Relational Hoare Logic for Sequential Program Verification

Shushu Wu

Shanghai Jiao Tong University

POPL 2025

An Encoding Theory

Theorem 1 (Encoding Correctness)

For any low-level statement c^{L} and program-as-resource assertions $\mathcal{P}$ and $\mathcal{Q}$,

$$\langle\ \mathcal{P}\ \rangle\ c^{\mathrm{L}}\ \langle\ \mathcal{Q}\ \rangle \text{ is valid} \quad \textit{iff.} \quad \forall X \vdash_{\forall} \{\ (|\ \mathcal{P}\ |)_X\ \}\ c^{\mathrm{L}}\ \{\ (|\ \mathcal{Q}\ |)_X\ \}$$

Decomposed assertion encoding

- $(\sigma^{\mathrm{L}}, \sigma^{\mathrm{H}}, c^{\mathrm{H}}) \models \lfloor P^{\mathrm{L}} \rfloor$ *iff.* $\sigma^{\mathrm{L}} \models P^{\mathrm{L}}$
- $(\sigma^{\mathrm{L}}, \sigma^{\mathrm{H}}, c^{\mathrm{H}}) \models \lceil P^{\mathrm{H}} \rceil$ *iff.* $\sigma^{\mathrm{H}} \models P^{\mathrm{H}}$
- $(\sigma^{\mathrm{L}}, \sigma^{\mathrm{H}}, c^{\mathrm{H}}) \models [c_0^{\mathrm{H}}]$ *iff.* $c^{\mathrm{H}} = c_0^{\mathrm{H}}$

encode

- $(|\ \exists a.\, \mathcal{P}(a)\ |)_X \iff \exists a.\, (|\ \mathcal{P}(a)\ |)_X$
- $(|\ B \wedge \mathcal{P}\ |)_X \iff B \wedge (|\ \mathcal{P}\ |)_X$
- $(|\ \lfloor P^{\mathrm{L}} \rfloor \wedge \lceil P^{\mathrm{H}} \rceil \wedge [c^{\mathrm{H}}]\ |)_X \iff \mathsf{Exec}_X(P^{\mathrm{H}}, c^{\mathrm{H}}) \wedge P^{\mathrm{L}}$

Relational rules encoding

HIGH-FOCUS

$$\frac{\vdash_{\exists} \{P^{\mathrm{H}}\}\ c_1^{\mathrm{H}}\ \{R^{\mathrm{H}}\} \qquad \langle \lfloor F^{\mathrm{L}} \rfloor \wedge \lceil R^{\mathrm{H}} \rceil \wedge [c_2^{\mathrm{H}}] \rangle\ c^{\mathrm{L}}\ \langle \mathcal{Q} \rangle}{\langle \lfloor F^{\mathrm{L}} \rfloor \wedge \lceil P^{\mathrm{H}} \rceil \wedge [c_1^{\mathrm{H}}; c_2^{\mathrm{H}}] \rangle\ c^{\mathrm{L}}\ \langle \mathcal{Q} \rangle}$$

encode

$$\frac{\mathsf{Exec}_X(P^{\mathrm{H}}, c_1^{\mathrm{H}}; c_2^{\mathrm{H}}) \Rightarrow \mathsf{Exec}_X(R^{\mathrm{H}}, c_2^{\mathrm{H}}) \qquad \vdash_{\forall} \{\mathsf{Exec}_X(R^{\mathrm{H}}, c_2^{\mathrm{H}}) \wedge F^{\mathrm{L}}\}\ c^{\mathrm{L}}\ \{(\!|\mathcal{Q}|\!)_X\}}{\vdash_{\forall} \{\mathsf{Exec}_X(P^{\mathrm{H}}, c_1^{\mathrm{H}}; c_2^{\mathrm{H}}) \wedge F^{\mathrm{L}}\}\ c^{\mathrm{L}}\ \{(\!|\mathcal{Q}|\!)_X\}}$$

LOW-FOCUS

$$\frac{\vdash_{\forall} \{P^{\mathrm{L}}\}\ c_1^{\mathrm{L}}\ \{R^{\mathrm{L}}\} \qquad \langle \lfloor R^{\mathrm{L}} \rfloor \wedge \lceil P^{\mathrm{H}} \rceil \wedge [c^{\mathrm{H}}] \rangle\ c_2^{\mathrm{L}}\ \langle \mathcal{Q} \rangle}{\langle \lfloor P^{\mathrm{L}} \rfloor \wedge \lceil P^{\mathrm{H}} \rceil \wedge [c^{\mathrm{H}}] \rangle\ c_1^{\mathrm{L}}; c_2^{\mathrm{L}}\ \langle \mathcal{Q} \rangle}$$

encode

$$\frac{\vdash_{\forall} \{P^{\mathrm{L}}\}\ c_1^{\mathrm{L}}\ \{R^{\mathrm{L}}\} \qquad \vdash_{\forall} \{\mathsf{Exec}_X(P^{\mathrm{H}}, c^{\mathrm{H}}) \wedge R^{\mathrm{L}}\}\ c_2^{\mathrm{L}}\ \{(\!|\mathcal{Q}|\!)_X\}}{\vdash_{\forall} \{\mathsf{Exec}_X(P^{\mathrm{H}}, c^{\mathrm{H}}) \wedge P^{\mathrm{L}}\}\ c_1^{\mathrm{L}}; c_2^{\mathrm{L}}\ \{(\!|\mathcal{Q}|\!)_X\}}$$

Proof encoding

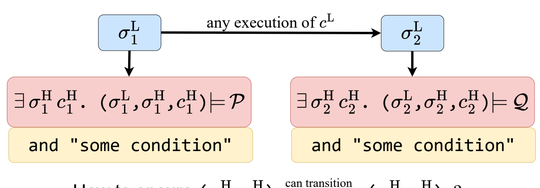
```
// ⟨[s = { }; s = s ∪ {a0}; ; s = s ∪ {a1}]⟩
// high-level step
// ⟨⌈s = ∅⌉ ∧ [s = s ∪ {a0}; s = s ∪ {a1}]⟩
x = 0;
// low-level step
// ⟨⌊x = 0⌋ ∧ ⌈s = ∅⌉ ∧ [s = s ∪ {a0}; s = s ∪ {a1}]⟩
x = x | (1 << a0);
// high-level and low-level steps
// ⟨⌊x = 2^a0⌋ ∧ ⌈s = {a0}⌉ ∧ [s = s ∪ {a1}]⟩
x = x | (1 << a1);
// high-level and low-level steps
// ⟨⌊x = 2^a0 + 2^a1⌋ ∧ ⌈s = {a0, a1}⌉ ∧ [skip]⟩
```

encode

```
// {Exec_X(True, s = { }; s = s ∪ {a0}; s = s ∪ {a1})}
// consequence rule
// {Exec_X(s = ∅, s = s ∪ {a0}; s = s ∪ {a1})}
x = 0;
// sequencing rule
// {Exec_X(s = ∅, s = s ∪ {a0}; s = s ∪ {a1}) ∧ x = 0}
x = x | (1 << a0);
// consequence and sequencing rules
// {Exec_X(s = {a0}, s = s ∪ {a1}) ∧ x = 2^a0}
x = x | (1 << a1);
// consequence and sequencing rules
// {Exec_X(s = {a0, a1}, skip) ∧ x = 2^a0 + 2^a1}
```

Intuition

- use existential quantification to embed high-level states
- employ program-as-resource assertions



How to ensure $(\sigma_1^{\mathrm{H}}, c_1^{\mathrm{H}}) \xrightarrow{\text{can transition}} (\sigma_2^{\mathrm{H}}, c_2^{\mathrm{H}})$?

Introduce a placeholder X !

Encoding Definition

- **assertion encoding**

  for any program-as-resource assertion $\mathcal{P}$, and subset of high-level states $X \subseteq \Sigma^{\mathrm{H}}$:

  $$(|\mathcal{P}|)_X \triangleq \lambda \sigma^{\mathrm{L}}.\ \exists\, \sigma^{\mathrm{H}}\, c^{\mathrm{H}}.\ (\sigma^{\mathrm{L}}, \sigma^{\mathrm{H}}, c^{\mathrm{H}}) \models \mathcal{P} \wedge \sigma^{\mathrm{H}} \models \mathsf{wp}(c^{\mathrm{H}}, X)$$

- **execution encoding**

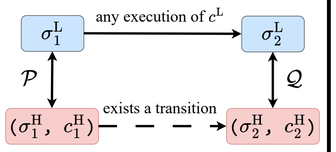
  for any high-level assertion $P^{\mathrm{H}} \subseteq \Sigma^{\mathrm{H}}$, subset of high-level states $X \subseteq \Sigma^{\mathrm{H}}$, and high-level program c^{H} :

  $$\mathsf{Exec}_X(P^{\mathrm{H}}, c^{\mathrm{H}}) \triangleq \mathsf{wp}(c^{\mathrm{H}}, X) \Rightarrow P^{\mathrm{H}}$$

Background

- program-as-resource assertions [2] $\mathcal{P} \subseteq \Sigma^{\mathrm{L}} \times \Sigma^{\mathrm{H}} \times \mathrm{Prog}^{\mathrm{H}}$
- relational Hoare triples $\langle\ \mathcal{P}\ \rangle\ c^{\mathrm{L}}\ \langle\ \mathcal{Q}\ \rangle$
- represent program refinement:

  $\langle\ \mathbb{P} \wedge [c^{\mathrm{H}}]\ \rangle\ c^{\mathrm{L}}\ \langle\ \mathbb{Q} \wedge [\mathsf{skip}]\ \rangle$



- decomposed program-as-resource assertions

  $\exists \vec{a}.\ B(\vec{a}) \wedge \lfloor P^{\mathrm{L}}(\vec{a}) \rfloor \wedge \lceil P^{\mathrm{H}}(\vec{a}) \rceil \wedge [c^{\mathrm{H}}]$

- functional correctness: algorithm correctness + implementation correctness

Related Work

- the $\forall\forall$ relational Hoare logic $\xrightarrow{\text{encode}}$ traditional Hoare logic ($\forall$) ✓
  - self-composition [1]
- the $\forall\exists$ relational Hoare logic $\xrightarrow{\text{encode}}$ a Hoare logic with ghost states ($\forall\exists$) ✓
  - program-as-resource [2]
- the $\forall\exists$ relational Hoare logic $\xrightarrow{\text{encode}}$ traditional Hoare logic ($\forall$) ?

Hybrid Reasoning

```
struct list *mergesort
        (struct list *x){
    int l = listlen(x);

    if (l < 8)
        return inssort(x);
    struct list *p, *q;
    p = NULL;
    q = NULL;
    split(x, &p, &q);
    p = mergesort(p);
    q = mergesort(q);
    return merge(p,q);
}
```

```
Def extsort_A (x : list Z) :=
    return r  st. permutation(x,r) ∧ sorted(r)

Def mergesort_A (x : list Z) :=
    choice(
        return extsort_A(x) ,
        test (Zlength x) ≥ 2) ;;
        (p1, q1) ← split_A(x, nil, nil) ;;
        p2 ← mergesort_A(p1) ;;
        q2 ← mergesort_A(q1) ;;
        merge_A(p2,q2);
    )
```

traditional hoare logic

functional correctness

$\{\ \mathsf{storelist}(\mathtt{x}, l)\ \}$
inssort$(\mathtt{x})$
$\{\ \exists l_r.\ \mathsf{permutation}(l, l_r) \wedge \mathsf{sorted}(l_r) \wedge \mathsf{storelist}(\mathtt{x}, l_r)\ \}$

Relational Nature

Example: insertion function of binary search trees

$\mathsf{storemap}(\mathtt{b}, m) \triangleq \exists\ (t\text{: tree}).\ \mathsf{storetree}(\mathtt{b}, t) \wedge t \sim m$

functional correctness

$\{\ \mathsf{storemap}(\mathtt{b}, m)\ \}$ **insert**$(\mathtt{b}, t, v)$ $\{\ \mathsf{storemap}(\mathtt{b}, m[k \mapsto v])\ \}$

algorithm correctness

$\forall t\, k\, v\, m.\ \mathsf{bst}(t) \wedge t \sim m \Rightarrow \mathsf{bst}(\mathbf{ins}(t, k, v)) \wedge \mathbf{ins}(t, k, v) \sim m[k \mapsto v]$

high-level functional correctness

$\{\ \mathsf{bst}(t) \wedge t \sim m\ \}$ $t :=$**ins**(t, k, v) $\{\ \mathsf{bst}(t) \wedge t \sim m[k \mapsto v]\ \}$

implementation correctness

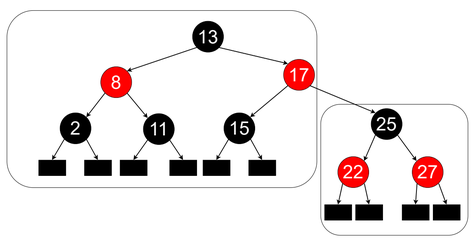
$\{\ \mathsf{storetree}(\mathtt{b}, t)$
insert$(\mathtt{b}, t, v)$
$\{\ \mathsf{storetree}(\mathtt{b}, \mathbf{ins}(t, k, v))\ \}$

program refinement

$\{\ \mathsf{Exec}_X(\mathrm{True}, t :=\mathbf{ins}(t, k, v))) \wedge \mathsf{storetree}(\mathtt{b}, t)\ \}$
insert$(\mathtt{b}, t, v)$
$\{\ \mathsf{Exec}_X(\mathrm{True}, \mathsf{skip}) \wedge \mathsf{storetree}(\mathtt{b}, t)\ \}$

```
void insert (struct tree **b, int k, int v){
    while (1) {
        ∃ (pt: partial tree) (t0: tree),
          pt ⊕ ins(t0, k, v) = ins(t, k, v) ∧
          storepartialtree(b, pt) ∗ storetree(b, t0)
        struct tree *p ; p = *b ;
        if (p == NULL) {
            p = malloc_tree_node() ;
            p → key = k ; p → value = v ;  p → left = NULL ; p → right = NULL ;
            *b = p ; return ;
        } else {
            int y ; y = p → key ;
            if (x < y) {  b = &p → left ;  } else if (y < x) { b = &p → right ;
            } else { p → value = v ; return ; }
        }
    }
}
```

∃ (*pt*: partial tree) (t_0: tree),
Exec_X(True,**ins_mid**$(pt, t_0, k, v)) \wedge$
$\mathsf{storepartialtree}(\mathtt{b}, pt) * \mathsf{storetree}(\mathtt{b}, t_0)$



[1] G. Barthe, P. R. D'Argenio and T. Rezk, "Secure information flow by self-composition," Proceedings. 17th IEEE Computer Security Foundations Workshop, 2004., Pacific Grove, CA, USA, 2004, pp. 100-114.
[2] Aaron J. Turon, Jacob Thamsborg, Amal Ahmed, Lars Birkedal, and Derek Dreyer. 2013. Logical relations for fine-grained concurrency. In Proceedings of the 40th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages (POPL '13). Association for Computing Machinery, New York, NY, USA, 343–356.