A calculus for logical refinements in higher-order separation logic

Dan Frumin[1] **Robbert Krebbers**[2]

CoqPL 2018, January 13

[1]Radboud University Nijmegen

[2]Delft University of Technology

Overview and goals

Introduction

Goal: mechanisation of a logic for relational reasoning about stateful, concurrent, higher-order programs.

Example:

$\Gamma \vDash \texttt{ticket\_lock} \precsim \texttt{spin\_lock}$
$: \exists\!:\ (\texttt{Unit} \to \texttt{TVar}\ 0) \times (\texttt{TVar}\ 0 \to \texttt{Unit}) \times (\texttt{TVar}\ 0 \to \texttt{Unit}).$

Features:

- Mechanised soundness proofs;
- Ease of reasoning;
- Modularity of the proofs.

Introduction

Goal: mechanisation of a logic for relational reasoning about stateful, concurrent, higher-order programs.

Example:

$\Gamma \vDash$ `ticket_lock` $\precsim$ `spin_lock`
$: \exists{:}\ ($`Unit` $\to$ `TVar` $0) \times ($`TVar` $0 \to$ `Unit`$) \times ($`TVar` $0 \to$ `Unit`$)$.

Features:

- Mechanised soundness proofs;
- Ease of reasoning;
- Modularity of the proofs.

Setting

- Object logic is higher-order separation logic with judgements $\Delta; \Gamma \models e_1 \precsim e_2 : \tau$.
- Programs in ML-like language: System $F + \exists + \mu +$ **ref** $+$ fork.
- Soundness with regard to contextual equivalence: $(\forall \Delta,\ \Delta; \Gamma \models e_1 \precsim e_2 : \tau) \implies \Gamma \vdash e_1 \precsim_{ctx} e_2 : \tau$.

History overview

- **Explicit step-indexing** (Appel-McAllester, Ahmed, . . .)
    - Definitions: low-level using explicit step-indexing and resources.
    - Proofs: unfolding the definitions, lots of low-level details.

History overview

- **Explicit step-indexing** (Appel-McAllester, Ahmed, . . .)
  - Definitions: low-level using explicit step-indexing and resources.
  - Proofs: unfolding the definitions, lots of low-level details.
- **Logical approach** (LSLR, LADR, CaReSL, Iris, . . .)
  - Definitions: high-level using a logic to hide explicit steps and/or resources.
  - Proofs: unfolding the definitions, simpler proofs in {LSLR, LADR, CaReSL, Iris, . . . }

History overview

- **Explicit step-indexing** (Appel-McAllester, Ahmed, . . .)
  - Definitions: low-level using explicit step-indexing and resources.
  - Proofs: unfolding the definitions, lots of low-level details.
- **Logical approach** (LSLR, LADR, CaReSL, Iris, . . .)
  - Definitions: high-level using a logic to hide explicit steps and/or resources.
  - Proofs: unfolding the definitions, simpler proofs in {LSLR, LADR, CaReSL, Iris, . . . }

Problem: breaks abstraction!

History overview

- **Explicit step-indexing** (Appel-McAllester, Ahmed, . . .)
  - Definitions: low-level using explicit step-indexing and resources.
  - Proofs: unfolding the definitions, lots of low-level details.
- **Logical approach** (LSLR, LADR, CaReSL, Iris, . . .)
  - Definitions: high-level using a logic to hide explicit steps and/or resources.
  - Proofs: unfolding the definitions, simpler proofs in {LSLR, LADR, CaReSL, Iris, . . . }

Problem: breaks abstraction!

- **This work**
  - Definitions: high-level using a logic to hide explicit steps and/or resources.
  - Proofs: using high-level proof rules + reasoning principles of Iris (ghost state, invariants, . . .)

Example 1: representation independence

Example: bit module

A bit interface:

$$\texttt{bitT} \triangleq \exists \alpha . \alpha \times (\alpha \to \alpha) \times (\alpha \to \texttt{bool})$$

- initial state
- flip the bit
- view the bit as a Boolean

Two implementations:

```
Definition bit_bool : val :=
  pack (#true,
        (λ: "b", ¬ "b"),
        (λ: "b", "b")).
```

```
Definition bit_nat : val :=
  pack (#1,
     (λ: "n", if: ("n" = #0)
           then #1 else #0),
     (λ: "b", "b" = #1)).
```

Example: bit refinement

In order to prove the refinement:

$$\texttt{bit\_bool} \precsim \texttt{bit\_nat} : \texttt{bitT}$$

we select a relation linking together the underlying types of `bit_bool` and `bit_nat`:

$$R \subseteq \texttt{Bool} \times \texttt{Nat} = \{(\texttt{true}, 1), (\texttt{false}, 0)\}.$$

```
Definition f (b : bool) : nat := if b then 1 else 0.
Definition R : D := valrel (λ v1 v2,
  (∃ b : bool, ⌜v1 = #b⌝ ∗ ⌜v2 = #(f b)⌝))%I.
```

DEMO

```
Lemma bit_refinement Δ Γ :
  {Δ;Γ} ⊨ bit_bool ≾ bit_nat : bitT.
 Proof.




Qed.
```

```
Δ : list D
Γ : stringmap type
================================ (1/1)
{Δ; Γ} ⊨ bit_bool ≾ bit_nat : bitT
```

DEMO

```
Lemma bit_refinement Δ Γ :
  {Δ;Γ} ⊨ bit_bool ≾ bit_nat : bitT.
 Proof.
  unlock bit_bool bit_nat; simpl.









Qed.
```

```
Δ : list D
Γ : stringmap type
============================== (1/1)
{Δ; Γ} ⊨ bit_bool ≾ bit_nat : bitT
```

DEMO

```
Lemma bit_refinement Δ Γ :
  {Δ;Γ} ⊨ bit_bool ≾ bit_nat : bitT.
 Proof.
  unlock bit_bool bit_nat; simpl.




Qed.
```

```
Δ : list D
Γ : stringmap type
================================ (1/1)
{Δ; Γ} ⊨
  pack (#true,
        λ: "b", ¬ "b",
        λ: "b", "b")
≾
  pack (#1,
        λ: "n", if: "n" = #0 then #1
                else #0,
        λ: "b", "b" = #1) : bitT
```

DEMO

```
Lemma bit_refinement Δ Γ :
  {Δ;Γ} ⊨ bit_bool ≾ bit_nat : bitT.
 Proof.
  unlock bit_bool bit_nat; simpl.
  iApply (bin_log_related_pack _ R).




Qed.
```

```
Δ : list D
Γ : stringmap type
══════════════════════════════ (1/1)
{Δ; Γ} ⊨
  pack (#true,
        λ: "b", ¬ "b",
        λ: "b", "b")
≾
  pack (#1,
        λ: "n", if: "n" = #0 then #1
              else #0,
        λ: "b", "b" = #1) : bitT
```

DEMO

```
Lemma bit_refinement Δ Γ :
  {Δ;Γ} ⊨ bit_bool ≾ bit_nat : bitT.
 Proof.
  unlock bit_bool bit_nat; simpl.
  iApply (bin_log_related_pack _ R).




Qed.
```

```
Δ : list D
Γ : stringmap type
══════════════════════════════ (1/1)
{(R :: Δ); ↑Γ} ⊨
  (#true, λ: "b", ¬ "b", λ: "b", "b")
≾
  (#1, λ: "n", if: "n" = #0 then #1
        else #0, λ: "b", "b" = #1) :
TVar 0 × (TVar 0 → TVar 0) × (TVar 0
      → Bool)
```

DEMO

```
Lemma bit_refinement Δ Γ :
  {Δ;Γ} ⊨ bit_bool ≾ bit_nat : bitT.
 Proof.
  unlock bit_bool bit_nat; simpl.
  iApply (bin_log_related_pack _ R).
  repeat iApply bin_log_related_pair.






Qed.
```

```
Δ : list D
Γ : stringmap type
══════════════════════════════════ (1/1)
{(R :: Δ); ↑Γ} ⊨
  (#true, λ: "b", ¬ "b", λ: "b", "b")
≾
  (#1, λ: "n", if: "n" = #0 then #1
          else #0, λ: "b", "b" = #1) :
TVar 0 × (TVar 0 → TVar 0) × (TVar 0
      → Bool)
```

DEMO

```
Lemma bit_refinement Δ Γ :
  {Δ;Γ} ⊨ bit_bool ≾ bit_nat : bitT.
 Proof.
  unlock bit_bool bit_nat; simpl.
  iApply (bin_log_related_pack _ R).
  repeat iApply bin_log_related_pair.







Qed.
```

```
3 subgoals

Δ : list D
Γ : stringmap type
================================(1/3)
{(R :: Δ); ↑Γ} ⊨ #true ≾ #1 : TVar 0

Δ : list D
Γ : stringmap type
================================(2/3)
{(R :: Δ); ↑Γ} ⊨
  (λ: "b", ¬ "b")
≾
  (λ: "n", if: "n" = #0 then #1 else
       #0) : (TVar 0 → TVar 0)


...
```

DEMO

```
Lemma bit_refinement Δ Γ :
  {Δ;Γ} ⊨ bit_bool ≾ bit_nat : bitT.
 Proof.
  unlock bit_bool bit_nat; simpl.
  iApply (bin_log_related_pack _ R).
  repeat iApply bin_log_related_pair.
  -

Qed.
```

```
Δ : list D
Γ : stringmap type
=================================== (1/1)
{(R :: Δ); ↑Γ} ⊨ #true ≾ #1 : TVar 0
```

DEMO

```
Lemma bit_refinement Δ Γ :
  {Δ;Γ} ⊨ bit_bool ≾ bit_nat : bitT.
 Proof.
  unlock bit_bool bit_nat; simpl.
  iApply (bin_log_related_pack _ R).
  repeat iApply bin_log_related_pair.
  - rel_finish.

Qed.
```

```
Δ : list D
Γ : stringmap type
================================ (1/1)
{(R :: Δ); ↑Γ} ⊨ #true ≾ #1 : TVar 0
```

DEMO

```
Lemma bit_refinement Δ Γ :
  {Δ;Γ} ⊨ bit_bool ≾ bit_nat : bitT.
 Proof.
  unlock bit_bool bit_nat; simpl.
  iApply (bin_log_related_pack _ R).
  repeat iApply bin_log_related_pair.
  - rel_finish.









Qed.
```

```
This subproof is complete, but there
     are some unfocused goals.
Focus next goal with bullet -.
```

DEMO

```
Lemma bit_refinement Δ Γ :
  {Δ;Γ} ⊨ bit_bool ≾ bit_nat : bitT.
 Proof.
  unlock bit_bool bit_nat; simpl.
  iApply (bin_log_related_pack _ R).
  repeat iApply bin_log_related_pair.
  - rel_finish.
  -
Qed.
```

```
Δ : list D
Γ : stringmap type
================================ (1/1)
{(R :: Δ); ↑Γ} ⊨
  (λ: "b", ¬ "b")
≾
  (λ: "n", if: "n" = #0 then #1 else
       #0) : (TVar 0 → TVar 0)
```

DEMO

```
Lemma bit_refinement Δ Γ :
  {Δ;Γ} ⊨ bit_bool ≾ bit_nat : bitT.
 Proof.
  unlock bit_bool bit_nat; simpl.
  iApply (bin_log_related_pack _ R).
  repeat iApply bin_log_related_pair.
  - rel_finish.
  - rel_arrow_val.  simpl.









Qed.
```

```
Δ : list D
Γ : stringmap type
═══════════════════════════════ (1/1)
{(R :: Δ); ↑Γ} ⊨
  (λ: "b", ¬ "b")
≾
  (λ: "n", if: "n" = #0 then #1 else
        #0) : (TVar 0 → TVar 0)
```

DEMO

```
Lemma bit_refinement Δ Γ :
  {Δ;Γ} ⊨ bit_bool ≾ bit_nat : bitT.
 Proof.
  unlock bit_bool bit_nat; simpl.
  iApply (bin_log_related_pack _ R).
  repeat iApply bin_log_related_pair.
  - rel_finish.
  - rel_arrow_val.  simpl.


Qed.
```

```
Δ : list D
Γ : stringmap type
==============================
□ (∀ v1 v2 : valC,
   □ (∃ b : bool, ⌜v1 = #b⌝ ∗ ⌜v2 =
        #(f b)⌝) -∗
   {(R :: Δ); ↑Γ} ⊨
     (let: "b" := v1 in ¬ "b")
  ≾
     (let: "n" := v2 in if: "n" = #0
          then #1 else #0) :
   TVar 0)
```

DEMO

```
Lemma bit_refinement Δ Γ :
  {Δ;Γ} ⊨ bit_bool ≾ bit_nat : bitT.
 Proof.
  unlock bit_bool bit_nat; simpl.
  iApply (bin_log_related_pack _ R).
  repeat iApply bin_log_related_pair.
  - rel_finish.
  - rel_arrow_val.  simpl.
    iIntros "!#" (v1 v2).






Qed.
```

```
Δ : list D
Γ : stringmap type
════════════════════════════════════
□ (∀ v1 v2 : valC,
   □ (∃ b : bool, ⌜v1 = #b⌝ ∗ ⌜v2 =
        #(f b)⌝) -∗
   {(R :: Δ); ↑Γ} ⊨
     (let: "b" := v1 in ¬ "b")
   ≾
     (let: "n" := v2 in if: "n" = #0
          then #1 else #0) :
   TVar 0)
```

DEMO

```
Lemma bit_refinement Δ Γ :
  {Δ;Γ} ⊨ bit_bool ≾ bit_nat : bitT.
 Proof.
  unlock bit_bool bit_nat; simpl.
  iApply (bin_log_related_pack _ R).
  repeat iApply bin_log_related_pair.
  - rel_finish.
  - rel_arrow_val.  simpl.
    iIntros "!#" (v1 v2).







Qed.
```

```
Δ : list D
Γ : stringmap type
==================================
□ (∃ b : bool, ⌜v1 = #b⌝ ∗ ⌜v2 = #(
    f b)⌝) −∗
{(R :: Δ); ↑Γ} ⊨
  (let: "b" := v1 in ¬ "b")
≾
  (let: "n" := v2 in if: "n" = #0 then
      #1 else #0) :
TVar 0
```

DEMO

```
Lemma bit_refinement Δ Γ :
  {Δ;Γ} ⊨ bit_bool ≾ bit_nat : bitT.
 Proof.
  unlock bit_bool bit_nat; simpl.
  iApply (bin_log_related_pack _ R).
  repeat iApply bin_log_related_pair.
  - rel_finish.
  - rel_arrow_val.  simpl.
    iIntros "!#" (v1 v2).
    iIntros ([b [?  ?]]);simplify_eq/=.

Qed.
```

```
Δ : list D
Γ : stringmap type
==================================
□ (∃ b : bool, ⌜ v1 = #b ⌝ ∗ ⌜ v2 = #(
    f b) ⌝) -∗
{(R :: Δ); ↑Γ} ⊨
  (let: "b" := v1 in ¬ "b")
≾
  (let: "n" := v2 in if: "n" = #0 then
      #1 else #0) :
TVar 0
```

DEMO

```
Lemma bit_refinement Δ Γ :
  {Δ;Γ} ⊨ bit_bool ≾ bit_nat : bitT.
 Proof.
  unlock bit_bool bit_nat; simpl.
  iApply (bin_log_related_pack _ R).
  repeat iApply bin_log_related_pair.
  - rel_finish.
  - rel_arrow_val.  simpl.
    iIntros "!#" (v1 v2).
    iIntros ([b [?  ?]]);simplify_eq/=.




Qed.
```

```
Δ : list D
Γ : stringmap type
b : bool
==============================
{(R :: Δ); ↑Γ} ⊨
  (let: "b" := #b in ¬ "b")
≾
  (let: "n" := #(f b) in if: "n" = #0
       then #1 else #0) :
TVar 0
```

DEMO

```
Lemma bit_refinement Δ Γ :
  {Δ;Γ} ⊨ bit_bool ≾ bit_nat : bitT.
 Proof.
  unlock bit_bool bit_nat; simpl.
  iApply (bin_log_related_pack _ R).
  repeat iApply bin_log_related_pair.
  - rel_finish.
  - rel_arrow_val.  simpl.
    iIntros "!#" (v1 v2).
    iIntros ([b [? ?]]);simplify_eq/=.
    rel_rec_l.
Qed.
```

```
Δ : list D
Γ : stringmap type
b : bool
================================
{(R :: Δ); ↑Γ} ⊨
  (let: "b" := #b in ¬ "b")
≾
  (let: "n" := #(f b) in if: "n" = #0
       then #1 else #0) :
TVar 0
```

DEMO

```
Lemma bit_refinement Δ Γ :
  {Δ;Γ} ⊨ bit_bool ≾ bit_nat : bitT.
 Proof.
  unlock bit_bool bit_nat; simpl.
  iApply (bin_log_related_pack _ R).
  repeat iApply bin_log_related_pair.
  - rel_finish.
  - rel_arrow_val.  simpl.
    iIntros "!#" (v1 v2).
    iIntros ([b [?  ?]]);simplify_eq/=.
    rel_rec_l.



Qed.
```

```
Δ : list D
Γ : stringmap type
b : bool
════════════════════════════════
{(R :: Δ); ↑Γ} ⊨
  (¬ #b)
≾
  (let: "n" := #(f b) in if: "n" = #0
       then #1 else #0) :
TVar 0
```

DEMO

```
Lemma bit_refinement Δ Γ :
  {Δ;Γ} ⊨ bit_bool ≾ bit_nat : bitT.
 Proof.
  unlock bit_bool bit_nat; simpl.
  iApply (bin_log_related_pack _ R).
  repeat iApply bin_log_related_pair.
  - rel_finish.
  - rel_arrow_val.  simpl.
    iIntros "!#" (v1 v2).
    iIntros ([b [?  ?]]);simplify_eq/=.
    rel_rec_l. rel_rec_r.

Qed.
```

```
Δ : list D
Γ : stringmap type
b : bool
================================
{(R :: Δ); ↑Γ} ⊨
  (¬ #b)
≾
  (let: "n" := #(f b) in if: "n" = #0
       then #1 else #0) :
TVar 0
```

DEMO

```
Lemma bit_refinement Δ Γ :
  {Δ;Γ} ⊨ bit_bool ≾ bit_nat : bitT.
 Proof.
  unlock bit_bool bit_nat; simpl.
  iApply (bin_log_related_pack _ R).
  repeat iApply bin_log_related_pair.
  - rel_finish.
  - rel_arrow_val.  simpl.
    iIntros "!#" (v1 v2).
    iIntros ([b [?  ?]]);simplify_eq/=.
    rel_rec_l. rel_rec_r.




Qed.
```

```
Δ : list D
Γ : stringmap type
b : bool
══════════════════════════════
{(R :: Δ); ↑Γ} ⊨
  (¬ #b)
≾
  (if: #(f b) = #0 then #1 else #0) :
      TVar 0
```

DEMO

```
Lemma bit_refinement Δ Γ :
  {Δ;Γ} ⊨ bit_bool ≾ bit_nat : bitT.
 Proof.
  unlock bit_bool bit_nat; simpl.
  iApply (bin_log_related_pack _ R).
  repeat iApply bin_log_related_pair.
  - rel_finish.
  - rel_arrow_val.  simpl.
    iIntros "!#" (v1 v2).
    iIntros ([b [?  ?]]);simplify_eq/=.
    rel_rec_l. rel_rec_r.
    rel_op_l.




Qed.
```

```
Δ : list D
Γ : stringmap type
b : bool
================================
{(R :: Δ); ↑Γ} ⊨
  (¬ #b)
≾
  (if: #(f b) = #0 then #1 else #0) :
       TVar 0
```

DEMO

```
Lemma bit_refinement Δ Γ :
  {Δ;Γ} ⊨ bit_bool ≾ bit_nat : bitT.
 Proof.
  unlock bit_bool bit_nat; simpl.
  iApply (bin_log_related_pack _ R).
  repeat iApply bin_log_related_pair.
  - rel_finish.
  - rel_arrow_val.  simpl.
    iIntros "!#" (v1 v2).
    iIntros ([b [?  ?]]);simplify_eq/=.
    rel_rec_l. rel_rec_r.
    rel_op_l.




Qed.
```

```
Δ : list D
Γ : stringmap type
b : bool
================================
{(R :: Δ); ↑Γ} ⊨
  #(xorb b true)
≾
  (if: #(f b) = #0 then #1 else #0) :
      TVar 0
```

DEMO

```
Lemma bit_refinement Δ Γ :
  {Δ;Γ} ⊨ bit_bool ≾ bit_nat : bitT.
 Proof.
  unlock bit_bool bit_nat; simpl.
  iApply (bin_log_related_pack _ R).
  repeat iApply bin_log_related_pair.
  - rel_finish.
  - rel_arrow_val.  simpl.
    iIntros "!#" (v1 v2).
    iIntros ([b [?  ?]]);simplify_eq/=.
    rel_rec_l. rel_rec_r.
    rel_op_l. rel_op_r.

Qed.
```

```
Δ : list D
Γ : stringmap type
b : bool
______________________________________
{(R :: Δ); ↑Γ} ⊨
  #(xorb b true)
≾
  (if: #(f b) = #0 then #1 else #0) :
       TVar 0
```

DEMO

```
Lemma bit_refinement Δ Γ :
  {Δ;Γ} ⊨ bit_bool ≾ bit_nat : bitT.
 Proof.
  unlock bit_bool bit_nat; simpl.
  iApply (bin_log_related_pack _ R).
  repeat iApply bin_log_related_pair.
  - rel_finish.
  - rel_arrow_val.  simpl.
    iIntros "!#" (v1 v2).
    iIntros ([b [?  ?]]);simplify_eq/=.
    rel_rec_l. rel_rec_r.
    rel_op_l. rel_op_r.

Qed.
```

```
Δ : list D
Γ : stringmap type
b : bool
==============================
{(R :: Δ); ↑Γ} ⊨
  #(xorb b true)
≾
  (if: #(if decide (f b = 0) then true
        else false) then #1 else #0) :
: TVar 0
```

DEMO

```
Lemma bit_refinement Δ Γ :
  {Δ;Γ} ⊨ bit_bool ≾ bit_nat : bitT.
 Proof.
  unlock bit_bool bit_nat; simpl.
  iApply (bin_log_related_pack _ R).
  repeat iApply bin_log_related_pair.
  - rel_finish.
  - rel_arrow_val.  simpl.
    iIntros "!#" (v1 v2).
    iIntros ([b [?  ?]]);simplify_eq/=.
    rel_rec_l. rel_rec_r.
    rel_op_l. rel_op_r.
    destruct b; simpl; rel_if_r;
      rel_finish.

Qed.
```

```
Δ : list D
Γ : stringmap type
b : bool
================================
{(R :: Δ); ↑Γ} ⊨
  #(xorb b true)
≾
  (if: #(if decide (f b = 0) then true
       else false) then #1 else #0) :
: TVar 0
```

DEMO

```
Lemma bit_refinement Δ Γ :
  {Δ;Γ} ⊨ bit_bool ≾ bit_nat : bitT.
 Proof.
  unlock bit_bool bit_nat; simpl.
  iApply (bin_log_related_pack _ R).
  repeat iApply bin_log_related_pair.
  - rel_finish.
  - rel_arrow_val.  simpl.
    iIntros "!#" (v1 v2).
    iIntros ([b [?  ?]]);simplify_eq/=.
    rel_rec_l. rel_rec_r.
    rel_op_l. rel_op_r.
    destruct b; simpl; rel_if_r;
      rel_finish.

Qed.
```

```
This subproof is complete, but there
     are some unfocused goals.
Focus next goal with bullet -.
```

DEMO

```
Lemma bit_refinement Δ Γ :
  {Δ;Γ} ⊨ bit_bool ≾ bit_nat : bitT.
 Proof.
  unlock bit_bool bit_nat; simpl.
  iApply (bin_log_related_pack _ R).
  repeat iApply bin_log_related_pair.
  - rel_finish.
  - rel_arrow_val.  simpl.
    iIntros "!#" (v1 v2).
    iIntros ([b [?  ?]]);simplify_eq/=.
    rel_rec_l. rel_rec_r.
    rel_op_l. rel_op_r.
    destruct b; simpl; rel_if_r;
      rel_finish.
  - ...
Qed.
```

```
This subproof is complete, but there
     are some unfocused goals.
Focus next goal with bullet -.
```

Rules

Some of the rules we have used in the proof:

$$\frac{(R :: \Delta); \uparrow \Gamma \models e \precsim e' : \tau}{\Delta; \Gamma \models \mathsf{pack}\ e \precsim \mathsf{pack}\ e' : \exists \tau}$$

Rules

Some of the rules we have used in the proof:

$$\frac{(R :: \Delta); \uparrow \Gamma \models e \precsim e' : \tau}{\Delta; \Gamma \models \mathsf{pack}\ e \precsim \mathsf{pack}\ e' : \exists \tau}$$

$$\frac{\Box(\forall v\, v',\ [\![\tau]\!]_\Delta(v, v') \twoheadrightarrow \Delta; \Gamma \models (\lambda x.e)\ v \precsim (\lambda x'.e')\ v' : \tau')}{\Delta; \Gamma \models \lambda x.e \precsim \lambda x'.e' : \tau \to \tau'}$$

Rules

Some of the rules we have used in the proof:

$$\frac{(R :: \Delta); \uparrow \Gamma \models e \precsim e' : \tau}{\Delta; \Gamma \models \mathsf{pack}\ e \precsim \mathsf{pack}\ e' : \exists \tau}$$

$$\frac{\Box(\forall v\, v',\ [\![\tau]\!]_\Delta(v, v') \mathrel{-\!\!*} \Delta; \Gamma \models (\lambda x.e)\ v \precsim (\lambda x'.e')\ v' : \tau')}{\Delta; \Gamma \models \lambda x.e \precsim \lambda x'.e' : \tau \to \tau'}$$

$$\frac{e \xrightarrow{\text{pure}} e' \qquad \Delta; \Gamma \models K[e'] \precsim t : \tau}{\Delta; \Gamma \models K[e] \precsim t : \tau}$$

Handling mutable state

Mutable state

We have seen an example refinement in the pure fragment of the language, but what about:

- *Mutable state*
- Concurrency

Use separation logic for managing resources for mutable state.

Use concurrent separation logic for managing resources for concurrency.

Rules and lemmas

Inference rules correspond to lemmas inside Iris:

Rule

$$\frac{l \mapsto_s v \quad * \quad (l \mapsto_s v' \twoheadrightarrow \Delta; \Gamma \models e \precsim K[()] : \tau)}{\Delta; \Gamma \models e \precsim K[l \leftarrow v'] : \tau}$$

Corresponding Coq lemma

```
Lemma bin_log_related_store_r Δ Γ K l e e' v v' τ :
  to_val e' = Some v' →
  l ↦ₛ v -∗
  (l ↦ₛ v' -∗ {Δ;Γ} ⊨ e ≾ fill K (#()) : τ) -∗
  {Δ;Γ} ⊨ e ≾ fill K (#l ← e') : τ.
```

Rules for load

```
Lemma bin_log_related_load_l' Δ Γ K l v t τ :
  l ↦ᵢ v -∗
  (l ↦ᵢ v -∗ ({Δ;Γ} ⊨ fill K (of_val v) ≾ t : τ)) -∗
  {Δ;Γ} ⊨ fill K !#l ≾ t : τ.
```

```
Lemma bin_log_related_load_r Δ Γ K l v t τ :
  l ↦ₛ v -∗
  (l ↦ₛ v -∗ {Δ;Γ} ⊨ t ≾ fill K (of_val v) : τ) -∗
  {Δ;Γ} ⊨ t ≾ fill K (! #l)) : τ.
```

Demo

```
Lemma test_goal Δ Γ (l k : loc) :
 l ↦ᵢ #1 -∗ k ↦ₛ #0 -∗
 {Δ;Γ} ⊨ !#l ≾ (#k ← #1;; !#k) : TNat.
Proof.



Qed.
```

```
Δ : list D
Γ : stringmap type
l, k : loc
================================
l ↦ᵢ #1 -∗ k ↦ₛ #0 -∗
{Δ;Γ} ⊨ ! #l ≾ (#k ← #1;; ! #k) : TNat
```

Demo

```
Lemma test_goal Δ Γ (l k : loc) :
 l ↦ᵢ #1 -∗ k ↦ₛ #0 -∗
 {Δ;Γ} ⊨ !#l ≾ (#k ← #1;; !#k) : TNat.
Proof.
 iIntros "Hl Hk".



Qed.
```

```
Δ : list D
Γ : stringmap type
l, k : loc
══════════════════════════════════
l ↦ᵢ #1 -∗ k ↦ₛ #0 -∗
{Δ;Γ} ⊨ ! #l ≾ (#k ← #1;; ! #k) : TNat
```

Demo

```
Lemma test_goal Δ Γ (l k : loc) :
 l ↦ᵢ #1 -∗ k ↦ₛ #0 -∗
 {Δ;Γ} ⊨ !#l ≾ (#k ← #1;; !#k) : TNat.
Proof.
 iIntros "Hl Hk".


Qed.
```

```
Δ : list D
Γ : stringmap type
l, k : loc
==================================
"Hl" : l ↦ᵢ #1
"Hk" : k ↦ₛ #0
--------------------------------------∗
{Δ;Γ} ⊨ ! #l ≾ (#k ← #1;; ! #k) : TNat
```

Demo

```
Lemma test_goal Δ Γ (l k : loc) :
 l ↦ᵢ #1 -∗ k ↦ₛ #0 -∗
 {Δ;Γ} ⊨ !#l ≾ (#k ← #1;; !#k) : TNat.
Proof.
 iIntros "Hl Hk".
 rel_store_r.


Qed.
```

```
Δ : list D
Γ : stringmap type
l, k : loc
==================================
"Hl" : l ↦ᵢ #1
"Hk" : k ↦ₛ #0
--------------------------------------∗
{Δ;Γ} ⊨ ! #l ≾ (#k ← #1;; ! #k) : TNat
```

Demo

```
Lemma test_goal Δ Γ (l k : loc) :
 l ↦ᵢ #1 -∗ k ↦ₛ #0 -∗
 {Δ;Γ} ⊨ !#l ≾ (#k ← #1;; !#k) : TNat.
Proof.
 iIntros "Hl Hk".
 rel_store_r.


Qed.
```

```
Δ : list D
Γ : stringmap type
l, k : loc
==================================
"Hl" : l ↦ᵢ #1
"Hk" : k ↦ₛ #1
-----------------------------------------∗
{Δ;Γ} ⊨ ! #l ≾ (#();; ! #k) : TNat
```

Demo

```
Lemma test_goal Δ Γ (l k : loc) :
 l ↦ᵢ #1 -∗ k ↦ₛ #0 -∗
 {Δ;Γ} ⊨ !#l ≾ (#k ← #1;; !#k) : TNat.
Proof.
 iIntros "Hl Hk".
 rel_store_r. rel_seq_r.


Qed.
```

```
Δ : list D
Γ : stringmap type
l, k : loc
=================================
"Hl" : l ↦ᵢ #1
"Hk" : k ↦ₛ #1
-----------------------------------------∗
{Δ;Γ} ⊨ ! #l ≾ (#();; ! #k) : TNat
```

Demo

```
Lemma test_goal Δ Γ (l k : loc) :
 l ↦ᵢ #1 -∗ k ↦ₛ #0 -∗
 {Δ;Γ} ⊨ !#l ≾ (#k ← #1;; !#k) : TNat.
Proof.
 iIntros "Hl Hk".
 rel_store_r. rel_seq_r.


Qed.
```

```
Δ : list D
Γ : stringmap type
l, k : loc
==============================
"Hl" : l ↦ᵢ #1
"Hk" : k ↦ₛ #1
--------------------------------------∗
{Δ;Γ} ⊨ ! #l ≾ ! #k : TNat
```

Demo

```
Lemma test_goal Δ Γ (l k : loc) :
 l ↦ᵢ #1 -∗ k ↦ₛ #0 -∗
 {Δ;Γ} ⊨ !#l ≾ (#k ← #1;; !#k) : TNat.
Proof.
 iIntros "Hl Hk".
 rel_store_r. rel_seq_r.
 rel_load_l.

Qed.
```

```
Δ : list D
Γ : stringmap type
l, k : loc
================================
"Hl" : l ↦ᵢ #1
"Hk" : k ↦ₛ #1
--------------------------------------∗
{Δ;Γ} ⊨ ! #l ≾ ! #k : TNat
```

Demo

```
Lemma test_goal Δ Γ (l k : loc) :
 l ↦ᵢ #1 -∗ k ↦ₛ #0 -∗
 {Δ;Γ} ⊨ !#l ≾ (#k ← #1;; !#k) : TNat.
Proof.
 iIntros "Hl Hk".
 rel_store_r. rel_seq_r.
 rel_load_l.

Qed.
```

```
Δ : list D
Γ : stringmap type
l, k : loc
================================
"Hl" : l ↦ᵢ #1
"Hk" : k ↦ₛ #1
--------------------------------------
{Δ;Γ} ⊨ #1 ≾ ! #k : TNat
```

Demo

```
Lemma test_goal Δ Γ (l k : loc) :
 l ↦ᵢ #1 -∗ k ↦ₛ #0 -∗
 {Δ;Γ} ⊨ !#l ≾ (#k ← #1;; !#k) : TNat.
Proof.
 iIntros "Hl Hk".
 rel_store_r. rel_seq_r.
 rel_load_l. rel_load_r.

Qed.
```

```
Δ : list D
Γ : stringmap type
l, k : loc
══════════════════════════════
"Hl" : l ↦ᵢ #1
"Hk" : k ↦ₛ #1
──────────────────────────────────────
{Δ;Γ} ⊨ #1 ≾ ! #k : TNat
```

Demo

```
Lemma test_goal Δ Γ (l k : loc) :
 l ↦ᵢ #1 -∗ k ↦ₛ #0 -∗
 {Δ;Γ} ⊨ !#l ≾ (#k ← #1;; !#k) : TNat.
Proof.
 iIntros "Hl Hk".
 rel_store_r. rel_seq_r.
 rel_load_l. rel_load_r.

Qed.
```

```
Δ : list D
Γ : stringmap type
l, k : loc
══════════════════════════
"Hl" : l ↦ᵢ #1
"Hk" : k ↦ₛ #1
──────────────────────────────────
{Δ;Γ} ⊨ #1 ≾ #1 : TNat
```

Demo

```
Lemma test_goal Δ Γ (l k : loc) :
 l ↦ᵢ #1 -∗ k ↦ₛ #0 -∗
 {Δ;Γ} ⊨ !#l ≾ (#k ← #1;; !#k) : TNat.
Proof.
 iIntros "Hl Hk".
 rel_store_r. rel_seq_r.
 rel_load_l. rel_load_r.
 iApply bin_log_related_nat.
Qed.
```

```
Δ : list D
Γ : stringmap type
l, k : loc
══════════════════════════════
"Hl" : l ↦ᵢ #1
"Hk" : k ↦ₛ #1
──────────────────────────────────────
{Δ;Γ} ⊨ #1 ≾ #1 : TNat
```

Demo

```
Lemma test_goal Δ Γ (l k : loc) :
 l ↦ᵢ #1 -∗ k ↦ₛ #0 -∗
 {Δ;Γ} ⊨ !#l ≾ (#k ← #1;; !#k) : TNat.
Proof.
 iIntros "Hl Hk".
 rel_store_r. rel_seq_r.
 rel_load_l. rel_load_r.
 iApply bin_log_related_nat.
Qed.
```

```
No more subgoals.
```

Example 2: higher-order functions with state

Example: HO stateful refinement

```
Lemma higher_order_stateful Δ Γ :
  {Δ;Γ} ⊨
   let: "x" := ref #1 in
   (λ: "f", "f" #();; !"x")
  ≾
   (λ: "f", "f" #();; #1)
  : ((Unit → Unit) → TNat).
Proof.

Qed.
```

```
Δ : list D
Γ : stringmap type
============================ (1/1)
{Δ; Γ} ⊨
  (let: "x" := ref #1 in λ: "f", "f" #();; ! "x")
≾
  (λ: "f", "f" #();; #1) : ((Unit → Unit) → TNat)
```

Example: HO stateful refinement

```
Lemma higher_order_stateful Δ Γ :
  {Δ;Γ} ⊨
   let: "x" := ref #1 in
   (λ: "f", "f" #();; !"x")
  ≾
   (λ: "f", "f" #();; #1)
  : ((Unit → Unit) → TNat).
Proof.
  rel_alloc_l as l "Hl".
Qed.
```

```
Δ : list D
Γ : stringmap type
============================ (1/1)
{Δ; Γ} ⊨
  (let: "x" := ref #1 in λ: "f", "f" #();; ! "x")
≾
  (λ: "f", "f" #();; #1) : ((Unit → Unit) → TNat)
```

Example: HO stateful refinement

```
Lemma higher_order_stateful Δ Γ :
  {Δ;Γ} ⊨
   let: "x" := ref #1 in
   (λ: "f", "f" #();; !"x")
  ≾
   (λ: "f", "f" #();; #1)
  : ((Unit → Unit) → TNat).
Proof.
  rel_alloc_l as l "Hl".




Qed.
```

```
Δ : list D
Γ : stringmap type
l : loc
============================ (1/1)
"Hl" : l ↦ᵢ #1
--------------------------------------∗
{Δ; Γ} ⊨
  (let: "x" := #l in λ: "f", "f" #();; ! "x")
≾
  (λ: "f", "f" #();; #1) : ((Unit → Unit) → TNat)
```

Example: HO stateful refinement

```
Lemma higher_order_stateful Δ Γ :
  {Δ;Γ} ⊨
   let: "x" := ref #1 in
   (λ: "f", "f" #();; !"x")
  ≾
   (λ: "f", "f" #();; #1)
  : ((Unit → Unit) → TNat).
Proof.
  rel_alloc_l as l "Hl".
  rel_let_l.



Qed.
```

```
Δ : list D
Γ : stringmap type
l : loc
============================== (1/1)
"Hl" : l ↦ᵢ #1
--------------------------------------∗
{Δ; Γ} ⊨
  (let: "x" := #l in λ: "f", "f" #();; ! "x")
≾
  (λ: "f", "f" #();; #1) : ((Unit → Unit) → TNat)
```

Example: HO stateful refinement

```
Lemma higher_order_stateful Δ Γ :
  {Δ;Γ} ⊨
   let: "x" := ref #1 in
   (λ: "f", "f" #();; !"x")
  ≾
   (λ: "f", "f" #();; #1)
  : ((Unit → Unit) → TNat).
Proof.
  rel_alloc_l as l "Hl".
  rel_let_l.

Qed.
```

```
Δ : list D
Γ : stringmap type
l : loc
============================== (1/1)
"Hl" : l ↦ᵢ #1
--------------------------------------∗
{Δ; Γ} ⊨
  (λ: "f", "f" #();; ! #l)
≾
  (λ: "f", "f" #();; #1) : ((Unit → Unit) → TNat)
```

Example: HO stateful refinement

```
Lemma higher_order_stateful Δ Γ :
  {Δ;Γ} ⊨
   let: "x" := ref #1 in
   (λ: "f", "f" #();; !"x")
  ≾
   (λ: "f", "f" #();; #1)
  : ((Unit → Unit) → TNat).
Proof.
  rel_alloc_l as l "Hl".
  rel_let_l.
  iMod (inv_alloc N _ (l ↦ᵢ #1)%I with "Hl")
    as "#Hinv".


Qed.
```

```
Δ : list D
Γ : stringmap type
l : loc
============================== (1/1)
"Hl" : l ↦ᵢ #1
--------------------------------------∗
{Δ; Γ} ⊨
  (λ: "f", "f" #();; ! #l)
≾
  (λ: "f", "f" #();; #1) : ((Unit → Unit) → TNat)
```

Example: HO stateful refinement

```
Lemma higher_order_stateful Δ Γ :
  {Δ;Γ} ⊨
   let: "x" := ref #1 in
   (λ: "f", "f" #();; !"x")
  ≾
   (λ: "f", "f" #();; #1)
  : ((Unit → Unit) → TNat).
Proof.
  rel_alloc_l as l "Hl".
  rel_let_l.
  iMod (inv_alloc N _ (l ↦ᵢ #1)%I with "Hl")
    as "#Hinv".
Qed.
```

```
Δ : list D
Γ : stringmap type
l : loc
============================ (1/1)
"Hinv" : inv N (l ↦ᵢ #1)%I
--------------------------------------□
{Δ; Γ} ⊨
  (λ: "f", "f" #();; ! #l)
≾
  (λ: "f", "f" #();; #1) : ((Unit → Unit) → TNat)
```

Example: HO stateful refinement

```
Lemma higher_order_stateful Δ Γ :
  {Δ;Γ} ⊨
   let: "x" := ref #1 in
   (λ: "f", "f" #();; !"x")
  ≾
   (λ: "f", "f" #();; #1)
  : ((Unit → Unit) → TNat).
Proof.
  rel_alloc_l as l "Hl".
  rel_let_l.
  iMod (inv_alloc N _ (l ↦ᵢ #1)%I with "Hl")
    as "#Hinv".
  rel_arrow.



Qed.
```

```
Δ : list D
Γ : stringmap type
l : loc
============================ (1/1)
"Hinv" : inv N (l ↦ᵢ #1)%I
--------------------------------------□
{Δ; Γ} ⊨
  (λ: "f", "f" #();; ! #l)
≾
  (λ: "f", "f" #();; #1) : ((Unit → Unit) → TNat)
```

Example: HO stateful refinement

```
Lemma higher_order_stateful Δ Γ :
  {Δ;Γ} ⊨
   let: "x" := ref #1 in
   (λ: "f", "f" #();; !"x")
  ≾
   (λ: "f", "f" #();; #1)
  : ((Unit → Unit) → TNat).
Proof.
  rel_alloc_l as l "Hl".
  rel_let_l.
  iMod (inv_alloc N _ (l ↦ᵢ #1)%I with "Hl")
    as "#Hinv".
  rel_arrow.




Qed.
```

```
Δ : list D
Γ : stringmap type
l : loc
============================ (1/1)
"Hinv" : inv N (l ↦ᵢ #1)%I
--------------------------------------□
□ (∀ v1 v2 : val,
   □ ({Δ; Γ} ⊨ v1 ≾ v2 : (Unit → Unit)) -∗
   {Δ; Γ} ⊨
     (let: "f" := v1 in "f" #();; ! #l)
   ≾
     (let: "f" := v2 in "f" #();; #1) : TNat)
```

Example: HO stateful refinement

```
Lemma higher_order_stateful Δ Γ :
  {Δ;Γ} ⊨
   let: "x" := ref #1 in
   (λ: "f", "f" #();; !"x")
  ≾
   (λ: "f", "f" #();; #1)
  : ((Unit → Unit) → TNat).
Proof.
  rel_alloc_l as l "Hl".
  rel_let_l.
  iMod (inv_alloc N _ (l ↦ᵢ #1)%I with "Hl")
    as "#Hinv".
  rel_arrow.
  iIntros "!#" (f1 f2) "#Hf".
Qed.
```

```
Δ : list D
Γ : stringmap type
l : loc
============================(1/1)
"Hinv" : inv N (l ↦ᵢ #1)%I
--------------------------------------□
□ (∀ v1 v2 : val,
   □ ({Δ; Γ} ⊨ v1 ≾ v2 : (Unit → Unit)) -∗
  {Δ; Γ} ⊨
    (let: "f" := v1 in "f" #();; ! #l)
  ≾
    (let: "f" := v2 in "f" #();; #1) : TNat)
```

Example: HO stateful refinement

```
Lemma higher_order_stateful Δ Γ :
  {Δ;Γ} ⊨
   let: "x" := ref #1 in
   (λ: "f", "f" #();; !"x")
  ≾
   (λ: "f", "f" #();; #1)
  : ((Unit → Unit) → TNat).
Proof.
  rel_alloc_l as l "Hl".
  rel_let_l.
  iMod (inv_alloc N _ (l ↦ᵢ #1)%I with "Hl")
    as "#Hinv".
  rel_arrow.
  iIntros "!#" (f1 f2) "#Hf".


Qed.
```

```
Δ : list D
Γ : stringmap type
l : loc
f1, f2 : val
============================ (1/1)
"Hinv" : inv N (l ↦ᵢ #1)%I
"Hf" : {Δ; Γ} ⊨ f1 ≾ f2 : (Unit → Unit)
--------------------------------------□
{Δ; Γ} ⊨
  (let: "f" := f1 in "f" #();; ! #l)
≾
  (let: "f" := f2 in "f" #();; #1) : TNat
```

Example: HO stateful refinement

```
Lemma higher_order_stateful Δ Γ :
  {Δ;Γ} ⊨
   let: "x" := ref #1 in
   (λ: "f", "f" #();; !"x")
  ≾
   (λ: "f", "f" #();; #1)
  : ((Unit → Unit) → TNat).
Proof.
  rel_alloc_l as l "Hl".
  rel_let_l.
  iMod (inv_alloc N _ (l ↦ᵢ #1)%I with "Hl")
    as "#Hinv".
  rel_arrow.
  iIntros "!#" (f1 f2) "#Hf".
  rel_let_l; rel_let_r.

Qed.
```

```
Δ : list D
Γ : stringmap type
l : loc
f1, f2 : val
============================ (1/1)
"Hinv" : inv N (l ↦ᵢ #1)%I
"Hf" : {Δ; Γ} ⊨ f1 ≾ f2 : (Unit → Unit)
--------------------------------------□
{Δ; Γ} ⊨
  (let: "f" := f1 in "f" #();; ! #l)
≾
  (let: "f" := f2 in "f" #();; #1) : TNat
```

Example: HO stateful refinement

```
Lemma higher_order_stateful Δ Γ :
  {Δ;Γ} ⊨
   let: "x" := ref #1 in
   (λ: "f", "f" #();; !"x")
  ≾
   (λ: "f", "f" #();; #1)
  : ((Unit → Unit) → TNat).
Proof.
  rel_alloc_l as l "Hl".
  rel_let_l.
  iMod (inv_alloc N _ (l ↦ᵢ #1)%I with "Hl")
    as "#Hinv".
  rel_arrow.
  iIntros "!#" (f1 f2) "#Hf".
  rel_let_l; rel_let_r.
Qed.
```

```
Δ : list D
Γ : stringmap type
l : loc
f1, f2 : val
============================== (1/1)
"Hinv" : inv N (l ↦ᵢ #1)%I
"Hf" : {Δ; Γ} ⊨ f1 ≾ f2 : (Unit → Unit)
--------------------------------------□
{Δ; Γ} ⊨ (f1 #();; ! #l) ≾ (f2 #();; #1) : TNat
```

Example: HO stateful refinement

```
Lemma higher_order_stateful Δ Γ :
  {Δ;Γ} ⊨
   let: "x" := ref #1 in
   (λ: "f", "f" #();; !"x")
  ≾
   (λ: "f", "f" #();; #1)
  : ((Unit → Unit) → TNat).
Proof.
  rel_alloc_l as l "Hl".
  rel_let_l.
  iMod (inv_alloc N _ (l ↦ᵢ #1)%I with "Hl")
    as "#Hinv".
  rel_arrow.
  iIntros "!#" (f1 f2) "#Hf".
  rel_let_l; rel_let_r.
  iApply bin_log_related_seq; auto.
Qed.
```

```
Δ : list D
Γ : stringmap type
l : loc
f1, f2 : val
============================== (1/1)
"Hinv" : inv N (l ↦ᵢ #1)%I
"Hf" : {Δ; Γ} ⊨ f1 ≾ f2 : (Unit → Unit)
--------------------------------------□
{Δ; Γ} ⊨ (f1 #();; ! #l) ≾ (f2 #();; #1) : TNat
```

Example: HO stateful refinement

```
Lemma higher_order_stateful Δ Γ :
  {Δ;Γ} ⊨
   let: "x" := ref #1 in
   (λ: "f", "f" #();; !"x")
  ≾
   (λ: "f", "f" #();; #1)
  : ((Unit → Unit) → TNat).
Proof.
  rel_alloc_l as l "Hl".
  rel_let_l.
  iMod (inv_alloc N _ (l ↦ᵢ #1)%I with "Hl")
    as "#Hinv".
  rel_arrow.
  iIntros "!#" (f1 f2) "#Hf".
  rel_let_l; rel_let_r.
  iApply bin_log_related_seq; auto.




Qed.
```

```
2 subgoals

Δ : list D
Γ : stringmap type
l : loc
f1, f2 : val
============================== (1/2)
"Hinv" : inv N (l ↦ᵢ #1)%I
"Hf" : {Δ; Γ} ⊨ f1 ≾ f2 : (Unit → Unit)
--------------------------------------□
{Δ; Γ} ⊨ f1 #() ≾ f2 #() : ?H7

Δ : list D
Γ : stringmap type
l : loc
f1, f2 : val
============================== (2/2)
"Hinv" : inv N (l ↦ᵢ #1)%I
"Hf" : {Δ; Γ} ⊨ f1 ≾ f2 : (Unit → Unit)
--------------------------------------□
{Δ; Γ} ⊨ ! #l ≾ #1 : TNat
```

Example: HO stateful refinement

```
Lemma higher_order_stateful Δ Γ :
  {Δ;Γ} ⊨
   let: "x" := ref #1 in
   (λ: "f", "f" #();; !"x")
  ≾
   (λ: "f", "f" #();; #1)
  : ((Unit → Unit) → TNat).
Proof.
  rel_alloc_l as l "Hl".
  rel_let_l.
  iMod (inv_alloc N _ (l ↦ᵢ #1)%I with "Hl")
    as "#Hinv".
  rel_arrow.
  iIntros "!#" (f1 f2) "#Hf".
  rel_let_l; rel_let_r.
  iApply bin_log_related_seq; auto.
  -
Qed.
```

```
Δ : list D
Γ : stringmap type
l : loc
f1, f2 : val
============================ (1/1)
"Hinv" : inv N (l ↦ᵢ #1)%I
"Hf" : {Δ; Γ} ⊨ f1 ≾ f2 : (Unit → Unit)
--------------------------------------□
{Δ; Γ} ⊨ f1 #() ≾ f2 #() : ?H7
```

Example: HO stateful refinement

```
Lemma higher_order_stateful Δ Γ :
  {Δ;Γ} ⊨
   let: "x" := ref #1 in
   (λ: "f", "f" #();; !"x")
  ≾
   (λ: "f", "f" #();; #1)
  : ((Unit → Unit) → TNat).
Proof.
  rel_alloc_l as l "Hl".
  rel_let_l.
  iMod (inv_alloc N _ (l ↦ᵢ #1)%I with "Hl")
    as "#Hinv".
  rel_arrow.
  iIntros "!#" (f1 f2) "#Hf".
  rel_let_l; rel_let_r.
  iApply bin_log_related_seq; auto.
  - iApply (bin_log_related_app with "Hf").
Qed.
```

```
Δ : list D
Γ : stringmap type
l : loc
f1, f2 : val
============================ (1/1)
"Hinv" : inv N (l ↦ᵢ #1)%I
"Hf" : {Δ; Γ} ⊨ f1 ≾ f2 : (Unit → Unit)
--------------------------------------□
{Δ; Γ} ⊨ f1 #() ≾ f2 #() : ?H7
```

Example: HO stateful refinement

```
Lemma higher_order_stateful Δ Γ :
  {Δ;Γ} ⊨
   let: "x" := ref #1 in
   (λ: "f", "f" #();; !"x")
  ≾
   (λ: "f", "f" #();; #1)
  : ((Unit → Unit) → TNat).
Proof.
  rel_alloc_l as l "Hl".
  rel_let_l.
  iMod (inv_alloc N _ (l ↦ᵢ #1)%I with "Hl")
    as "#Hinv".
  rel_arrow.
  iIntros "!#" (f1 f2) "#Hf".
  rel_let_l; rel_let_r.
  iApply bin_log_related_seq; auto.
  - iApply (bin_log_related_app with "Hf").
Qed.
```

```
Δ : list D
Γ : stringmap type
l : loc
f1, f2 : val
============================ (1/1)
"Hinv" : inv N (l ↦ᵢ #1)%I
"Hf" : {Δ; Γ} ⊨ f1 ≾ f2 : (Unit → Unit)
--------------------------------------□
{Δ; Γ} ⊨ #() ≾ #() : Unit
```

Example: HO stateful refinement

```
Lemma higher_order_stateful Δ Γ :
  {Δ;Γ} ⊨
   let: "x" := ref #1 in
   (λ: "f", "f" #();; !"x")
  ≾
   (λ: "f", "f" #();; #1)
  : ((Unit → Unit) → TNat).
Proof.
  rel_alloc_l as l "Hl".
  rel_let_l.
  iMod (inv_alloc N _ (l ↦ᵢ #1)%I with "Hl")
    as "#Hinv".
  rel_arrow.
  iIntros "!#" (f1 f2) "#Hf".
  rel_let_l; rel_let_r.
  iApply bin_log_related_seq; auto.
  - iApply (bin_log_related_app with "Hf").
    iApply bin_log_related_unit.



Qed.
```

```
Δ : list D
Γ : stringmap type
l : loc
f1, f2 : val
============================== (1/1)
"Hinv" : inv N (l ↦ᵢ #1)%I
"Hf" : {Δ; Γ} ⊨ f1 ≾ f2 : (Unit → Unit)
--------------------------------------□
{Δ; Γ} ⊨ #() ≾ #() : Unit
```

Example: HO stateful refinement

```
Lemma higher_order_stateful Δ Γ :
  {Δ;Γ} ⊨
   let: "x" := ref #1 in
   (λ: "f", "f" #();; !"x")
  ≾
   (λ: "f", "f" #();; #1)
  : ((Unit → Unit) → TNat).
Proof.
  rel_alloc_l as l "Hl".
  rel_let_l.
  iMod (inv_alloc N _ (l ↦ᵢ #1)%I with "Hl")
    as "#Hinv".
  rel_arrow.
  iIntros "!#" (f1 f2) "#Hf".
  rel_let_l; rel_let_r.
  iApply bin_log_related_seq; auto.
  - iApply (bin_log_related_app with "Hf").
    iApply bin_log_related_unit.
Qed.
```

```
This subproof is complete, but there are some
      unfocused goals.
Focus next goal with bullet -.
```

Example: HO stateful refinement

```
Lemma higher_order_stateful Δ Γ :
  {Δ;Γ} ⊨
   let: "x" := ref #1 in
   (λ: "f", "f" #();; !"x")
  ≾
   (λ: "f", "f" #();; #1)
  : ((Unit → Unit) → TNat).
Proof.
  rel_alloc_l as l "Hl".
  rel_let_l.
  iMod (inv_alloc N _ (l ↦ᵢ #1)%I with "Hl")
    as "#Hinv".
  rel_arrow.
  iIntros "!#" (f1 f2) "#Hf".
  rel_let_l; rel_let_r.
  iApply bin_log_related_seq; auto.
  - iApply (bin_log_related_app with "Hf").
    iApply bin_log_related_unit.
  -
Qed.
```

```
Δ : list D
Γ : stringmap type
l : loc
f1, f2 : val
============================ (1/1)
"Hinv" : inv N (l ↦ᵢ #1)%I
"Hf" : {Δ; Γ} ⊨ f1 ≾ f2 : (Unit → Unit)
--------------------------------------□
{Δ; Γ} ⊨ ! #l ≾ #1 : TNat
```

Example: HO stateful refinement

```
Lemma higher_order_stateful Δ Γ :
  {Δ;Γ} ⊨
   let: "x" := ref #1 in
   (λ: "f", "f" #();; !"x")
  ≾
   (λ: "f", "f" #();; #1)
  : ((Unit → Unit) → TNat).
Proof.
  rel_alloc_l as l "Hl".
  rel_let_l.
  iMod (inv_alloc N _ (l ↦ᵢ #1)%I with "Hl")
    as "#Hinv".
  rel_arrow.
  iIntros "!#" (f1 f2) "#Hf".
  rel_let_l; rel_let_r.
  iApply bin_log_related_seq; auto.
  - iApply (bin_log_related_app with "Hf").
    iApply bin_log_related_unit.
  - rel_load_l_atomic;
    iInv N as "Hl" "Hcl"; iModIntro.




Qed.
```

```
Δ : list D
Γ : stringmap type
l : loc
f1, f2 : val
============================ (1/1)
"Hinv" : inv N (l ↦ᵢ #1)%I
"Hf" : {Δ; Γ} ⊨ f1 ≾ f2 : (Unit → Unit)
--------------------------------------□
{Δ; Γ} ⊨ ! #l ≾ #1 : TNat
```

Example: HO stateful refinement

```
Lemma higher_order_stateful Δ Γ :
  {Δ;Γ} ⊨
   let: "x" := ref #1 in
   (λ: "f", "f" #();; !"x")
  ≾
   (λ: "f", "f" #();; #1)
  : ((Unit → Unit) → TNat).
Proof.
  rel_alloc_l as l "Hl".
  rel_let_l.
  iMod (inv_alloc N _ (l ↦ᵢ #1)%I with "Hl")
    as "#Hinv".
  rel_arrow.
  iIntros "!#" (f1 f2) "#Hf".
  rel_let_l; rel_let_r.
  iApply bin_log_related_seq; auto.
  - iApply (bin_log_related_app with "Hf").
    iApply bin_log_related_unit.
  - rel_load_l_atomic;
    iInv N as "Hl" "Hcl"; iModIntro.



Qed.
```

```
Δ : list D
Γ : stringmap type
l : loc
f1, f2 : val
============================== (1/1)
"Hinv" : inv N (l ↦ᵢ #1)%I
"Hf" : {Δ; Γ} ⊨ f1 ≾ f2 : (Unit → Unit)
--------------------------------------□
"Hl" : ▷ l ↦ᵢ #1
"Hcl" : ▷ l ↦ᵢ #1 ={⊤ \ ↑N,⊤}=∗ True
--------------------------------------∗
∃ v' : val,
▷ l ↦ᵢ v'
∗ ▷ (l ↦ᵢ v' -∗ {⊤ \ ↑N,⊤; Δ; Γ} ⊨ v' ≾ #1 : TNat)
```

Example: HO stateful refinement

```
Lemma higher_order_stateful Δ Γ :
  {Δ;Γ} ⊨
   let: "x" := ref #1 in
   (λ: "f", "f" #();; !"x")
  ≾
   (λ: "f", "f" #();; #1)
  : ((Unit → Unit) → TNat).
Proof.
  rel_alloc_l as l "Hl".
  rel_let_l.
  iMod (inv_alloc N _ (l ↦ᵢ #1)%I with "Hl")
    as "#Hinv".
  rel_arrow.
  iIntros "!#" (f1 f2) "#Hf".
  rel_let_l; rel_let_r.
  iApply bin_log_related_seq; auto.
  - iApply (bin_log_related_app with "Hf").
    iApply bin_log_related_unit.
  - rel_load_l_atomic;
    iInv N as "Hl" "Hcl"; iModIntro.
    iExists #1.  iNext.  iFrame "Hl".  simpl.




Qed.
```

```
Δ : list D
Γ : stringmap type
l : loc
f1, f2 : val
============================== (1/1)
"Hinv" : inv N (l ↦ᵢ #1)%I
"Hf" : {Δ; Γ} ⊨ f1 ≾ f2 : (Unit → Unit)
--------------------------------------□
"Hl" : ▷ l ↦ᵢ #1
"Hcl" : ▷ l ↦ᵢ #1 ={⊤ \ ↑N,⊤}=∗ True
--------------------------------------∗
∃ v' : val,
▷ l ↦ᵢ v'
∗ ▷ (l ↦ᵢ v' −∗ {⊤ \ ↑N,⊤; Δ; Γ} ⊨ v' ≾ #1 : TNat)
```

Example: HO stateful refinement

```
Lemma higher_order_stateful Δ Γ :
  {Δ;Γ} ⊨
   let: "x" := ref #1 in
   (λ: "f", "f" #();; !"x")
  ≾
   (λ: "f", "f" #();; #1)
  : ((Unit → Unit) → TNat).
Proof.
  rel_alloc_l as l "Hl".
  rel_let_l.
  iMod (inv_alloc N _ (l ↦ᵢ #1)%I with "Hl")
    as "#Hinv".
  rel_arrow.
  iIntros "!#" (f1 f2) "#Hf".
  rel_let_l; rel_let_r.
  iApply bin_log_related_seq; auto.
  - iApply (bin_log_related_app with "Hf").
    iApply bin_log_related_unit.
  - rel_load_l_atomic;
    iInv N as "Hl" "Hcl"; iModIntro.
    iExists #1.  iNext.  iFrame "Hl".  simpl.

Qed.
```

```
Δ : list D
Γ : stringmap type
l : loc
f1, f2 : val
============================== (1/1)
"Hinv" : inv N (l ↦ᵢ #1)%I
"Hf" : {Δ; Γ} ⊨ f1 ≾ f2 : (Unit → Unit)
--------------------------------------□
"Hcl" : ▷ l ↦ᵢ #1 ={⊤ \ ↑N,⊤}=∗ True
--------------------------------------∗
l ↦ᵢ #1 -∗ {⊤ \ ↑N,⊤; Δ; Γ} ⊨ #1 ≾ #1 : TNat
```

Example: HO stateful refinement

```
Lemma higher_order_stateful Δ Γ :
  {Δ;Γ} ⊨
   let: "x" := ref #1 in
   (λ: "f", "f" #();; !"x")
  ≾
   (λ: "f", "f" #();; #1)
  : ((Unit → Unit) → TNat).
Proof.
  rel_alloc_l as l "Hl".
  rel_let_l.
  iMod (inv_alloc N _ (l ↦ᵢ #1)%I with "Hl")
    as "#Hinv".
  rel_arrow.
  iIntros "!#" (f1 f2) "#Hf".
  rel_let_l; rel_let_r.
  iApply bin_log_related_seq; auto.
  - iApply (bin_log_related_app with "Hf").
    iApply bin_log_related_unit.
  - rel_load_l_atomic;
    iInv N as "Hl" "Hcl"; iModIntro.
    iExists #1.  iNext.  iFrame "Hl".  simpl.
    iIntros "Hl".


Qed.
```

```
Δ : list D
Γ : stringmap type
l : loc
f1, f2 : val
============================== (1/1)
"Hinv" : inv N (l ↦ᵢ #1)%I
"Hf" : {Δ; Γ} ⊨ f1 ≾ f2 : (Unit → Unit)
------------------------------------------□
"Hcl" : ▷ l ↦ᵢ #1 ={⊤ \ ↑N,⊤}=∗ True
------------------------------------------∗
l ↦ᵢ #1 -∗ {⊤ \ ↑N,⊤; Δ; Γ} ⊨ #1 ≾ #1 : TNat
```

Example: HO stateful refinement

```
Lemma higher_order_stateful Δ Γ :
  {Δ;Γ} ⊨
   let: "x" := ref #1 in
   (λ: "f", "f" #();; !"x")
  ≾
   (λ: "f", "f" #();; #1)
  : ((Unit → Unit) → TNat).
Proof.
  rel_alloc_l as l "Hl".
  rel_let_l.
  iMod (inv_alloc N _ (l ↦ᵢ #1)%I with "Hl")
    as "#Hinv".
  rel_arrow.
  iIntros "!#" (f1 f2) "#Hf".
  rel_let_l; rel_let_r.
  iApply bin_log_related_seq; auto.
  - iApply (bin_log_related_app with "Hf").
    iApply bin_log_related_unit.
  - rel_load_l_atomic;
    iInv N as "Hl" "Hcl"; iModIntro.
    iExists #1.  iNext.  iFrame "Hl".  simpl.
    iIntros "Hl".



Qed.
```

```
Δ : list D
Γ : stringmap type
l : loc
f1, f2 : val
============================== (1/1)
"Hinv" : inv N (l ↦ᵢ #1)%I
"Hf" : {Δ; Γ} ⊨ f1 ≾ f2 : (Unit → Unit)
--------------------------------------□
"Hcl" : ▷ l ↦ᵢ #1 ={⊤ \ ↑N,⊤}=∗ True
"Hl" : l ↦ᵢ #1
--------------------------------------∗
{⊤ \ ↑N,⊤; Δ; Γ} ⊨ #1 ≾ #1 : TNat
```

Example: HO stateful refinement

```
Lemma higher_order_stateful Δ Γ :
  {Δ;Γ} ⊨
   let: "x" := ref #1 in
   (λ: "f", "f" #();; !"x")
  ≾
   (λ: "f", "f" #();; #1)
  : ((Unit → Unit) → TNat).
Proof.
  rel_alloc_l as l "Hl".
  rel_let_l.
  iMod (inv_alloc N _ (l ↦ᵢ #1)%I with "Hl")
    as "#Hinv".
  rel_arrow.
  iIntros "!#" (f1 f2) "#Hf".
  rel_let_l; rel_let_r.
  iApply bin_log_related_seq; auto.
  - iApply (bin_log_related_app with "Hf").
    iApply bin_log_related_unit.
  - rel_load_l_atomic;
    iInv N as "Hl" "Hcl"; iModIntro.
    iExists #1.  iNext.  iFrame "Hl".  simpl.
    iIntros "Hl".
    iMod ("Hcl" with "Hl") as "_".

Qed.
```

```
Δ : list D
Γ : stringmap type
l : loc
f1, f2 : val
============================ (1/1)
"Hinv" : inv N (l ↦ᵢ #1)%I
"Hf" : {Δ; Γ} ⊨ f1 ≾ f2 : (Unit → Unit)
--------------------------------------□
"Hcl" : ▷ l ↦ᵢ #1 ={⊤ \ ↑N,⊤}=∗ True
"Hl" : l ↦ᵢ #1
--------------------------------------∗
{⊤ \ ↑N,⊤; Δ; Γ} ⊨ #1 ≾ #1 : TNat
```

Example: HO stateful refinement

```
Lemma higher_order_stateful Δ Γ :
  {Δ;Γ} ⊨
   let: "x" := ref #1 in
   (λ: "f", "f" #();; !"x")
  ≾
   (λ: "f", "f" #();; #1)
  : ((Unit → Unit) → TNat).
Proof.
  rel_alloc_l as l "Hl".
  rel_let_l.
  iMod (inv_alloc N _ (l ↦ᵢ #1)%I with "Hl")
    as "#Hinv".
  rel_arrow.
  iIntros "!#" (f1 f2) "#Hf".
  rel_let_l; rel_let_r.
  iApply bin_log_related_seq; auto.
  - iApply (bin_log_related_app with "Hf").
    iApply bin_log_related_unit.
  - rel_load_l_atomic;
    iInv N as "Hl" "Hcl"; iModIntro.
    iExists #1.  iNext.  iFrame "Hl".  simpl.
    iIntros "Hl".
    iMod ("Hcl" with "Hl") as "_".

Qed.
```

```
Δ : list D
Γ : stringmap type
l : loc
f1, f2 : val
============================ (1/1)
"Hinv" : inv N (l ↦ᵢ #1)%I
"Hf" : {Δ; Γ} ⊨ f1 ≾ f2 : (Unit → Unit)
--------------------------------------□
{Δ; Γ} ⊨ #1 ≾ #1 : TNat
```

Example: HO stateful refinement

```
Lemma higher_order_stateful Δ Γ :
  {Δ;Γ} ⊨
   let: "x" := ref #1 in
   (λ: "f", "f" #();; !"x")
  ≾
   (λ: "f", "f" #();; #1)
  : ((Unit → Unit) → TNat).
Proof.
  rel_alloc_l as l "Hl".
  rel_let_l.
  iMod (inv_alloc N _ (l ↦ᵢ #1)%I with "Hl")
    as "#Hinv".
  rel_arrow.
  iIntros "!#" (f1 f2) "#Hf".
  rel_let_l; rel_let_r.
  iApply bin_log_related_seq; auto.
  - iApply (bin_log_related_app with "Hf").
    iApply bin_log_related_unit.
  - rel_load_l_atomic;
    iInv N as "Hl" "Hcl"; iModIntro.
    iExists #1.  iNext.  iFrame "Hl".  simpl.
    iIntros "Hl".
    iMod ("Hcl" with "Hl") as "_".
    iApply bin_log_related_nat.
Qed.
```

```
Δ : list D
Γ : stringmap type
l : loc
f1, f2 : val
============================== (1/1)
"Hinv" : inv N (l ↦ᵢ #1)%I
"Hf" : {Δ; Γ} ⊨ f1 ≾ f2 : (Unit → Unit)
----------------------------------------□
{Δ; Γ} ⊨ #1 ≾ #1 : TNat
```

Example: HO stateful refinement

```
Lemma higher_order_stateful Δ Γ :
  {Δ;Γ} ⊨
   let: "x" := ref #1 in
   (λ: "f", "f" #();; !"x")
  ≾
   (λ: "f", "f" #();; #1)
  : ((Unit → Unit) → TNat).
Proof.
  rel_alloc_l as l "Hl".
  rel_let_l.
  iMod (inv_alloc N _ (l ↦ᵢ #1)%I with "Hl")
    as "#Hinv".
  rel_arrow.
  iIntros "!#" (f1 f2) "#Hf".
  rel_let_l; rel_let_r.
  iApply bin_log_related_seq; auto.
  - iApply (bin_log_related_app with "Hf").
    iApply bin_log_related_unit.
  - rel_load_l_atomic;
    iInv N as "Hl" "Hcl"; iModIntro.
    iExists #1.  iNext.  iFrame "Hl".  simpl.
    iIntros "Hl".
    iMod ("Hcl" with "Hl") as "_".
    iApply bin_log_related_nat.
Qed.
```

```
No more subgoals.
```

Case studies

Several examples that were formalized in the logic:

- Concurrent counter & Treiber stack
  (Done before in Iris, but now with cleaner and faster proofs).
- Ticket based lock refines spin lock.
- Algebraic laws for non-deterministic choice, *e.g.*,
  $\Gamma \vDash$ `or e1 e2` $\precsim$ `or e2 e1 : Unit`.
- Concurrent stateful ADTs, *e.g.*, a symbol lookup table with and without superfluous checks.
- Higher-order stateful functions in the presence of concurrency.

Thank you for listening!

Find the development online at

`https://gitlab.mpi-sws.org/dfrumin/logrel-conc`

Additional slide: overview of the development

- Small-step CBV semantics for the object programming language (untyped);
- Reified syntax for solving some properties by reflection (e.g. whether an expression is closed, is a value);
- Typing system for the object language and typed contextual refinement;
- Encoding of logical relations in Iris with a soundness proof;
- Proofs of the primitive rules (in the model) and of the derived rules (in the logic);
- Tactics and tactic lemmas for actually using the calculus.

Plus some intermediate developments such as the machinery for the weakest precondition calculus and ghost state