v') \triangleq \mu f.\, \exists w, w'.\, v = \texttt{fold}\, w \land v' = \texttt{fold}\, w' \land \triangleright \llbracket \tau \rrbracket_{\Delta[X \mapsto f]}(w, w')$$

$$\llbracket X \rrbracket_\Delta(v, v') \triangleq \Delta(X)(v, v')$$

$$\llbracket \texttt{ref}(\tau) \rrbracket_\Delta(v, v') \triangleq \exists \ell, \ell'.\, v = \ell \land v' = \ell' \land \boxed{\exists w, w'.\, \ell \mapsto_i w * \ell' \mapsto_s w' * \llbracket \tau \rrbracket_\Delta(w, w')}^{\mathcal{N}.\ell.\ell'}$$

Binary logical relation (for contextual refinement)

Definition in Iris: expression relation

The idea[1]: simulate the running of the right hand side as *ghost state*

- Ghost state for threads on the right hand side: $j \Rightarrow e$
- Ghost state for the heap of the right hand side: $\ell \mapsto_s v$

[1] See: A. Turon, D. Dreyer, and L. Birkedal. **Unifying refinement and hoare-style reasoning in a logic for higher-order concurrency**. In Proceedings of ICFP, 2013.
and M. Krogh-Jespersen, K. Svendsen, and L. Birkedal. **A relational model of types-and-effects in higher-order concurrent separation logic**. In Proceedings of POPL 2017, 2017.

Binary logical relation (for contextual refinement)

Definition in Iris: expression relation

The idea[1]: simulate the running of the right hand side as *ghost state*

- Ghost state for threads on the right hand side: $j \Rightarrow e$
- Ghost state for the heap of the right hand side: $\ell \mapsto_s v$

$$[\![\tau]\!]^{\mathcal{E}}_{\Delta}(e, e') \triangleq \forall j, K\ \{j \Rightarrow K[e']\}\, e\, \{w.\, \exists w'.\, j \Rightarrow K[w'] * [\![\tau]\!]_{\Delta}(w, w')\}$$

[1] See: A. Turon, D. Dreyer, and L. Birkedal. **Unifying refinement and hoare-style reasoning in a logic for higher-order concurrency**. In Proceedings of ICFP, 2013.
and M. Krogh-Jespersen, K. Svendsen, and L. Birkedal. **A relational model of types-and-effects in higher-order concurrent separation logic**. In Proceedings of POPL 2017, 2017.

Binary logical relation (for contextual refinement)

Definition in Iris: expression relation

The idea[1]: simulate the running of the right hand side as *ghost state*

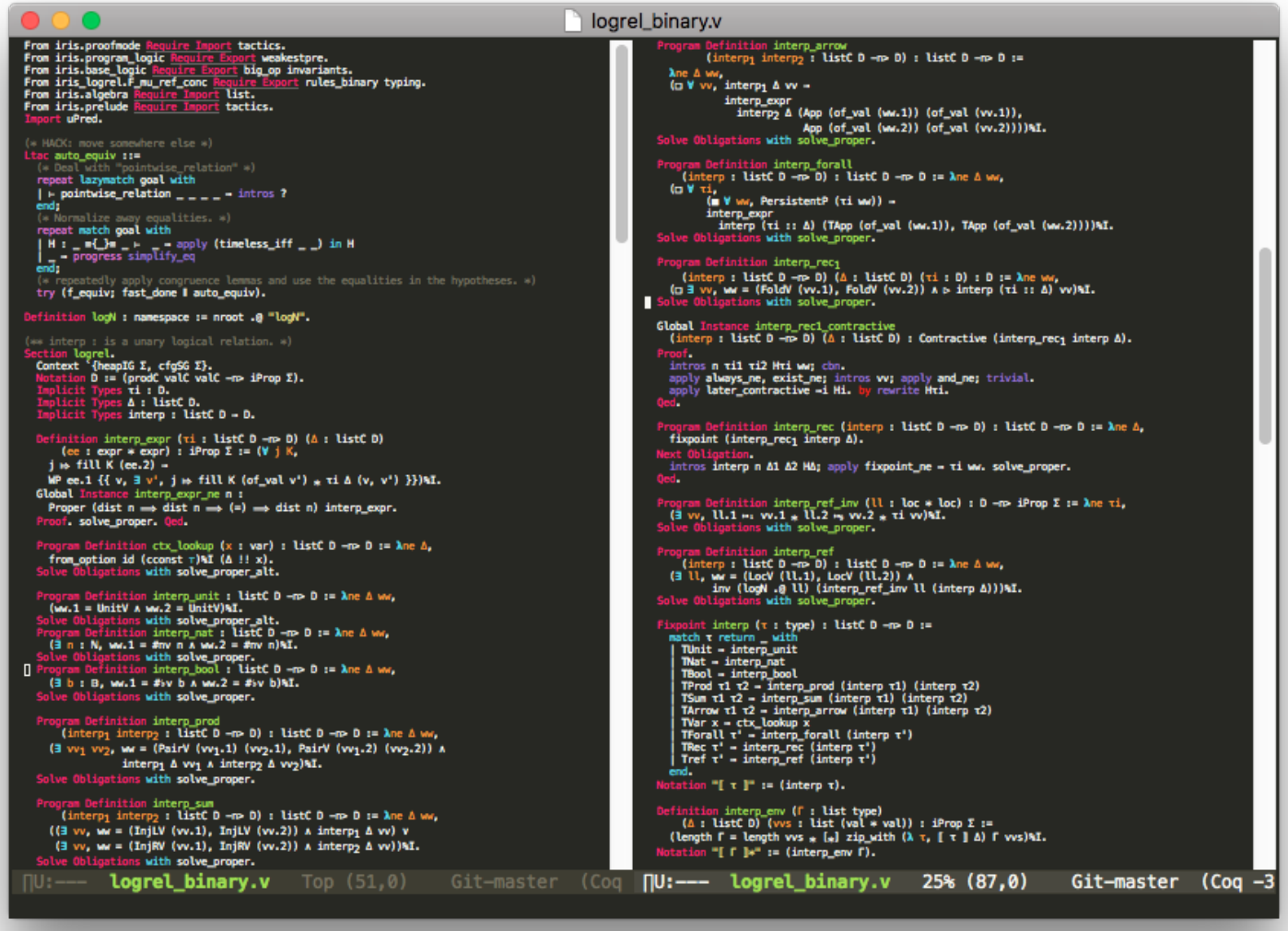
- Ghost state for threads on the right hand side: $j \Rightarrowtriangle e$
- Ghost state for the heap of the right hand side: $\ell \mapsto_s v$

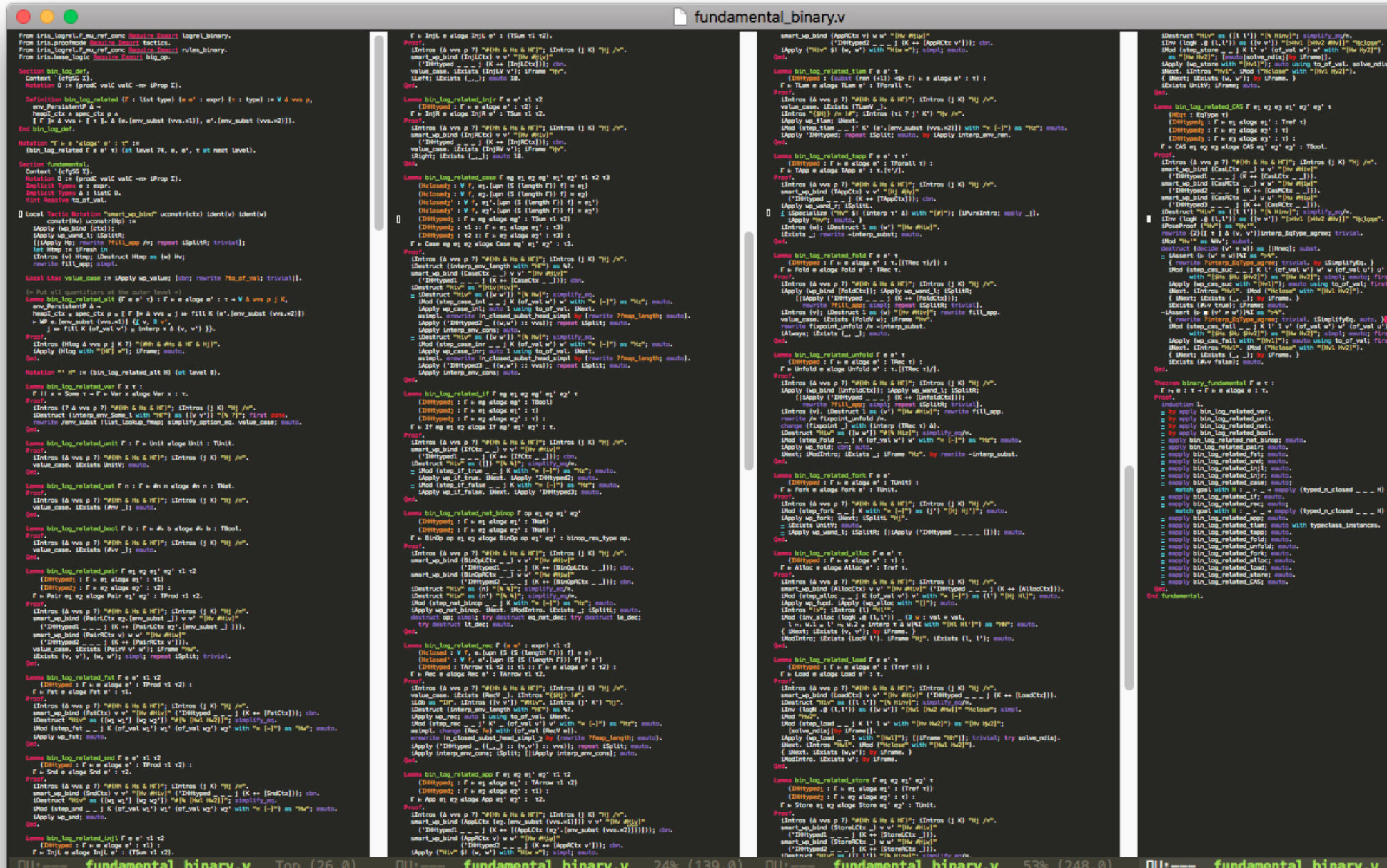
$$[\![\tau]\!]^{\mathcal{E}}_{\Delta}(e, e') \triangleq \forall j, K\ \{j \Rightarrowtriangle K[e']\}\, e\, \{w.\, \exists w'.\, j \Rightarrowtriangle K[w'] * [\![\tau]\!]_{\Delta}(w, w')\}$$

$$[\![\tau_1 \to \tau_2]\!]_{\Delta}(v, v') \triangleq \forall w, w', j, K.$$
$$\{[\![\tau_1]\!]_{\Delta}(w, w') * j \Rightarrowtriangle K[v'\ w']\}\, v\ w\, \{z.\, \exists z'.\, j \Rightarrowtriangle K[z'] * [\![\tau_2]\!]_{\Delta}(z, z')\}$$

---

[1] See: A. Turon, D. Dreyer, and L. Birkedal. **Unifying refinement and hoare-style reasoning in a logic for higher-order concurrency**. In Proceedings of ICFP, 2013.
and M. Krogh-Jespersen, K. Svendsen, and L. Birkedal. **A relational model of types-and-effects in higher-order concurrent separation logic**. In Proceedings of POPL 2017, 2017.


logrel_binary.v
From iris.proofmode Require Import tactics.
From iris.program_logic Require Export weakestpre.
From iris.base_logic Require Export big_op invariants.
From iris_logrel.F_mu_ref_conc Require Export rules_binary typing.
From iris.algebra Require Import list.
From iris.prelude Require Import tactics.
Import uPred.
(* HACK: move somewhere else *)
Ltac auto_equiv ::=
(* Deal with "pointwise_relation" *)
repeat lazymatch goal with
| ⊢ pointwise_relation _ _ _ _ → intros ?
end;
(* Normalize away equalities. *)
repeat match goal with
| H : _ ≡{_}≡ _ ⊢ _ → apply (timeless_iff _ _) in H
| _ → progress simplify_eq
end;
(* repeatedly apply congruence lemmas and use the equalities in the hypotheses. *)
try (f_equiv; fast_done || auto_equiv).
Definition logN : namespace := nroot .@ "logN".
(** interp : is a unary logical relation. *)
Section logrel.
Context `{heapIG Σ, cfgSG Σ}.
Notation D := (prodC valC valC -n> iProp Σ).
Implicit Types τi : D.
Implicit Types Δ : listC D.
Implicit Types interp : listC D → D.
Definition interp_expr (τi : listC D -n> D) (Δ : listC D)
(ee : expr * expr) : iProp Σ := (∀ j K,
j ⤇ fill K (ee.2) →
WP ee.1 {{ v, ∃ v', j ⤇ fill K (of_val v') ∗ τi Δ (v, v') }})%I.
Global Instance interp_expr_ne n :
Proper (dist n ==> dist n ==> (=) ==> dist n) interp_expr.
Proof. solve_proper. Qed.
Program Definition ctx_lookup (x : var) : listC D -n> D := λne Δ,
from_option id (cconst τ)%I (Δ !! x).
Solve Obligations with solve_proper_alt.
Program Definition interp_unit : listC D -n> D := λne Δ ww,
(ww.1 = UnitV ∧ ww.2 = UnitV)%I.
Solve Obligations with solve_proper_alt.
Program Definition interp_nat : listC D -n> D := λne Δ ww,
(∃ n : N, ww.1 = #nv n ∧ ww.2 = #nv n)%I.
Solve Obligations with solve_proper.
Program Definition interp_bool : listC D -n> D := λne Δ ww,
(∃ b : B, ww.1 = #sv b ∧ ww.2 = #sv b)%I.
Solve Obligations with solve_proper.
Program Definition interp_prod
(interp1 interp2 : listC D -n> D) : listC D -n> D := λne Δ ww,
(∃ vv1 vv2, ww = (PairV (vv1.1) (vv2.1), PairV (vv1.2) (vv2.2)) ∧
interp1 Δ vv1 ∧ interp2 Δ vv2)%I.
Solve Obligations with solve_proper.
Program Definition interp_sum
(interp1 interp2 : listC D -n> D) : listC D -n> D := λne Δ ww,
((∃ vv, ww = (InjLV (vv.1), InjLV (vv.2)) ∧ interp1 Δ vv) ∨
(∃ vv, ww = (InjRV (vv.1), InjRV (vv.2)) ∧ interp2 Δ vv))%I.
Solve Obligations with solve_proper.
Program Definition interp_arrow
(interp1 interp2 : listC D -n> D) : listC D -n> D :=
λne Δ ww,
(□ ∀ vv, interp1 Δ vv →
interp_expr
interp2 Δ (App (of_val (ww.1)) (of_val (vv.1)),
App (of_val (ww.2)) (of_val (vv.2))))%I.
Solve Obligations with solve_proper.
Program Definition interp_forall
(interp : listC D -n> D) : listC D -n> D := λne Δ ww,
(□ ∀ τi,
(⌜∀ ww, PersistentP (τi ww)⌝ →
interp_expr
interp (τi :: Δ) (TApp (of_val (ww.1)), TApp (of_val (ww.2))))%I.
Solve Obligations with solve_proper.
Program Definition interp_rec1
(interp : listC D -n> D) (Δ : listC D) (τi : D) : D := λne ww,
(□ ∃ vv, ww = (FoldV (vv.1), FoldV (vv.2)) ∧ ▷ interp (τi :: Δ) vv)%I.
Solve Obligations with solve_proper.
Global Instance interp_rec1_contractive
(interp : listC D -n> D) (Δ : listC D) : Contractive (interp_rec1 interp Δ).
Proof.
intros n τi1 τi2 Hτi ww; cbn.
apply always_ne, exist_ne; intros vv; apply and_ne; trivial.
apply later_contractive =>i Hi. by rewrite Hτi.
Qed.
Program Definition interp_rec (interp : listC D -n> D) : listC D -n> D := λne Δ,
fixpoint (interp_rec1 interp Δ).
Next Obligation.
intros interp n Δ1 Δ2 HΔ; apply fixpoint_ne => τi ww. solve_proper.
Qed.
Program Definition interp_ref_inv (ll : loc * loc) : D -n> iProp Σ := λne τi,
(∃ vv, ll.1 ↦ᵢ vv.1 ∗ ll.2 ↦ₛ vv.2 ∗ τi vv)%I.
Solve Obligations with solve_proper.
Program Definition interp_ref
(interp : listC D -n> D) : listC D -n> D := λne Δ ww,
(∃ ll, ww = (LocV (ll.1), LocV (ll.2)) ∧
inv (logN .@ ll) (interp_ref_inv ll (interp Δ)))%I.
Solve Obligations with solve_proper.
Fixpoint interp (τ : type) : listC D -n> D :=
match τ return _ with
| TUnit => interp_unit
| TNat => interp_nat
| TBool => interp_bool
| TProd τ1 τ2 => interp_prod (interp τ1) (interp τ2)
| TSum τ1 τ2 => interp_sum (interp τ1) (interp τ2)
| TArrow τ1 τ2 => interp_arrow (interp τ1) (interp τ2)
| TVar x => ctx_lookup x
| TForall τ' => interp_forall (interp τ')
| TRec τ' => interp_rec (interp τ')
| Tref τ' => interp_ref (interp τ')
end.
Notation "⟦ τ ⟧" := (interp τ).
Definition interp_env (Γ : list type)
(Δ : listC D) (vvs : list (val * val)) : iProp Σ :=
(length Γ = length vvs ∗ [∗] zip_with (λ τ, ⟦ τ ⟧ Δ) Γ vvs)%I.
Notation "⟦ Γ ⟧*" := (interp_env Γ).
U:--- logrel_binary.v Top (51,0) Git-master (Coq
U:--- logrel_binary.v 25% (87,0) Git-master (Coq -3


fundamental_binary.v
fundamental_binary.v Top (26,0)
fundamental_binary.v 24% (139,0)
fundamental_binary.v 53% (248,0)
fundamental_binary.v

Remark

- Much simpler than previous explicit definitions
  – even when formalized!

Explicit definition of interpretation

$$
\begin{array}{rcl}
\mathrm{V}[\![\alpha]\!]\rho & \stackrel{\text{def}}{=} & \rho(\alpha).r \\
\mathrm{V}[\![b]\!]\rho & \stackrel{\text{def}}{=} & \{(W,v,v) \in \mathrm{TermAtom}[b,b]\} \\
\mathrm{V}[\![\tau\times\tau']\!]\rho & \stackrel{\text{def}}{=} & \{(W,\langle v_1,v_1'\rangle,\langle v_2,v_2'\rangle) \in \mathrm{TermAtom}[\rho_1(\tau\times\tau'),\rho_2(\tau\times\tau')] \mid \\
 & & \quad (W,v_1,v_2)\in \mathrm{V}[\![\tau]\!]\rho \wedge (W,v_1',v_2')\in \mathrm{V}[\![\tau']\!]\rho\} \\
\mathrm{V}[\![\tau'\to\tau]\!]\rho & \stackrel{\text{def}}{=} & \{(W,\lambda x{:}\tau_1.e_1,\lambda x{:}\tau_2.e_2) \in \mathrm{TermAtom}[\rho_1(\tau'\to\tau),\rho_2(\tau'\to\tau)] \mid \\
 & & \quad \forall W',v_1,v_2.\ W'\sqsupseteq W \wedge (W',v_1,v_2)\in \mathrm{V}[\![\tau']\!]\rho \Rightarrow \\
 & & \quad (W',e_1[v_1/x],e_2[v_2/x])\in \mathrm{E}[\![\tau]\!]\rho\} \\
\mathrm{V}[\![\forall\alpha.\tau]\!]\rho & \stackrel{\text{def}}{=} & \{(W,\Lambda\alpha.e_1,\Lambda\alpha.e_2) \in \mathrm{TermAtom}[\rho_1(\forall\alpha.\tau),\rho_2(\forall\alpha.\tau)] \mid \\
 & & \quad \forall W'\sqsupseteq W.\ \forall(\tau_1,\tau_2,r)\in\mathrm{SomeValRel}. \\
 & & \quad (W',e_1[\tau_1/\alpha],e_2[\tau_2/\alpha])\in \mathrm{E}[\![\tau]\!]\rho,\alpha{\mapsto}(\tau_1,\tau_2,r)\} \\
\mathrm{V}[\![\exists\alpha.\tau]\!]\rho & \stackrel{\text{def}}{=} & \{(W,\mathsf{pack}\ \langle\tau_1,v_1\rangle\ \mathsf{as}\ \tau_1',\mathsf{pack}\ \langle\tau_2,v_2\rangle\ \mathsf{as}\ \tau_2') \in \mathrm{TermAtom}[\rho_1(\exists\alpha.\tau),\rho_2(\exists\alpha.\tau)] \mid \\
 & & \quad \exists r.\ (\tau_1,\tau_2,r)\in\mathrm{SomeValRel}\wedge(W,v_1,v_2)\in\mathrm{V}[\![\tau]\!]\rho,\alpha{\mapsto}(\tau_1,\tau_2,r)\} \\
\mathrm{V}[\![\mu\alpha.\tau]\!]\rho & \stackrel{\text{def}}{=} & \{(W,\mathsf{roll}_{\tau_1}\ v_1,\mathsf{roll}_{\tau_2}\ v_2) \in \mathrm{TermAtom}[\rho_1(\mu\alpha.\tau),\rho_2(\mu\alpha.\tau)] \mid \\
 & & \quad (W,v_1,v_2)\in \triangleright\mathrm{V}[\![\tau[\mu\alpha.\tau/\alpha]]\!]\rho\} \\
\mathrm{V}[\![\mathsf{ref}\ \tau]\!]\rho & \stackrel{\text{def}}{=} & \{(W,l_1,l_2)\in\mathrm{TermAtom}[\rho_1(\mathsf{ref}\ \tau),\rho_2(\mathsf{ref}\ \tau)] \mid \exists i.\ \forall W'\sqsupseteq W. \\
 & & \quad (l_1,l_2)\in\mathrm{bij}(W'(i).s)\wedge\exists\psi.\ W'(i).H(W'(i).s) = \\
 & & \quad \psi\otimes\{(\widetilde{W},\{l_1{\mapsto}v_1\},\{l_2{\mapsto}v_2\})\in\mathrm{HeapAtom}\mid(\widetilde{W},v_1,v_2)\in\mathrm{V}[\![\tau]\!]\rho\}\} \\
 & & \\
\mathrm{O} & \stackrel{\text{def}}{=} & \{(W,e_1,e_2)\mid\forall h_1,h_2.\ (h_1,h_2):W\wedge\langle h_1;e_1\rangle{\downarrow}^{<W.k}\Rightarrow\mathrm{consistent}(W)\wedge\langle h_2;e_2\rangle{\downarrow}\}
\end{array}
$$

$$
\begin{array}{rcl}
\mathrm{K}[\![\tau]\!]\rho & \stackrel{\text{def}}{=} & \{(W,K_1,K_2)\in\mathrm{ContAtom}[\rho_1(\tau),\rho_2(\tau)] \mid \\
 & & \quad \forall W',v_1,v_2.\ W'\sqsupseteq^{\mathsf{pub}} W\wedge(W',v_1,v_2)\in\mathrm{V}[\![\tau \\
 & & \\
\mathrm{E}[\![\tau]\!]\rho & \stackrel{\text{def}}{=} & \{(W,e_1,e_2)\in\mathrm{TermAtom}[\rho_1(\tau),\rho_2(\tau)] \mid \\
 & & \quad \forall K_1,K_2.\ (W,K_1,K_2)\in\mathrm{K}[\![\tau]\!]\rho\Rightarrow(W,K_1[e_1] \\
 & & \\
\mathrm{G}[\![\cdot]\!]\rho & \stackrel{\text{def}}{=} & \{(W,\emptyset)\mid W\in\mathrm{World}\} \\
\mathrm{G}[\![\Gamma,x{:}\tau]\!]\rho & \stackrel{\text{def}}{=} & \{(W,(\gamma,x{\mapsto}(v_1,v_2)))\mid(W,\gamma)\in\mathrm{G}[\![\Gamma]\!]\rho\wedge(W,v_1 \\
\mathrm{D}[\![\cdot]\!] & \stackrel{\text{def}}{=} & \{\emptyset\} \\
\mathrm{D}[\![\Delta,\alpha]\!] & \stackrel{\text{def}}{=} & \{\rho,\alpha{\mapsto}R\mid\rho\in\mathrm{D}[\![\Delta]\!]\wedge R\in\mathrm{SomeValRel}\} \\
\mathrm{S}[\![\cdot]\!] & \stackrel{\text{def}}{=} & \mathrm{World} \\
\mathrm{S}[\![\Sigma,l{:}\tau]\!] & \stackrel{\text{def}}{=} & \mathrm{S}[\![\Sigma]\!]\cap\{W\in\mathrm{World}\mid(W,l,l)\in\mathrm{V}[\![\mathsf{ref}\ \tau]\!]\emptyset\}
\end{array}
$$

$$
\Sigma;\Delta;\Gamma\vdash e_1\precsim_{\log} e_2:\tau \stackrel{\text{def}}{=} \Sigma;\Delta;\Gamma\vdash e_1:\tau\wedge\Sigma;\Delta;\Gamma\vdash e_2: \\
\forall W,\rho,\gamma.\ W\in\mathrm{S}[\![\Sigma]\!]\wedge\rho\in\mathrm{D}[\![\Delta \\
(W,\rho_1\gamma_1e_1,\rho_2\gamma_2e_2)\in\mathrm{E}[\![\tau]\!]\rho
$$

Fig. 6. A step-indexed biorthogonal Kripke logica

Figure taken from: D. Dreyer, G. Neis, and L. Birkedal. **The impact of higher-order state and control effects on local relational reasoning**. *Journal of Functional Programming*, February 2012.

Auxiliary definitions

$$
\begin{array}{rcl}
\mathrm{HeapAtom}_n & \stackrel{\text{def}}{=} & \{(W,h_1,h_2) \mid W \in \mathrm{World}_n\} \\
\mathrm{HeapRel}_n & \stackrel{\text{def}}{=} & \{\psi \subseteq \mathrm{HeapAtom}_n \mid \forall (W,h_1,h_2) \in \psi.\ \forall W' \sqsupseteq W.\ (W',h_1,h_2) \in \psi\} \\
\mathrm{Island}_n & \stackrel{\text{def}}{=} & \{\iota = (s,\delta,\varphi,\lightning,H) \mid s \in \mathrm{State} \wedge \delta \subseteq \mathrm{State}^2 \wedge \varphi \subseteq \delta \wedge \delta,\varphi \text{ reflexive} \wedge \\
 & & \quad \delta,\varphi \text{ transitive} \wedge \lightning \subseteq \mathrm{State} \wedge H \in \mathrm{State} \to \mathrm{HeapRel}_n\} \\
\mathrm{World}_n & \stackrel{\text{def}}{=} & \{W = (k,\Sigma_1,\Sigma_2,\omega) \mid k < n \wedge \exists m.\ \omega \in \mathrm{Island}_k^m\} \\
\mathrm{ContAtom}_n[\tau_1,\tau_2] & \stackrel{\text{def}}{=} & \{(W,K_1,K_2) \mid W \in \mathrm{World}_n \wedge W.\Sigma_1;\cdot;\cdot \vdash K_1 \div \tau_1 \wedge W.\Sigma_2;\cdot;\cdot \vdash K_2 \div \tau_2\} \\
\mathrm{TermAtom}_n[\tau_1,\tau_2] & \stackrel{\text{def}}{=} & \{(W,e_1,e_2) \mid W \in \mathrm{World}_n \wedge W.\Sigma_1;\cdot;\cdot \vdash e_1 : \tau_1 \wedge W.\Sigma_2;\cdot;\cdot \vdash e_2 : \tau_2\} \\
 & & \\
\mathrm{HeapAtom}[\tau_1,\tau_2] & \stackrel{\text{def}}{=} & \bigcup_n \mathrm{HeapAtom}_n[\tau_1,\tau_2] \\
\mathrm{World} & \stackrel{\text{def}}{=} & \bigcup_n \mathrm{World}_n \\
\mathrm{ContAtom}[\tau_1,\tau_2] & \stackrel{\text{def}}{=} & \bigcup_n \mathrm{ContAtom}_n[\tau_1,\tau_2] \\
\mathrm{TermAtom}[\tau_1,\tau_2] & \stackrel{\text{def}}{=} & \bigcup_n \mathrm{TermAtom}_n[\tau_1,\tau_2] \\
 & & \\
\mathrm{ValRel}[\tau_1,\tau_2] & \stackrel{\text{def}}{=} & \{r \subseteq \mathrm{TermAtom}^{\mathsf{val}}[\tau_1,\tau_2] \mid \forall (W,v_1,v_2) \in r.\ \forall W' \sqsupseteq W.\ (W',v_1,v_2) \in r\} \\
\mathrm{SomeValRel} & \stackrel{\text{def}}{=} & \{R = (\tau_1,\tau_2,r) \mid r \in \mathrm{ValRel}[\tau_1,\tau_2]\}
\end{array}
$$

$$
\begin{array}{rcl@{\qquad}rcl}
\lfloor(\iota_1,\ldots,\iota_m)\rfloor_k & \stackrel{\text{def}}{=} & (\lfloor\iota_1\rfloor_k,\ldots,\lfloor\iota_m\rfloor_k) & \lfloor H\rfloor_k & \stackrel{\text{def}}{=} & \lambda s.\lfloor H(s)\rfloor_k \\
\lfloor(s,\delta,\varphi,\lightning,H)\rfloor_k & \stackrel{\text{def}}{=} & (s,\delta,\varphi,\lightning,\lfloor H\rfloor_k) & \lfloor\psi\rfloor_k & \stackrel{\text{def}}{=} & \{(W,h_1,h_2) \in r \mid W.k < k\}
\end{array}
$$

$$
\begin{array}{rcl}
\triangleright(k+1,\Sigma_1,\Sigma_2,\omega) & \stackrel{\text{def}}{=} & (k,\Sigma_1,\Sigma_2,\lfloor\omega\rfloor_k) \\
\triangleright r & \stackrel{\text{def}}{=} & \{(W,e_1,e_2) \mid W.k > 0 \Rightarrow (\triangleright W,e_1,e_2) \in r\}
\end{array}
$$

$$
\begin{array}{rcl@{\qquad}rcl}
(k',\Sigma_1',\Sigma_2',\omega') & \sqsupseteq & (k,\Sigma_1,\Sigma_2,\omega) & \stackrel{\text{def}}{=} & k' \le k \wedge \Sigma_1' \supseteq \Sigma_1 \wedge \Sigma_2' \supseteq \Sigma_2 \wedge \omega' \sqsupseteq \lfloor\omega\rfloor_{k'} \\
(\iota_1',\ldots,\iota_{m'}') & \sqsupseteq & (\iota_1,\ldots,\iota_m) & \stackrel{\text{def}}{=} & m' \ge m \wedge \forall j \in \{1,\ldots,m\}.\ \iota_j' \sqsupseteq \iota_j \\
(s',\delta',\varphi',\lightning',H') & \sqsupseteq & (s,\delta,\varphi,\lightning,H) & \stackrel{\text{def}}{=} & (\delta',\varphi',\lightning',H') = (\delta,\varphi,\lightning,H) \wedge (s,s') \in \delta \\
 & & & & \\
(k',\Sigma_1',\Sigma_2',\omega') & \sqsupseteq^{\mathsf{pub}} & (k,\Sigma_1,\Sigma_2,\omega) & \stackrel{\text{def}}{=} & k' \le k \wedge \Sigma_1' \supseteq \Sigma_1 \wedge \Sigma_2' \supseteq \Sigma_2 \wedge \omega' \sqsupseteq^{\mathsf{pub}} \lfloor\omega\rfloor_{k'} \\
(\iota_1',\ldots,\iota_{m'}') & \sqsupseteq^{\mathsf{pub}} & (\iota_1,\ldots,\iota_m) & \stackrel{\text{def}}{=} & m' \ge m \wedge \forall j \in \{1,\ldots,m\}.\ \iota_j' \sqsupseteq^{\mathsf{pub}} \iota_j \wedge \\
 & & & & \quad \forall j \in \{m+1,\ldots,m'\}.\ \mathrm{safe}(\iota_j') \\
(s',\delta',\varphi',\lightning',H') & \sqsupseteq^{\mathsf{pub}} & (s,\delta,\varphi,\lightning,H) & \stackrel{\text{def}}{=} & (\delta',\varphi',\lightning',H') = (\delta,\varphi,\lightning,H) \wedge (s,s') \in \varphi
\end{array}
$$

$$
\mathrm{safe}(W) \stackrel{\text{def}}{=} \forall \iota \in W.\omega.\ \mathrm{safe}(\iota) \qquad \mathrm{safe}(\iota) \stackrel{\text{def}}{=} \forall s'.\ (\iota.s,s') \in \iota.\varphi \Rightarrow s' \notin \iota.\lightning
$$

$$
\mathrm{consistent}(W) \stackrel{\text{def}}{=} \nexists \iota \in W.\omega.\ \iota.s \in \iota.\lightning
$$

$$
\psi \otimes \psi' \stackrel{\text{def}}{=} \{(W,h_1 \uplus h_1',h_2 \uplus h_2') \mid (W,h_1,h_2) \in \psi \wedge (W,h_1',h_2') \in \psi'\}
$$

$$
(h_1,h_2) : W \stackrel{\text{def}}{=} \vdash h_1 : W.\Sigma_1 \wedge \vdash h_2 : W.\Sigma_2 \wedge (W.k > 0 \Rightarrow (\triangleright W,h_1,h_2) \in \bigotimes\{\iota.H(\iota.s) \mid \iota \in W.\omega\})
$$

Fig. 5. Worlds and auxiliary definitions.

it is defined on a particular set of "states of interest"—whether there is other junk in the State space is irrelevant.

Based on the two transition relations (full and public), we define two notions

Examples of refinement of concurrent programs

- Fine-grained/coarse-grained counter pair

```
let FG_counter =
  let c = ref 0 in
    let read () = !c in
    let rec increment () =
      let x = !c in if CAS(c, x, x+1) then () else
    increment ()
    in (increment, read)

let CG_counter =
  let c = ref 0 in let l = make_lock () in
    let read () = !c in
    let increment () = acquire l; c := !c + 1;
    release l
    in (increment, read)
```

  We show: $[\![(1 \to 1) \times (1 \to \mathbb{N})]\!]^{\mathcal{E}}_{\emptyset}(\texttt{FG\_counter}, \texttt{CG\_counter})$

- Fine-grained/coarse-grained stack pair with push, pop and iter operations

Trusted computing base

- The only thing that needs be trusted is **Coq**
- We use the **adequacy** of Iris to prove theorems (contextual refinement, type safety, ...) in Coq

The refinement proven in Coq

```
Theorem counter_ctx_refinement :
  [] ⊨ FG_counter ≤ctx≤ CG_counter :
        TProd (TArrow TUnit TUnit) (TArrow TUnit
    TNat).
```

```
Definition ctx_refines (Γ : list type)
    (e e' : expr) (τ : type) := ∀ K thp σ v,
  typed_ctx K Γ τ [] TUnit →
  rtc step ([fill_ctx K e], ∅) (of_val v :: thp, σ)
    →
  ∃ thp' σ' v', rtc step ([fill_ctx K e'], ∅) (of_
    val v' :: thp', σ').
Notation "Γ ⊨ e '≤ctx≤' e' : τ" :=
  (ctx_refines Γ e e' τ) (at level 74, e, e', τ at
    next level).
```

Models for More Advanced Type Systems

- M. Krogh-Jespersen, K. Svendsen, and L. Birkedal. **A relational model of types-and-effects in higher-order concurrent separation logic** [POPL 2017]
  - Relational model.
  - Used to show soundness of type-based parallelization.
- A. Timany, L. Stefanesco, M. Krogh-Jespersen, and L. Birkedal. **A Logical Relation for Monadic Encapsulation of State: Proving contextual equivalences in the presence of runST** [POPL 2018]
  - Relational model.
  - Used to show soundness of pure program equivalences in the presence of runST encapsulated state.
- R. Jung, J-H. Jourdan, R. Krebbers, D. Dreyer. **RustBelt: Securing the Foundations of the Rust Programming Language** [POPL 2018]
  - Unary model.
  - Used to model and show soundness of Rust type system.

Future work

- On the technical side:
  - Use a more sensible binding representation (we currently use De Bruijn indexes)
  - Better facilitate symbolic execution for $\mathsf{F}_{\mu,ref,conc}$
- Prove more interesting (and larger) cases of contextual refinements
  - Dan Frumin and Amin Timany just proved that a fine-grained stack *with helping* refines a coarse-grained stack.
- Logical relations for languages with richer type systems and features (e.g., continuations, type-and-effect systems, algebraic effects, . . .)
- Apply it to other application (e.g., non-interference proofs, compiler correctness, secure compilation, . . .)
- Your logical relation applications?! Please do not hesitate to talk to us!

Thanks!