Conditional Contextual Refinement

Iris Workshop 2023

Youngju Song, Minki Cho, Dongjae Lee, Chung-Kil Hur

Michael Sammler, Derek Dreyer

MAX PLANCK INSTITUTE
FOR SOFTWARE SYSTEMS

Conditional Contextual Refinement

Iris Workshop 2023

Youngju Song, Minki Cho, Dongjae Lee, Chung-Kil Hur

VERI TAS LUX MEA

Michael Sammler, Derek Dreyer

MAX PLANCK INSTITUTE
FOR SOFTWARE SYSTEMS

Separation Logic vs. Refinement

Separation Logic vs. Refinement

$\{\, P \,\}\; c \;\{\, Q \,\}$

Specifications?

$I \sqsubseteq A$

Separation Logic vs. Refinement

$\{ P \}\, c\, \{ Q \}$

Specifications?

$I \sqsubseteq A$

$l_1 \mapsto v_1 * \cdots * l_n \mapsto v_n$

(De)Composition?

$I \sqsubseteq M_1 \sqsubseteq \cdots \sqsubseteq M_n \sqsubseteq A$

Separation Logic vs. Refinement

$\{\, P \,\}\; c \;\{\, Q \,\}$

$l_1 \mapsto v_1 * \cdots * l_n \mapsto v_n$

$\{\, P * FR \,\}\; c \;\{\, Q * FR \,\}$

Specifications?

(De)Composition?

Proof Reuse?

$I \sqsubseteq A$

$I \sqsubseteq \boxed{M_1 \sqsubseteq \cdots \sqsubseteq M_n \sqsubseteq A}$

$\rotatebox{90}{$\sqsubseteq$}$

$\boxed{I'}$

Separation Logic vs. Refinement

$\{\, P \,\}\; c \;\{\, Q \,\}$

Separation Logic

$I \sqsubseteq A$

Refinement

Separation Logic vs. Refinement

$\{\, P \,\}\; c \;\{\, Q \,\}$

Separation Logic

$I \sqsubseteq A$

Refinement

✓ **Conditional specifications**

for modular reasoning about shared state

Separation Logic vs. Refinement

$\{\, P \,\}\; c \;\{\, Q \,\}$

Separation Logic

$I \sqsubseteq A$

Refinement

✓ **Conditional specifications**

for modular reasoning about shared state

$$l_1 \mapsto v_1 * \cdots * l_n \mapsto v_n$$

$$\{\, P * FR \,\}\; c \;\{\, Q * FR \,\}$$

Separation Logic vs. Refinement

$\{\, P \,\}\; c \;\{\, Q \,\}$

Separation Logic

$I \sqsubseteq A$

Refinement

✓ **Conditional specifications**

Unconditional ✗

for modular reasoning about shared state

$$l_1 \mapsto v_1 * \cdots * l_n \mapsto v_n$$

$$\{\, P * FR \,\}\; c \;\{\, Q * FR \,\}$$

Separation Logic vs. Refinement

$$\{\, P \,\}\; c \;\{\, Q \,\}$$

Separation Logic

✓ **Conditional specifications**

for modular reasoning about shared state

$$I \sqsubseteq A$$

Refinement

Unconditional ✗

e.g., contextual refinement ($\sqsubseteq_{ctx}$) quantifies “completely arbitrary” context

Transitive composition ✓

(e.g., as seen in CertikOS)

$$I \sqsubseteq M_1 \sqsubseteq \cdots \sqsubseteq M_n \sqsubseteq A$$

$$I' \sqsubseteq M_k \quad \Longrightarrow \quad I' \sqsubseteq A$$

Separation Logic vs. Refinement

$$\{\,P\,\}\;c\;\{\,Q\,\}$$

Separation Logic

✓ **Conditional specifications**

for modular reasoning about shared state

✗ **No transitive composition**

$$I \sqsubseteq A$$

Refinement

✗ **Unconditional**

e.g., contextual refinement ($\sqsubseteq_{ctx}$) quantifies "completely arbitrary" context

✓ **Transitive composition**

(e.g., as seen in CertikOS)

$$I \sqsubseteq M_1 \sqsubseteq \cdots \sqsubseteq M_n \sqsubseteq A$$

$$I' \sqsubseteq M_k \quad \Longrightarrow \quad I' \sqsubseteq A$$

Separation Logic vs. Refinement

	Separation Logic	Refinement	
	$\{ P \} \; c \; \{ Q \}$	$I \sqsubseteq A$	
✓	Conditional specifications	Unconditional	✗
✗	No transitive composition	Transitive composition (e.g., as seen in CertikOS)	✓

Goal: have best of both worlds.

Wait… what about: Relational Separation Logic

$\{ P \}\, c\, \{ Q \}$

Separation Logic

✓ **Conditional specifications**

esp. modular reasoning on shared states

✗ **No transitive composition**

$I \sqsubseteq A$

Refinement

Unconditional ✗

e.g., contextual refinement ($\sqsubseteq_{ctx}$) quantifies “completely arbitrary” context

Transitive composition ✓

(e.g., as seen in CertikOS)

Wait… what about: Relational Separation Logic

$\{ P \}\ I \leq A\ \{ Q \}$

Relational Separation Logic

✓ **Conditional specifications**

esp. modular reasoning on shared states

✗ **No transitive composition**

$I \sqsubseteq A$

Refinement

✗ **Unconditional**

e.g., contextual refinement ($\sqsubseteq_{ctx}$) quantifies "completely arbitrary" context

✓ **Transitive composition**

(e.g., as seen in CertikOS)

Wait... what about: Relational Separation Logic

$$\{\,P\,\}\; I \le A \;\{\,Q\,\}$$

Relational Separation Logic

✓ **Conditional specifications**

esp. modular reasoning on shared states

✗ **No transitive composition**

$$I \sqsubseteq A$$

Refinement

Unconditional ✗

e.g., contextual refinement ($\sqsubseteq_{ctx}$) quantifies "completely arbitrary" context

Transitive composition ✓

(e.g., as seen in CertikOS)

Wait… what about: Relational Separation Logic

$\{\, P \,\}\; I \leq A \;\{\, Q \,\}$

Relational Separation Logic

✓ **Conditional specifications**

esp. modular reasoning on shared states

✗ **No transitive composition**

Adequacy: for certain P/Q…

$I \sqsubseteq A$

Refinement

Unconditional ✗

e.g., contextual refinement ($\sqsubseteq_{ctx}$) quantifies "completely arbitrary" context

Transitive composition ✓

(e.g., as seen in CertikOS)

Wait… what about: Relational Separation Logic

$\{ P \}\ I \leq A\ \{ Q \}$

Relational Separation Logic

Adequacy: for certain P/Q…

$I \sqsubseteq A$

Refinement

✓ **Conditional specifications**

esp. modular reasoning on shared states

✗ **No transitive composition**

Unconditional ✗

e.g., contextual refinement ($\sqsubseteq_{ctx}$) quantifies “completely arbitrary” context

Transitive composition ✓

(e.g., as seen in CertikOS)

Wait... what about: Relational Separation Logic

$\{ P \}\ I \leq A\ \{ Q \}$

Relational Separation Logic

Adequacy: for certain P/Q...

$I \sqsubseteq A$

Refinement

✓ **Conditional specifications**

Unconditional ✗

Benefits are kept **separate**!

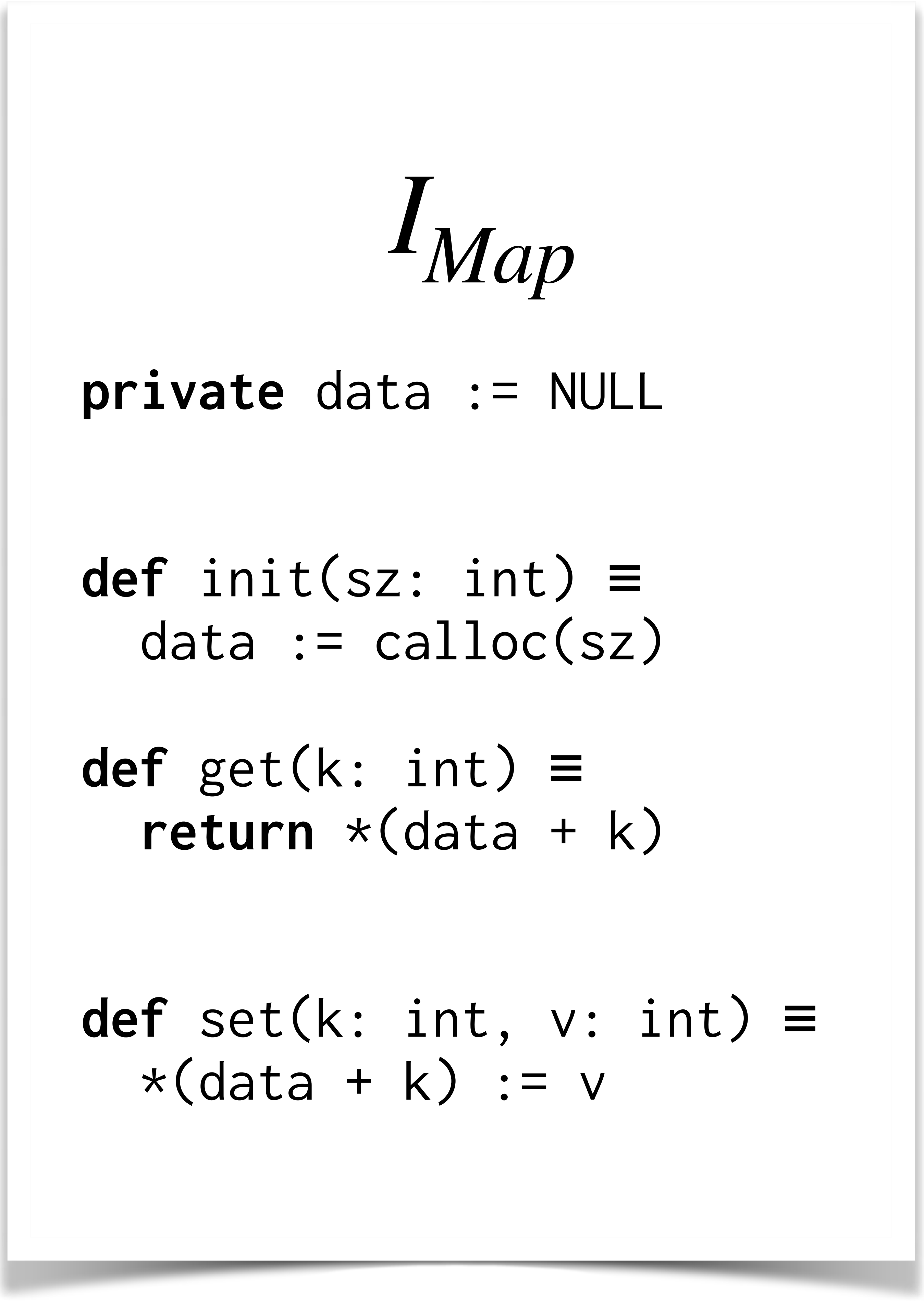
✗ **No transitive composition**

Transitive composition ✓

(e.g., as seen in CertikOS)

Motivating Example

Refinement alone is not enough



I_{Map}

```
private data := NULL


def init(sz: int) ≡
  data := calloc(sz)

def get(k: int) ≡
  return *(data + k)


def set(k: int, v: int) ≡
  *(data + k) := v
```

Motivating Example

Refinement alone is not enough

$$I_{Map}$$

```
private data := NULL


def init(sz: int) ≡
  data := calloc(sz)

def get(k: int) ≡
  return *(data + k)


def set(k: int, v: int) ≡
  *(data + k) := v
```

private (i.e., module-local) data:
completely hidden from outside.

Motivating Example

Refinement alone is not enough

$$I_{Map}$$

```
private data := NULL

def init(sz: int) ≡
  data := calloc(sz)

def get(k: int) ≡
  return *(data + k)


def set(k: int, v: int) ≡
  *(data + k) := v
```

private (i.e., module-local) data:
completely hidden from outside.

$$A_{Map}$$

```
private map := λ k. 0

def init(sz: int) ≡
  skip

def get(k: int) ≡
  return map[k]


def set(k: int, v: int) ≡
  map := map[k ↤ v]
```

Motivating Example

Refinement alone is not enough

Good: transitivity allows incremental verification:
1st: memory abstraction, 2nd: algorithm-specific reasoning

I_{Map}

```
private data := NULL

def init(sz: int) ≡
  data := calloc(sz)

def get(k: int) ≡
  return *(data + k)

def set(k: int, v: int) ≡
  *(data + k) := v
```

$\sqsubseteq$

M_{Map}

middle abstraction

$\sqsubseteq$

A_{Map}

```
private map := λ k. 0

def init(sz: int) ≡
  skip

def get(k: int) ≡
  return map[k]

def set(k: int, v: int) ≡
  map := map[k ↤ v]
```

Motivating Example

Refinement alone is not enough

I_{Map}

```
private data := NULL


def init(sz: int) ≡
  data := calloc(sz)

def get(k: int) ≡
  return *(data + k)


def set(k: int, v: int) ≡
  *(data + k) := v
```

$$\not\sqsubseteq$$

A_{Map}

```
private map := λ k. 0


def init(sz: int) ≡
  skip

def get(k: int) ≡
  return map[k]


def set(k: int, v: int) ≡
  map := map[k ↤ v]
```

Motivating Example

Refinement alone is not enough

Bad: Refinement does not hold in the first place!

It only holds **conditionally**: `init` should be called at most once and before any other operation.

I_{Map}

```
private data := NULL

def init(sz: int) ≡
  data := calloc(sz)

def get(k: int) ≡
  return *(data + k)

def set(k: int, v: int) ≡
  *(data + k) := v
```

$\not\sqsubseteq$

A_{Map}

```
private map := λ k. 0

def init(sz: int) ≡
  skip

def get(k: int) ≡
  return map[k]

def set(k: int, v: int) ≡
  map := map[k ↤ v]
```

Motivating Example

Benefits of separation logic

$$I_{Map}$$

```
private data := NULL


def init(sz: int) ≡
  data := calloc(sz)

def get(k: int) ≡
  return *(data + k)


def set(k: int, v: int) ≡
  *(data + k) := v
```

$\forall \mathsf{sz}.\ \{\boxed{\textit{pending}}\}\ \mathtt{init(sz)}$ $\{\mathop{\ast}_{k\in[0,\mathsf{sz})} \mathsf{k} \mapsto_{\mathsf{Map}} \mathtt{0}\}$

$\forall \mathsf{k}\ \mathsf{v}.\ \{\mathsf{k} \mapsto_{\mathsf{Map}} \mathsf{v}\}\ \mathtt{get(k)}$ $\{r.\ r = \mathsf{v} \land \mathsf{k} \mapsto_{\mathsf{Map}} \mathsf{v}\}$

$\forall \mathsf{k}\ \mathsf{w}\ \mathsf{v}.\ \{\mathsf{k} \mapsto_{\mathsf{Map}} \mathsf{w}\}\ \mathtt{set(k,v)}$ $\{\mathsf{k} \mapsto_{\mathsf{Map}} \mathsf{v}\}$

Motivating Example

Benefits of separation logic

I_{Map}

```
private data := NULL


def init(sz: int) ≡
  data := calloc(sz)

def get(k: int) ≡
  return *(data + k)


def set(k: int, v: int) ≡
  *(data + k) := v
```

$\boxed{pending}$ is an exclusive token, that gets consumed when calling `init`.

$\forall \mathsf{sz}.\ \{\boxed{pending}\}\ \mathtt{init(sz)}\ \{\mathop{\Large *}_{k \in [0,\mathsf{sz})} \mathsf{k} \mapsto_{\mathsf{Map}} \mathtt{0}\}$

$\forall \mathsf{k}\ \mathsf{v}.\ \{\mathsf{k} \mapsto_{\mathsf{Map}} \mathsf{v}\}\ \mathtt{get(k)}\ \{r.\ r = \mathsf{v} \wedge \mathsf{k} \mapsto_{\mathsf{Map}} \mathsf{v}\}$

$\forall \mathsf{k}\ \mathsf{w}\ \mathsf{v}.\ \{\mathsf{k} \mapsto_{\mathsf{Map}} \mathsf{w}\}\ \mathtt{set(k,v)}\ \{\mathsf{k} \mapsto_{\mathsf{Map}} \mathsf{v}\}$

Motivating Example

Benefits of separation logic

I_{Map}

```
private data := NULL


def init(sz: int) ≡
  data := calloc(sz)

def get(k: int) ≡
  return *(data + k)


def set(k: int, v: int) ≡
  *(data + k) := v
```

$\boxed{pending}$ is an exclusive token, that gets consumed when calling `init`.

"$\mathsf{k} \mapsto_{\mathsf{Map}} \mathsf{v}$" denotes that it is initialized, and a key k stores a value v.

$$\forall \mathsf{sz}.\ \{\boxed{pending}\}\ \texttt{init(sz)}\ \{\textstyle\mathop{\ast}_{k \in [0,\mathsf{sz})} \mathsf{k} \mapsto_{\mathsf{Map}} \texttt{0}\}$$

$$\forall \mathsf{k}\,\mathsf{v}.\ \{\mathsf{k} \mapsto_{\mathsf{Map}} \mathsf{v}\}\ \texttt{get(k)}\ \{r.\ r = \mathsf{v} \land \mathsf{k} \mapsto_{\mathsf{Map}} \mathsf{v}\}$$

$$\forall \mathsf{k}\,\mathsf{w}\,\mathsf{v}.\ \{\mathsf{k} \mapsto_{\mathsf{Map}} \mathsf{w}\}\ \texttt{set(k,v)}\ \{\mathsf{k} \mapsto_{\mathsf{Map}} \mathsf{v}\}$$

Our Contribution: CCR

$$S \models I \sqsubseteq A$$

Conditional Contextual Refinement

Our Contribution: CCR

$$S \models I \sqsubseteq A$$

$S : \mathit{String} \rightharpoonup \mathit{Cond}$

(Cond is pre/postcond in separation logic)

Conditional Contextual Refinement

Our Contribution: CCR

$$S \models I \sqsubseteq A$$

$S : \mathit{String} \rightharpoonup \mathit{Cond}$

(Cond is pre/postcond in separation logic)

Conditional Contextual Refinement

- Meaning that the refinement $I \sqsubseteq A$ holds under the separation logic conditions S

Our Contribution: CCR

$$S \models I \sqsubseteq A$$

$S : \mathit{String} \rightharpoonup \mathit{Cond}$

(Cond is pre/postcond in separation logic)

Conditional Contextual Refinement

Enjoys benefits of both sides

- Meaning that the refinement $I \sqsubseteq A$ holds under the separation logic conditions S

Our Contribution: CCR

$$S \models I \sqsubseteq A$$

$S : \mathit{String} \rightharpoonup \mathit{Cond}$
(Cond is pre/postcond in separation logic)

Conditional Contextual Refinement

Enjoys benefits of both sides

- Meaning that the refinement $I \sqsubseteq A$ holds under the separation logic conditions S

CCR is:

Our Contribution: CCR

$$S \models I \sqsubseteq A$$

$S : String \rightharpoonup Cond$

($Cond$ is pre/postcond in separation logic)

Conditional Contextual Refinement

Enjoys benefits of both sides

- Meaning that the refinement $I \sqsubseteq A$ holds under the separation logic conditions S

CCR is:

- Formalized in Coq

Our Contribution: CCR

$$S \models I \sqsubseteq A$$

$S : String \rightharpoonup Cond$

($Cond$ is pre/postcond in separation logic)

Conditional Contextual Refinement

Enjoys benefits of both sides

- Meaning that the refinement $I \sqsubseteq A$ holds under the separation logic conditions S

CCR is:

- Formalized in Coq

- Challenging case studies

shared memory, mutual recursion, function pointers, (non-)termination, system calls

Our Contribution: CCR

$$S \models I \sqsubseteq A$$

$S : \mathit{String} \rightharpoonup \mathit{Cond}$
(Cond is pre/postcond in separation logic)

Conditional Contextual Refinement

Enjoys benefits of both sides

- Meaning that the refinement $I \sqsubseteq A$ holds under the separation logic conditions S

CCR is:

- Formalized in Coq

- Challenging case studies

shared memory, mutual recursion, function pointers, (non-)termination, system calls

- Ready to use: end-to-end verification (with CompCert) and executable (with Interaction Trees)

Motivating Example

With Conditional Contextual Refinement

I_{Map}

```
private data := NULL


def init(sz: int) ≡
  data := calloc(sz)

def get(k: int) ≡
  return *(data + k)


def set(k: int, v: int) ≡
  *(data + k) := v
```

$\not\sqsubseteq$

A_{Map}

```
private map := λ k. 0


def init(sz: int) ≡
  skip

def get(k: int) ≡

  return map[k]

def set(k: int, v: int) ≡

  map := map[k ↤ v]
```

Motivating Example

With Conditional Contextual Refinement

I_{Map}

```
private data := NULL


def init(sz: int) ≡
  data := calloc(sz)

def get(k: int) ≡
  return *(data + k)


def set(k: int, v: int) ≡
  *(data + k) := v
```

$\not\sqsubseteq$

A_{Map}

```
private map := λ k. 0


def init(sz: int) ≡
  skip

def get(k: int) ≡

  return map[k]

def set(k: int, v: int) ≡

  map := map[k ↤ v]
```

Motivating Example

With Conditional Contextual Refinement

I_{Map}

```
private data := NULL


def init(sz: int) ≡
  data := calloc(sz)

def get(k: int) ≡
  return *(data + k)


def set(k: int, v: int) ≡
  *(data + k) := v
```

$\sqsubseteq$

A_{Map}

```
private map := λ k. 0


def init(sz: int) ≡
  skip

def get(k: int) ≡

  return map[k]

def set(k: int, v: int) ≡

  map := map[k ↤ v]
```

$\forall sz.\ \{\boxed{\mathit{pending}}\}\quad \mathtt{init(sz)}\ \{\ \ast_{k\in[0,sz)}\ k \mapsto_{Map} 0\ \}$

$\forall k\ v.\ \{\ k \mapsto_{Map} v\ \}\ \ \mathtt{get(k)}\quad \{\ r.\ r = v * k \mapsto_{Map} v\ \}$

$\forall k\ v.\ \{\ \exists w.\ k \mapsto_{Map} w\ \}\ \ \mathtt{set(k, v)}\ \{\ k \mapsto_{Map} v\ \}$

Motivating Example

With Conditional Contextual Refinement

I_{Map}

```
private data := NULL


def init(sz: int) ≡
  data := calloc(sz)

def get(k: int) ≡
  return *(data + k)


def set(k: int, v: int) ≡
  *(data + k) := v
```

$$\sqsubseteq$$

A_{Map}

```
private map := λ k. 0


def init(sz: int) ≡
  skip

def get(k: int) ≡

  return map[k]

def set(k: int, v: int) ≡

  map := map[k ↤ v]
```

$\forall \mathtt{sz}.\ \{\boxed{\textit{pending}}\}\ \ \mathtt{init(sz)}\ \ \{ \circledast_{\mathtt{k} \in [0,\mathtt{sz})}\ \mathtt{k} \mapsto_{\mathtt{Map}} \mathtt{0} \}$

$\forall \mathtt{k}\ \mathtt{v}.\ \{ \mathtt{k} \mapsto_{\mathtt{Map}} \mathtt{v} \}\ \ \mathtt{get(k)}\ \ \{ \mathtt{r}.\ \mathtt{r} = \mathtt{v} * \mathtt{k} \mapsto_{\mathtt{Map}} \mathtt{v} \}$

$\forall \mathtt{k}\ \mathtt{v}.\ \{ \exists w.\ \mathtt{k} \mapsto_{\mathtt{Map}} \mathtt{w} \}\ \ \mathtt{set(k,\ v)}\ \{ \mathtt{k} \mapsto_{\mathtt{Map}} \mathtt{v} \}$

Motivating Example

With Conditional Contextual Refinement

I_{Map}

```
private data := NULL

def init(sz: int) ≡
  data := calloc(sz)

def get(k: int) ≡
  return *(data + k)

def set(k: int, v: int) ≡
  *(data + k) := v
```

$\sqsubseteq$

M_{Map}

```
private map := λ k. 0
private size := 0

def init(sz: int) ≡
  size := sz

def get(k: int) ≡
  assume(0 ≤ k < size)
  return map[k]

def set(k: int, v: int) ≡
  assume(0 ≤ k < size)
  map := map[k ↤ v]
```

$\sqsubseteq$

A_{Map}

```
private map := λ k. 0

def init(sz: int) ≡
  skip

def get(k: int) ≡

  return map[k]

def set(k: int, v: int) ≡

  map := map[k ↤ v]
```

$\forall sz.\ \{\boxed{pending}\}\ \mathtt{init(sz)}\ \{\top\}$

$\forall k.\ \{\top\}\ \mathtt{get(k)}\ \{\top\}$

$\forall k\ v.\ \{\top\}\ \mathtt{set(k,\ v)}\ \{\top\}$

$\forall sz.\ \{\boxed{pending}\}\ \mathtt{init(sz)}\ \{\circledast_{k\in[0,sz)}\ k \mapsto_{Map} 0\}$

$\forall k\ v.\ \{k \mapsto_{Map} v\}\ \mathtt{get(k)}\ \{r.\ r = v * k \mapsto_{Map} v\}$

$\forall k\ v.\ \{\exists w.\ k \mapsto_{Map} w\}\ \mathtt{set(k,\ v)}\ \{k \mapsto_{Map} v\}$

Both benefits **at the same time**!

```
def init(sz: int) ≡
  data := calloc(sz)

def get(k: int) ≡
  return *(data + k)


def set(k: int, v: int) ≡
  *(data + k) := v
```

⊑

```
private size := 0

def init(sz: int) ≡
  size := sz

def get(k: int) ≡
  assume(0 ≤ k < size)
  return map[k]

def set(k: int, v: int) ≡
  assume(0 ≤ k < size)
  map := map[k ↤ v]
```

⊑

```
def init(sz: int) ≡
  skip

def get(k: int) ≡

  return map[k]

def set(k: int, v: int) ≡

  map := map[k ↤ v]
```

$\forall \mathtt{sz}.\ \{\boxed{\mathit{pending}}\}\ \mathtt{init(sz)}\ \{\top\}$
$\forall \mathtt{k}.\ \{\top\}\ \mathtt{get(k)}\ \{\top\}$
$\forall \mathtt{k}\ \mathtt{v}.\ \{\top\}\ \mathtt{set(k,\ v)}\ \{\top\}$

$\forall \mathtt{sz}.\ \{\boxed{\mathit{pending}}\}\ \mathtt{init(sz)}\ \{\mathop{\ast}_{\mathsf{k}\in[0,\mathsf{sz})} \mathsf{k} \mapsto_{\mathsf{Map}} \mathtt{0}\}$
$\forall \mathsf{k}\ \mathsf{v}.\ \{\mathsf{k} \mapsto_{\mathsf{Map}} \mathsf{v}\}\ \mathtt{get(k)}\ \{\mathsf{r}.\ \mathsf{r} = \mathsf{v} \ast \mathsf{k} \mapsto_{\mathsf{Map}} \mathsf{v}\}$
$\forall \mathsf{k}\ \mathsf{v}.\ \{\exists w.\ \mathsf{k} \mapsto_{\mathsf{Map}} w\}\ \mathtt{set(k,\ v)}\ \{\mathsf{k} \mapsto_{\mathsf{Map}} \mathsf{v}\}$

Motivating Example

With Conditional Contextual Refinement

I_{Map}

```
private data := NULL


def init(sz: int) ≡
  data := calloc(sz)

def get(k: int) ≡
  return *(data + k)


def set(k: int, v: int) ≡
  *(data + k) := v
```

$\sqsubseteq$

M_{Map}

```
private map := λ k. 0
private size := 0

def init(sz: int) ≡
  size := sz

def get(k: int) ≡
  assume(0 ≤ k < size)
  return map[k]

def set(k: int, v: int) ≡
  assume(0 ≤ k < size)
  map := map[k ↤ v]
```

$\forall sz.\ \{\boxed{\mathit{pending}}\}$ init(sz) $\{\top\}$

$\forall k.\ \{\top\}$ get(k) $\{\top\}$

$\forall k\ v.\ \{\top\}$ set(k, v) $\{\top\}$

Motivating Example

With Conditional Contextual Refinement

M_{Map}

```
private map := λ k. 0
private size := 0

def init(sz: int) ≡
  size := sz

def get(k: int) ≡
  assume(0 ≤ k < size)
  return map[k]

def set(k: int, v: int) ≡
  assume(0 ≤ k < size)
  map := map[k ↤ v]
```

$\sqsubseteq$

A_{Map}

```
private map := λ k. 0


def init(sz: int) ≡
  skip

def get(k: int) ≡

  return map[k]

def set(k: int, v: int) ≡

  map := map[k ↤ v]
```

$\forall \mathtt{sz}.\ \{\boxed{\mathit{pending}}\}\ \mathtt{init(sz)}\ \{ \ast_{\mathtt{k} \in [0,\mathtt{sz})}\ \mathtt{k} \mapsto_{\mathtt{Map}} 0 \}$

$\forall \mathtt{k}\ \mathtt{v}.\ \{ \mathtt{k} \mapsto_{\mathtt{Map}} \mathtt{v} \}\ \mathtt{get(k)}\ \{ \mathtt{r}.\ \mathtt{r} = \mathtt{v} \ast \mathtt{k} \mapsto_{\mathtt{Map}} \mathtt{v} \}$

$\forall \mathtt{k}\ \mathtt{v}.\ \{ \exists w.\ \mathtt{k} \mapsto_{\mathtt{Map}} \mathtt{w} \}\ \mathtt{set(k, v)}\ \{ \mathtt{k} \mapsto_{\mathtt{Map}} \mathtt{v} \}$

Motivating Example

With Conditional Contextual Refinement

I_{Map}

```
private data := NULL


def init(sz: int) ≡
  data := calloc(sz)

def get(k: int) ≡
  return *(data + k)


def set(k: int, v: int) ≡
  *(data + k) := v
```

$\sqsubseteq$

M_{Map}

```
private map := λ k. 0
private size := 0

def init(sz: int) ≡
  size := sz

def get(k: int) ≡
  assume(0 ≤ k < size)
  return map[k]

def set(k: int, v: int) ≡
  assume(0 ≤ k < size)
  map := map[k ↤ v]
```

$\sqsubseteq$

A_{Map}

```
private map := λ k. 0


def init(sz: int) ≡
  skip

def get(k: int) ≡

  return map[k]

def set(k: int, v: int) ≡

  map := map[k ↤ v]
```

$\forall sz.\ \{\boxed{\mathit{pending}}\}\ \texttt{init(sz)}\ \{\top\}$
$\forall k.\ \{\top\}\ \texttt{get(k)}\ \{\top\}$
$\forall k\ v.\ \{\top\}\ \texttt{set(k, v)}\ \{\top\}$

$\forall sz.\ \{\boxed{\mathit{pending}}\}\ \texttt{init(sz)}\ \{\ast_{k \in [0,sz)}\ k \mapsto_{Map} 0\}$
$\forall k\ v.\ \{k \mapsto_{Map} v\}\ \texttt{get(k)}\ \{r.\ r = v \ast k \mapsto_{Map} v\}$
$\forall k\ v.\ \{\exists w.\ k \mapsto_{Map} w\}\ \texttt{set(k, v)}\ \{k \mapsto_{Map} v\}$

Key Idea I: Wrapper

Wrapper

$$S \models I \sqsubseteq A \qquad \triangleq$$

Wrapper

$$S \models I \sqsubseteq A \quad \triangleq \quad I \sqsubseteq_{ctx} A$$

- We use unconditional refinement as an underlying notion, but

Wrapper

$$S \models I \sqsubseteq A \quad \triangleq \quad I \sqsubseteq_{ctx} \underbrace{\langle\, S \vdash A \,\rangle}_{\text{Wrapper}}$$

- We use unconditional refinement as an underlying notion, but

Wrapper

$$S \models I \sqsubseteq A \quad \triangleq \quad I \sqsubseteq_{ctx} \underbrace{\langle S \vdash A \rangle}_{\text{Wrapper}}$$

- We use unconditional refinement as an underlying notion, but
  - “**Operationalize**” the **conditions**, enforcing dynamically that they hold

Wrapper

$$S \models I \sqsubseteq A \quad \triangleq \quad I \sqsubseteq_{ctx} \underbrace{\langle S \vdash A \rangle}_{\text{Wrapper}}$$

- We use unconditional refinement as an underlying notion, but
  - “**Operationalize**” the **conditions**, enforcing dynamically that they hold
- **Good:** we can piggyback on all the existing benefits of unconditional refinement

Wrapper

$$S \vDash I \sqsubseteq A \quad \triangleq \quad I \sqsubseteq_{ctx} \underbrace{\langle S \vdash A \rangle}_{\text{Wrapper}}$$

- We use unconditional refinement as an underlying notion, but
  - "**Operationalize**" the **conditions**, enforcing dynamically that they hold
- **Good:** we can piggyback on all the existing benefits of unconditional refinement
  - Simple, universal definition

Wrapper

$$S \vDash I \sqsubseteq A \quad \triangleq \quad I \sqsubseteq_{ctx} \underbrace{\langle S \vdash A \rangle}_{\text{Wrapper}}$$

- We use unconditional refinement as an underlying notion, but
  - “**Operationalize**” the **conditions**, enforcing dynamically that they hold
- **Good:** we can piggyback on all the existing benefits of unconditional refinement
  - Simple, universal definition
  - Vertical compositionality (i.e., transitivity), Horizontal compositionality

Motivating Example

With Conditional Contextual Refinement

I_{Map}

```
private data := NULL


def init(sz: int) ≡
  data := calloc(sz)

def get(k: int) ≡
  return *(data + k)


def set(k: int, v: int) ≡
  *(data + k) := v
```

$\sqsubseteq$

$\forall sz.\ \{\boxed{pending}\}\ \texttt{init(sz)}\ \{\top\}$

$\forall k\, v.\ \{\top\}\ \texttt{get(k), set(k, v)}\ \{\top\}$

M_{Map}

```
private map := λ k. 0
private size := 0

def init(sz: int) ≡
  size := sz

def get(k: int) ≡
  assume(0 ≤ k < size)
  return map[k]

def set(k: int, v: int) ≡
  assume(0 ≤ k < size)
  map := map[k ↤ v]
```

$\sqsubseteq$

$\forall sz.\ \{\boxed{pending}\}\ \texttt{init(sz)}\ \{\ast_{k\in[0,sz)}\ k \mapsto_{Map} 0\}$

$\forall k\, v.\ \{k \mapsto_{Map} v\}\ \texttt{get(k)}\ \{r.\, r = v \ast k \mapsto_{Map} v\}$

$\forall k\, v.\ \{\exists w.\, k \mapsto_{Map} w\}\ \texttt{set(k, v)}\ \{k \mapsto_{Map} v\}$

A_{Map}

```
private map := λ k. 0


def init(sz: int) ≡
  skip

def get(k: int) ≡

  return map[k]

def set(k: int, v: int) ≡

  map := map[k ↤ v]
```

Motivating Example

With Conditional Contextual Refinement

I_{Map}

```
private data := NULL


def init(sz: int) ≡
  data := calloc(sz)

def get(k: int) ≡
  return *(data + k)


def set(k: int, v: int) ≡
  *(data + k) := v
```

$\sqsubseteq$

M_{Map}

```
private map := λ k. 0
private size := 0

def init(sz: int) ≡
  size := sz

def get(k: int) ≡
  assume(0 ≤ k < size)
  return map[k]

def set(k: int, v: int) ≡
  assume(0 ≤ k < size)
  map := map[k ↤ v]
```

$\sqsubseteq$

A_{Map}

```
private map := λ k. 0


def init(sz: int) ≡
  skip

def get(k: int) ≡

  return map[k]

def set(k: int, v: int) ≡

  map := map[k ↤ v]
```

Motivating Example

With Conditional Contextual Refinement

I_{Map}

```
private data := NULL


def init(sz: int) ≡
  data := calloc(sz)

def get(k: int) ≡
  return *(data + k)


def set(k: int, v: int) ≡
  *(data + k) := v
```

$\sqsubseteq$

$\langle S'_{Map} \vdash M_{Map} \rangle$

```
private map := λ k. 0
private size := 0

def init(sz: int) ≡
  OPERATIONALIZED_CONDS(…)
  size := sz
  OPERATIONALIZED_CONDS(…)

def get(k: int) ≡
  OPERATIONALIZED_CONDS(…)
  assume(0 ≤ k < size)
  return map[k]
  OPERATIONALIZED_CONDS(…)

def set(k: int, v: int) ≡
  OPERATIONALIZED_CONDS(…)
  assume(0 ≤ k < size)
  map := map[k ↤ v]
  OPERATIONALIZED_CONDS(…)
```

$\sqsubseteq$

$\langle S_{Map} \vdash A_{Map} \rangle$

```
private map := λ k. 0


def init(sz: int) ≡
  OPERATIONALIZED_CONDS(…)
  skip
  OPERATIONALIZED_CONDS(…)

def get(k: int) ≡
  OPERATIONALIZED_CONDS(…)

  return map[k]
  OPERATIONALIZED_CONDS(…)

def set(k: int, v: int) ≡
  OPERATIONALIZED_CONDS(…)

  map := map[k ↤ v]
  OPERATIONALIZED_CONDS(…)
```

Motivating Example

With Conditional Contextual Refinement

I_{Map}

```
private data := NULL
```

```
def set(k: int, v: int)
  *(data + k) := v
```

$\langle\, S'_{Map} \vdash M_{Map} \,\rangle$

```
private map := λ k. 0
private size := 0
```

```
  return map[k]
  OPERATIONALIZED_CONDS(…)

def set(k: int, v: int) ≡
  OPERATIONALIZED_CONDS(…)
  assume(0 ≤ k < size)
  map := map[k ↤ v]
  OPERATIONALIZED_CONDS(…)
```

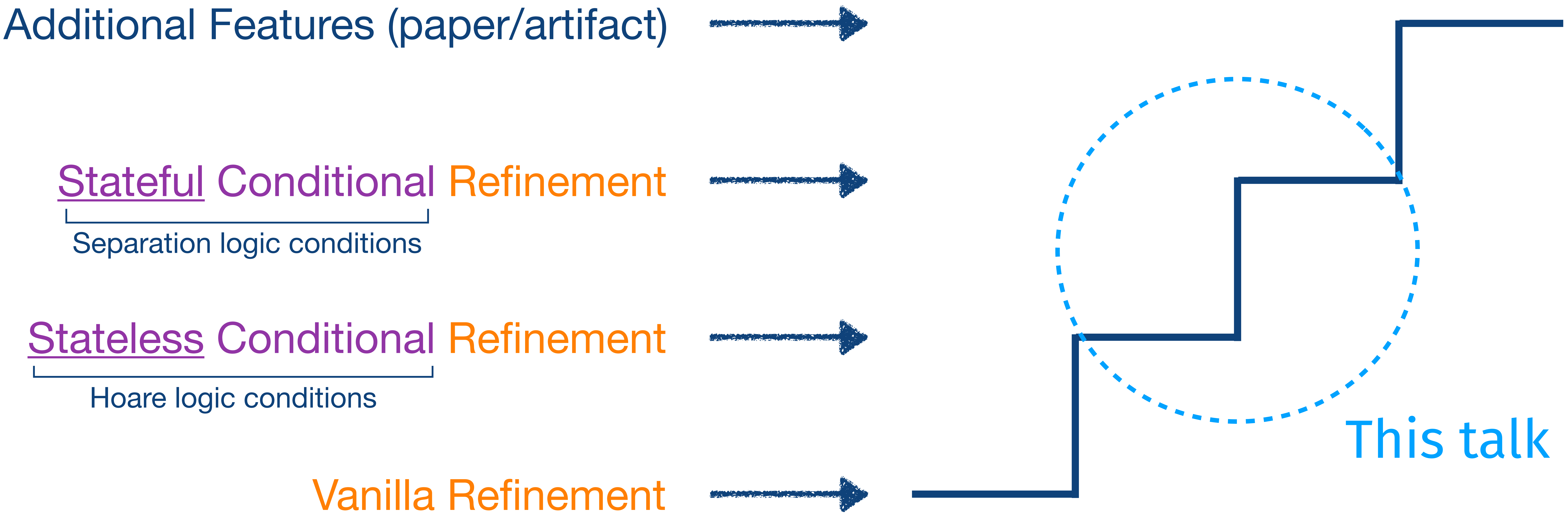
$\langle\, S_{Map} \vdash A_{Map} \,\rangle$

```
private map := λ k. 0
```

```
  return map[k]
  OPERATIONALIZED_CONDS(…)

def set(k: int, v: int) ≡
  OPERATIONALIZED_CONDS(…)

  map := map[k ↤ v]
  OPERATIONALIZED_CONDS(…)
```

What should be the definition of OPERATIONALIZED_CONDS?

Towards the Wrapper



Additional Features (paper/artifact)

Stateful Conditional Refinement

Separation logic conditions

Stateless Conditional Refinement

Hoare logic conditions

Vanilla Refinement

This talk

Towards the Wrapper

Additional Features (paper/artifact) →

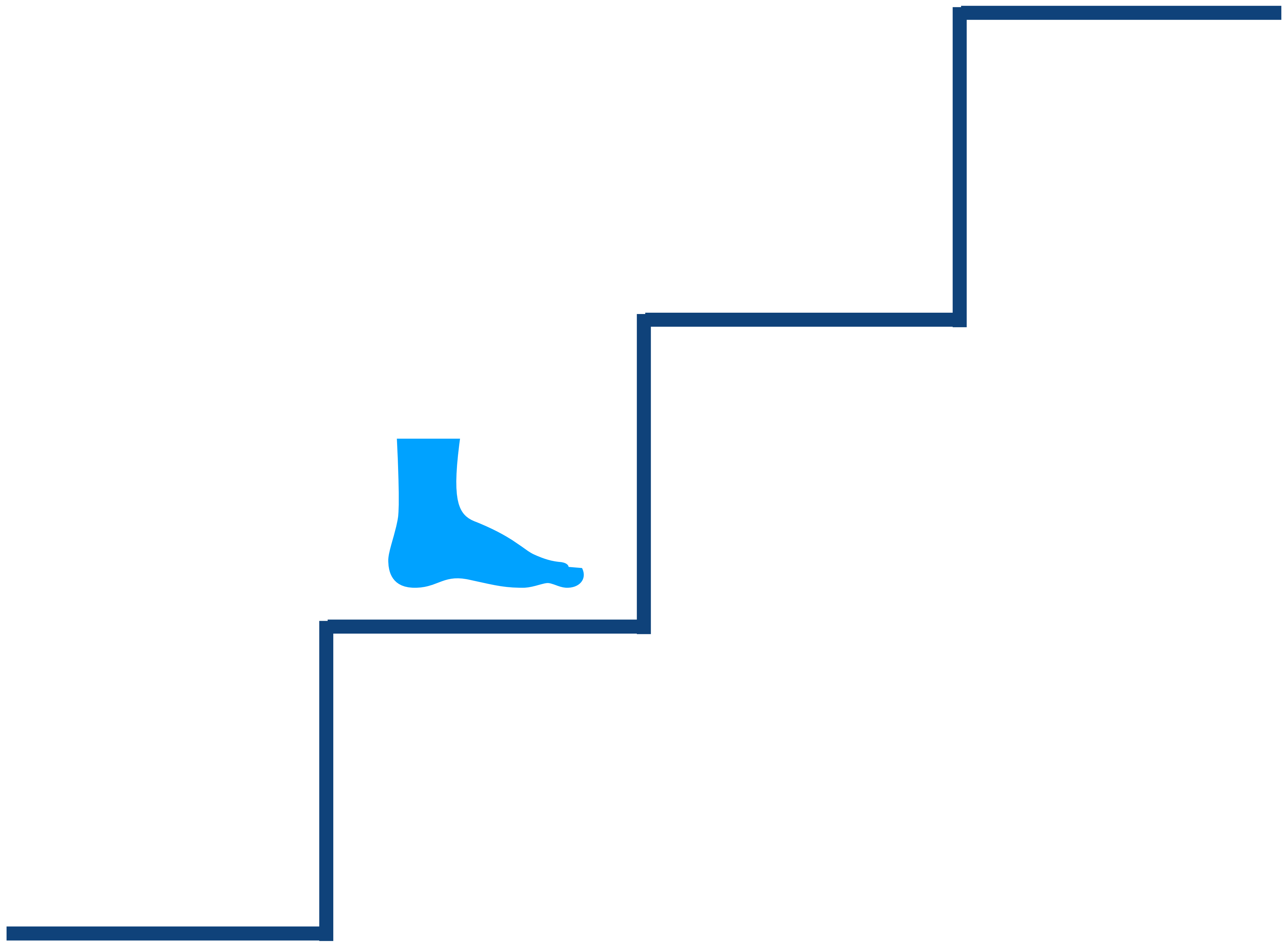
Stateful Conditional Refinement →

Separation logic conditions

Stateless Conditional Refinement →

Hoare logic conditions

Vanilla Refinement →



Stateless Conditional Refinement

I_{Expn}

```
def exp(x: int, n: int) ≡
  if n == 0
  then return 1
  else x * exp(x, n-1)
```

$\sqsubseteq$

A_{Expn}

```
def exp(x: int, n: int) ≡

  var r := x^n

  return r
```

$\{ n \geq 0 \}$ exp(x,n) $\{ r.r = x^n \}$

Stateless Conditional Refinement

I_{Expn}

```
def exp(x: int, n: int) ≡
  if n == 0
  then return 1
  else x * exp(x, n-1)
```

$\sqsubseteq$

A_{Expn}

```
def exp(x: int, n: int) ≡

  var r := $x^n$

  return r
```

Stateless Conditional Refinement

I_{Expn}

```
def exp(x: int, n: int) ≡
  if n == 0
  then return 1
  else x * exp(x, n-1)
```

$$\sqsubseteq$$

$\langle S \vdash A_{Expn} \rangle$

```
def exp(x: int, n: int) ≡
  OPERATIONALIZED_COND(…)
  var r := x^n
  OPERATIONALIZED_COND(…)
  return r
```

Stateless Conditional Refinement

I_{Expn}

```
def exp(x: int, n: int) ≡
  if n == 0
  then return 1
  else x * exp(x, n-1)
```

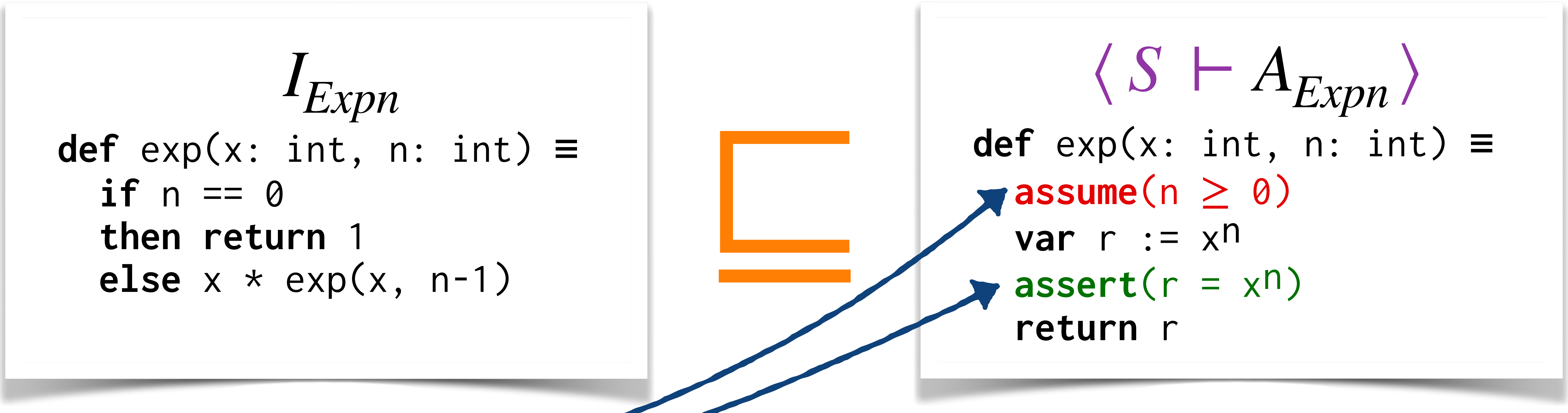
$$\sqsubseteq$$

$\langle S \vdash A_{Expn} \rangle$

```
def exp(x: int, n: int) ≡
  assume(n ≥ 0)
  var r := x^n
  assert(r = x^n)
  return r
```

Inspired by **Refinement Calculus**
[Ralph-Johan Back 1978]

Stateless Conditional Refinement



I_{Expn}

```
def exp(x: int, n: int) ≡
  if n == 0
  then return 1
  else x * exp(x, n-1)
```

$\sqsubseteq$

$\langle S \vdash A_{Expn} \rangle$

```
def exp(x: int, n: int) ≡
  assume(n ≥ 0)
  var r := xⁿ
  assert(r = xⁿ)
  return r
```

Inspired by **Refinement Calculus**
[Ralph-Johan Back 1978]

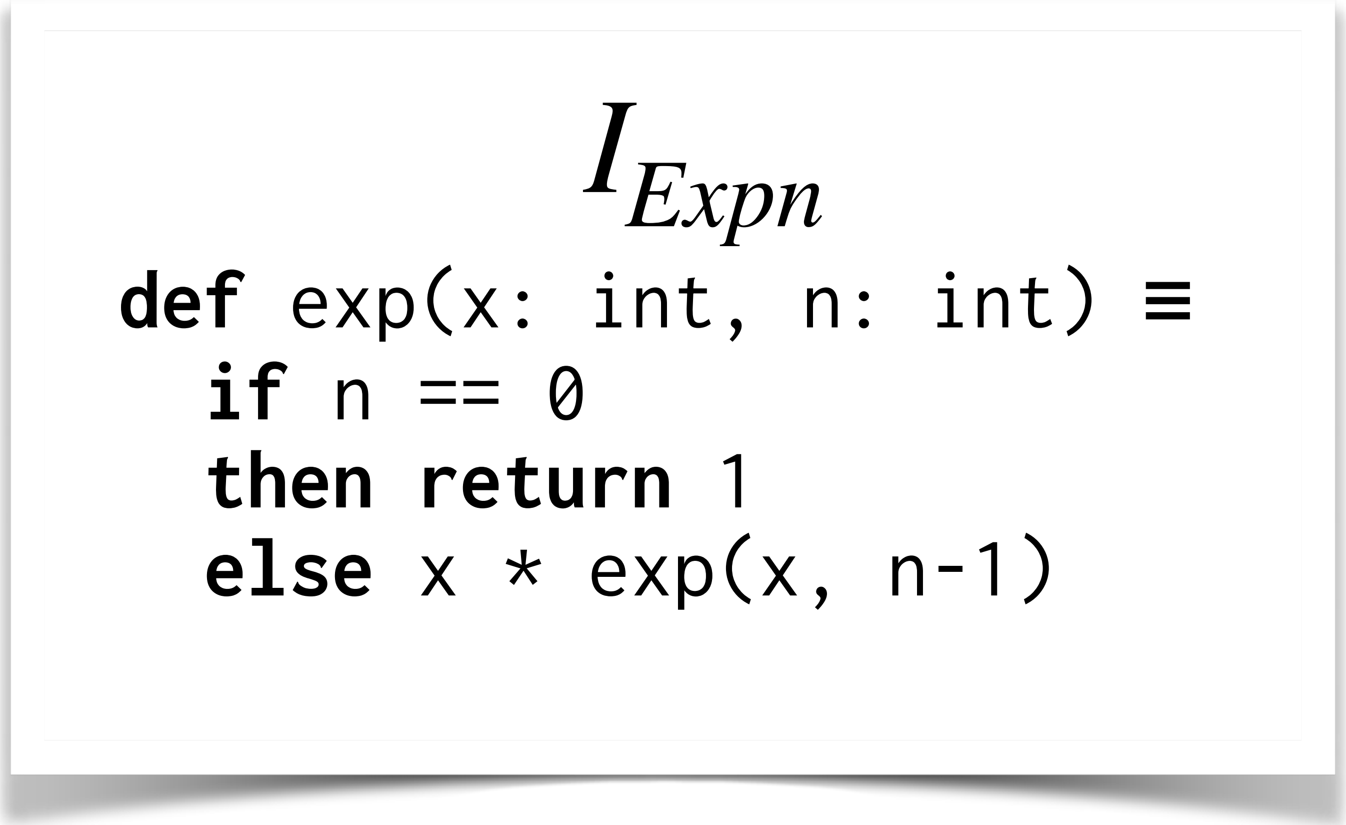
(ASMR)

$$\frac{P \implies \mathbb{T} \sqsubseteq \mathbb{S}}{T \sqsubseteq \textbf{assume}(P); \mathbb{S}}$$

(ASTR)

$$\frac{P \qquad \mathbb{T} \sqsubseteq \mathbb{S}}{\mathbb{T} \sqsubseteq \texttt{assert}(P); \mathbb{S}}$$

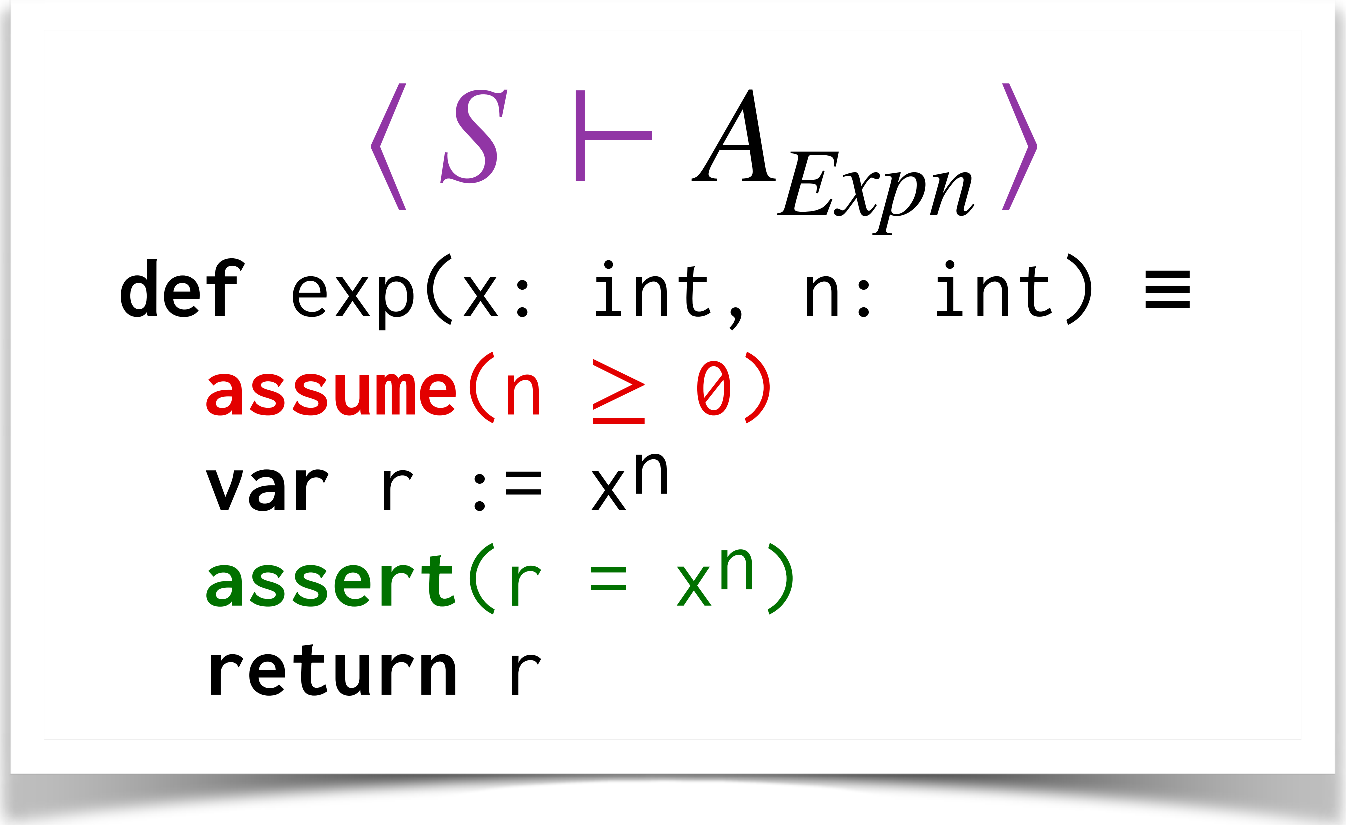
Stateless Conditional Refinement



I_{Expn}

```
def exp(x: int, n: int) ≡
  if n == 0
  then return 1
  else x * exp(x, n-1)
```

$\sqsubseteq$



$\langle S \vdash A_{Expn} \rangle$

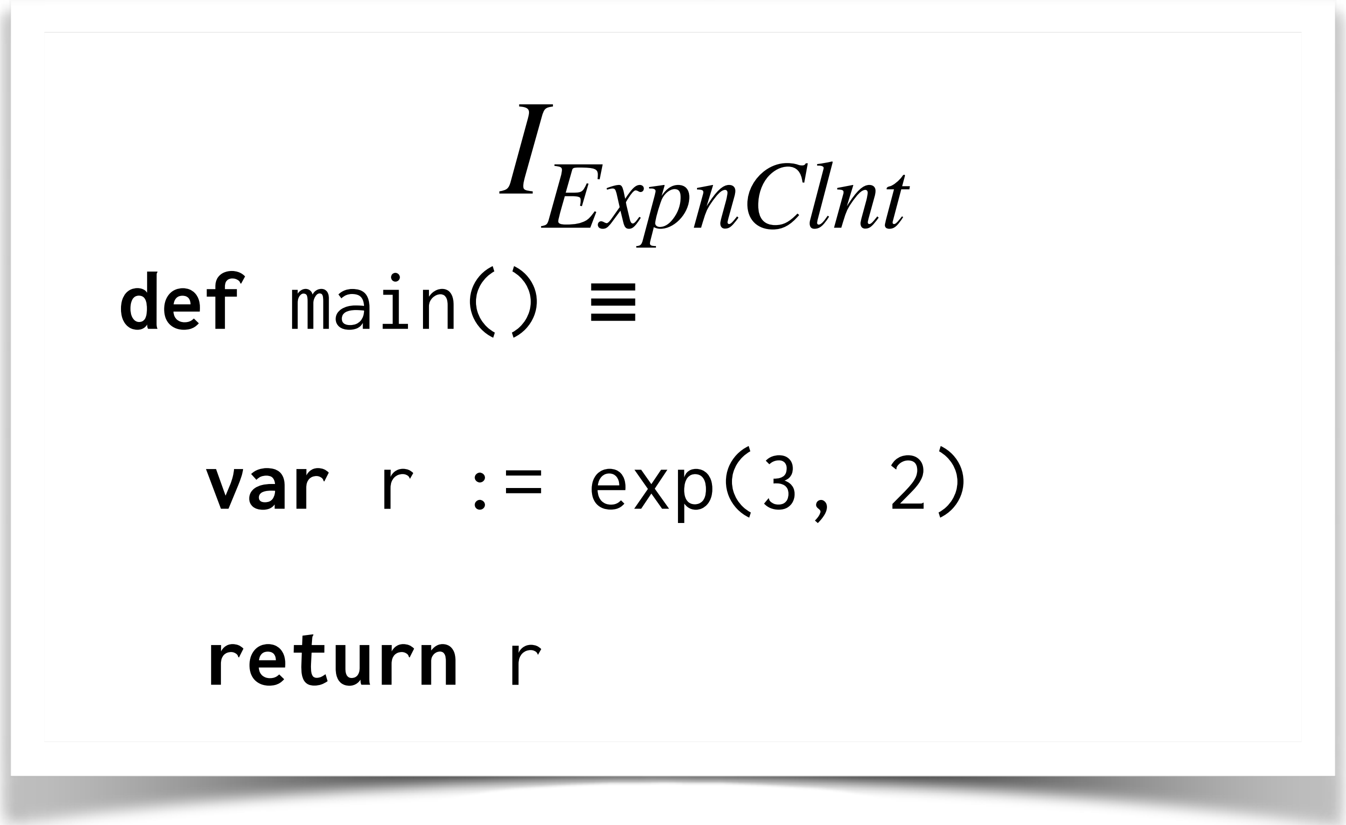
```
def exp(x: int, n: int) ≡
  assume(n ≥ 0)
  var r := x^n
  assert(r = x^n)
  return r
```

(ASMR)

$$\frac{P \implies \mathbb{T} \sqsubseteq \mathbb{S}}{T \sqsubseteq \textbf{assume}(P); \mathbb{S}}$$

(ASTR)

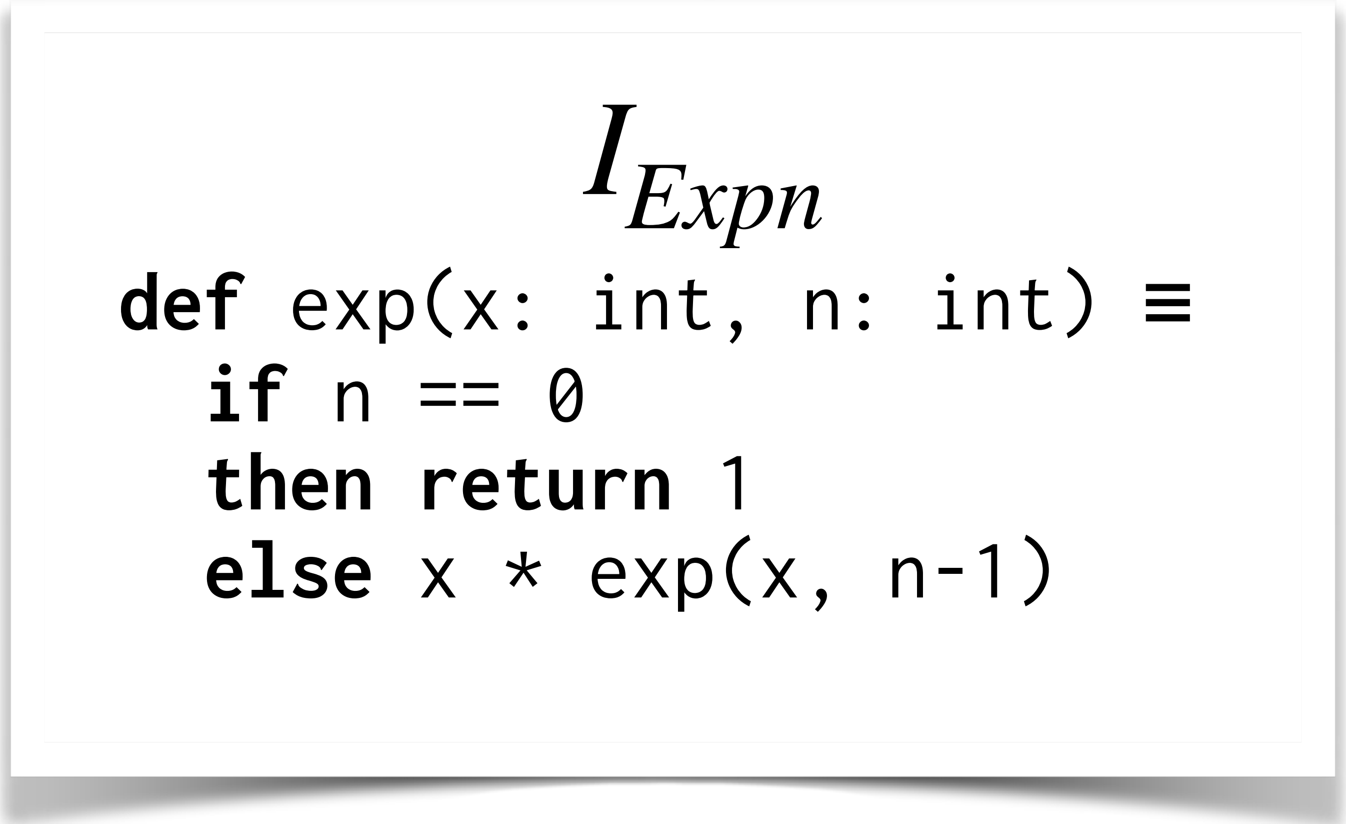
$$\frac{P \qquad \mathbb{T} \sqsubseteq \mathbb{S}}{\mathbb{T} \sqsubseteq \textbf{assert}(P); \mathbb{S}}$$



$I_{ExpnClnt}$

```
def main() ≡

  var r := exp(3, 2)

  return r
```

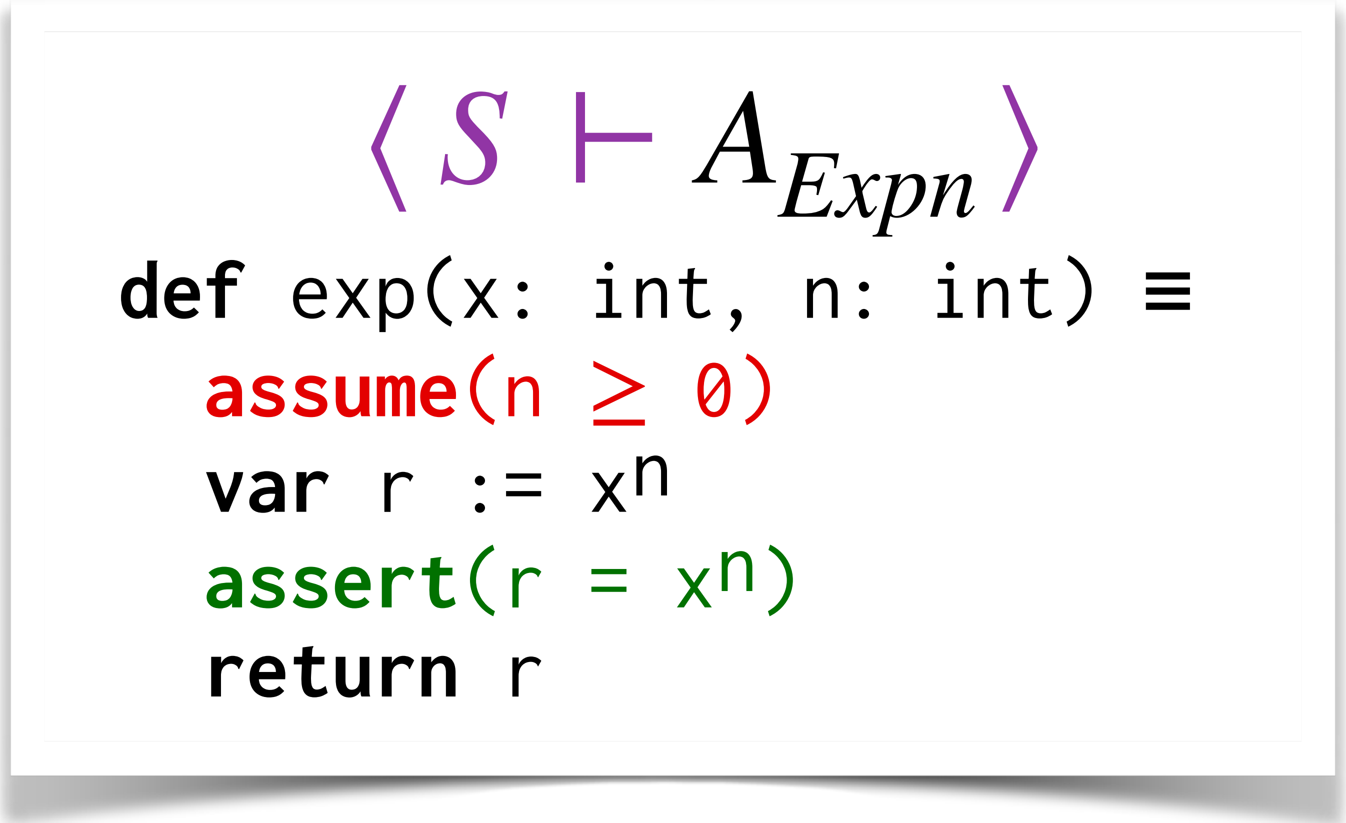
Stateless Conditional Refinement



I_{Expn}

```
def exp(x: int, n: int) ≡
  if n == 0
  then return 1
  else x * exp(x, n-1)
```

$\sqsubseteq$



$\langle S \vdash A_{Expn} \rangle$

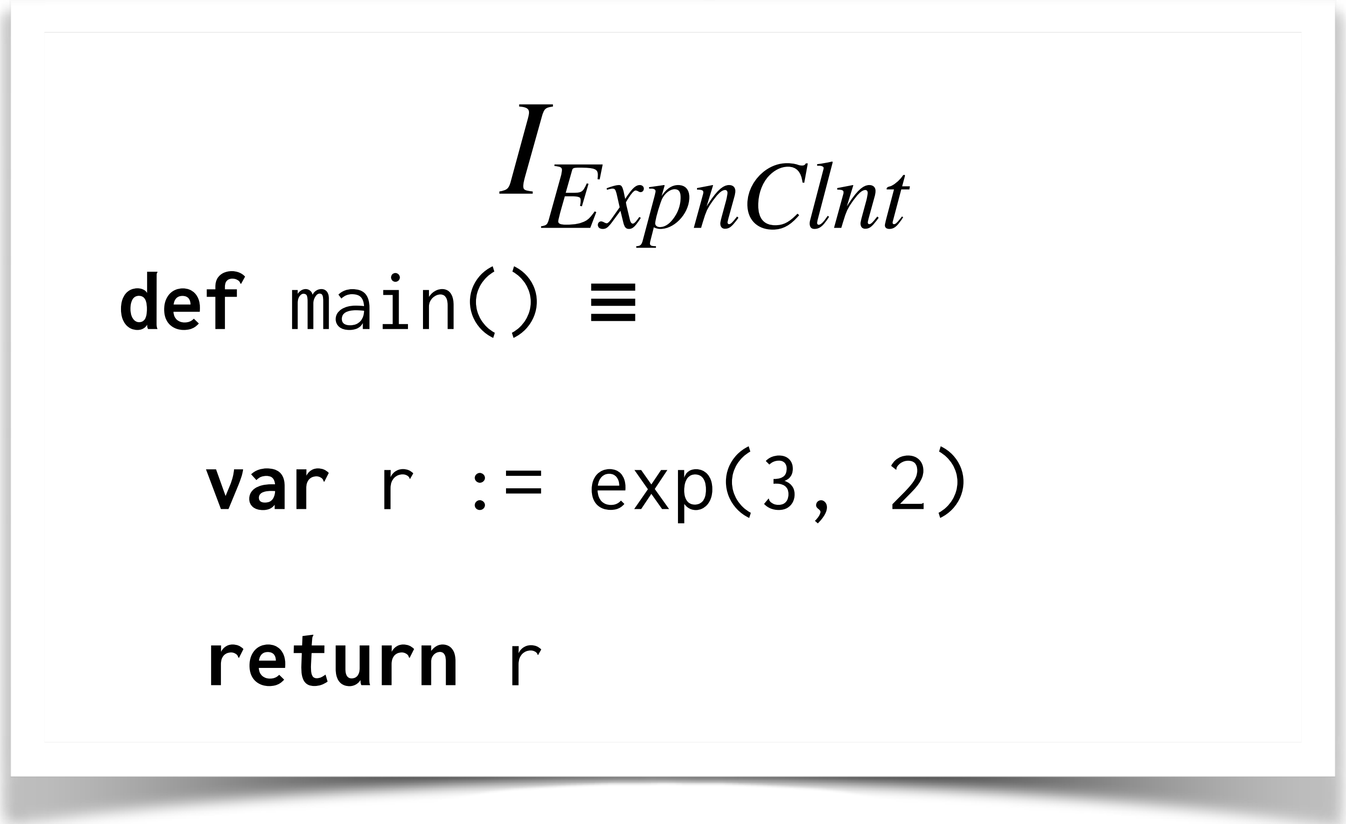
```
def exp(x: int, n: int) ≡
  assume(n ≥ 0)
  var r := x^n
  assert(r = x^n)
  return r
```

(ASMR)

$$\frac{P \Longrightarrow \mathbb{T} \sqsubseteq \mathbb{S}}{T \sqsubseteq \mathbf{assume}(P); \mathbb{S}}$$

(ASTR)

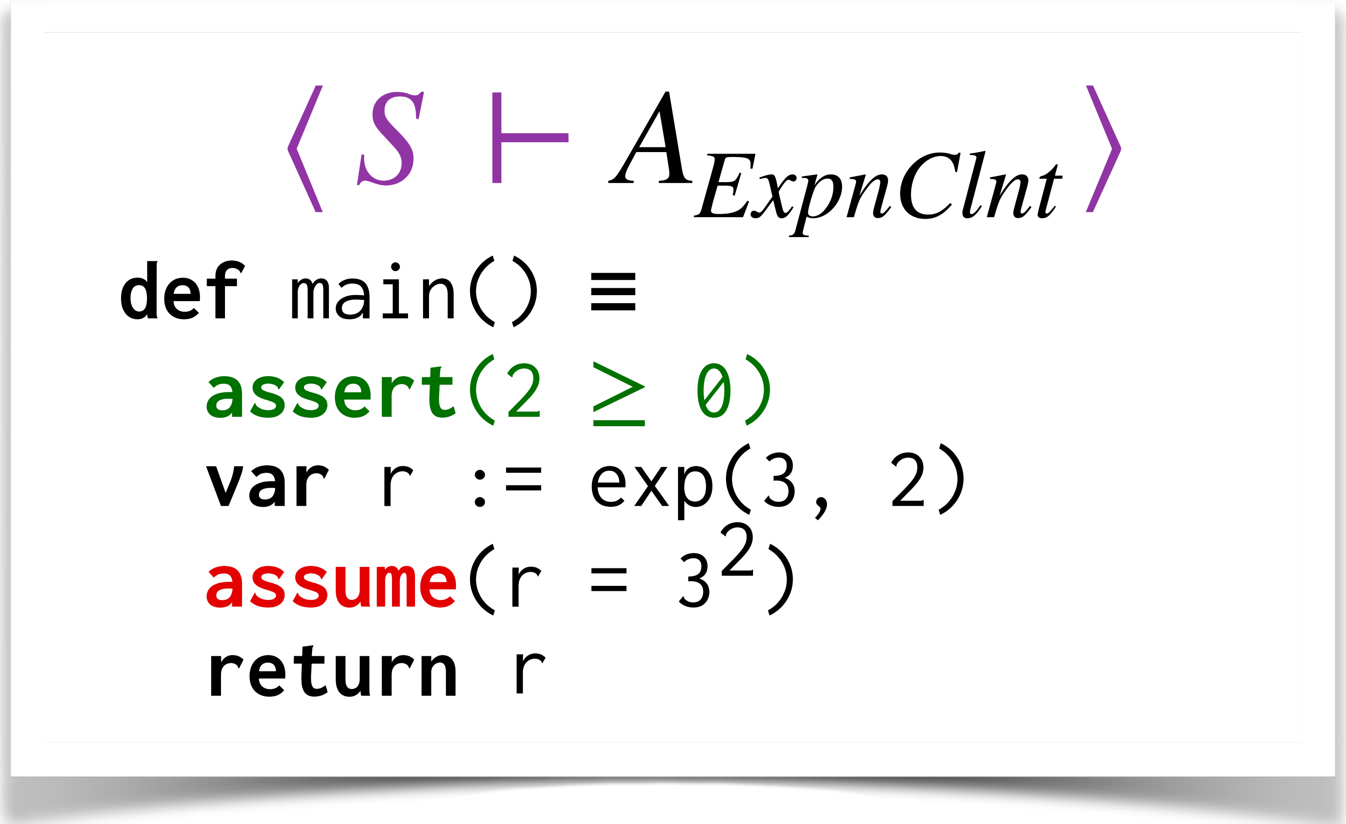
$$\frac{P \qquad \mathbb{T} \sqsubseteq \mathbb{S}}{\mathbb{T} \sqsubseteq \mathbf{assert}(P); \mathbb{S}}$$



$I_{ExpnClnt}$

```
def main() ≡

  var r := exp(3, 2)

  return r
```

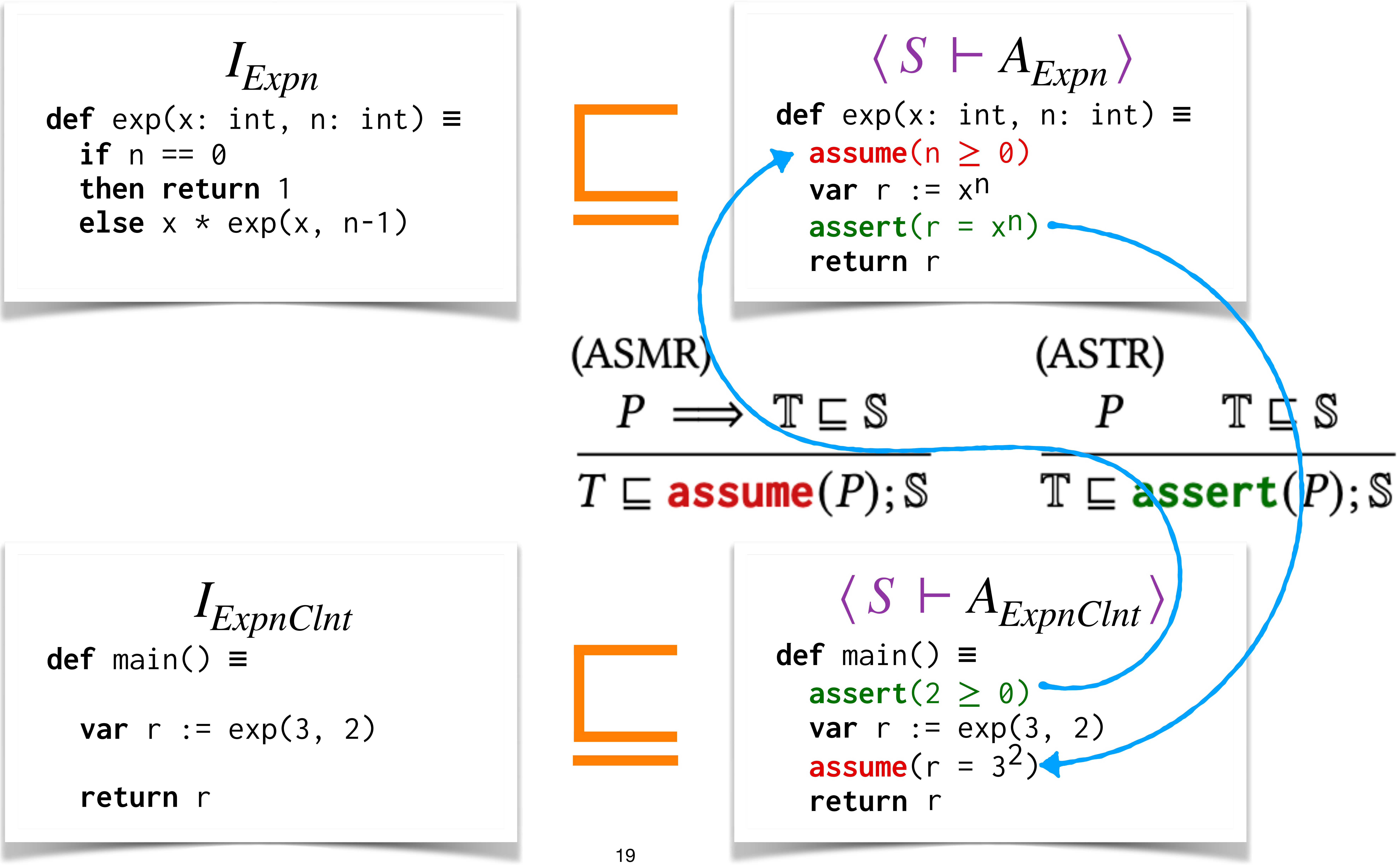
$\sqsubseteq$



$\langle S \vdash A_{ExpnClnt} \rangle$

```
def main() ≡
  assert(2 ≥ 0)
  var r := exp(3, 2)
  assume(r = 3^2)
  return r
```

Stateless Conditional Refinement



I_{Expn}

```
def exp(x: int, n: int) ≡
  if n == 0
  then return 1
  else x * exp(x, n-1)
```

$\sqsubseteq$

$\langle S \vdash A_{Expn} \rangle$

```
def exp(x: int, n: int) ≡
  assume(n ≥ 0)
  var r := xⁿ
  assert(r = xⁿ)
  return r
```

(ASMR)

$$\frac{P \implies \mathbb{T} \sqsubseteq \mathbb{S}}{T \sqsubseteq \mathbf{assume}(P);\mathbb{S}}$$

(ASTR)

$$\frac{P \qquad \mathbb{T} \sqsubseteq \mathbb{S}}{\mathbb{T} \sqsubseteq \mathbf{assert}(P);\mathbb{S}}$$

$I_{ExpnClnt}$

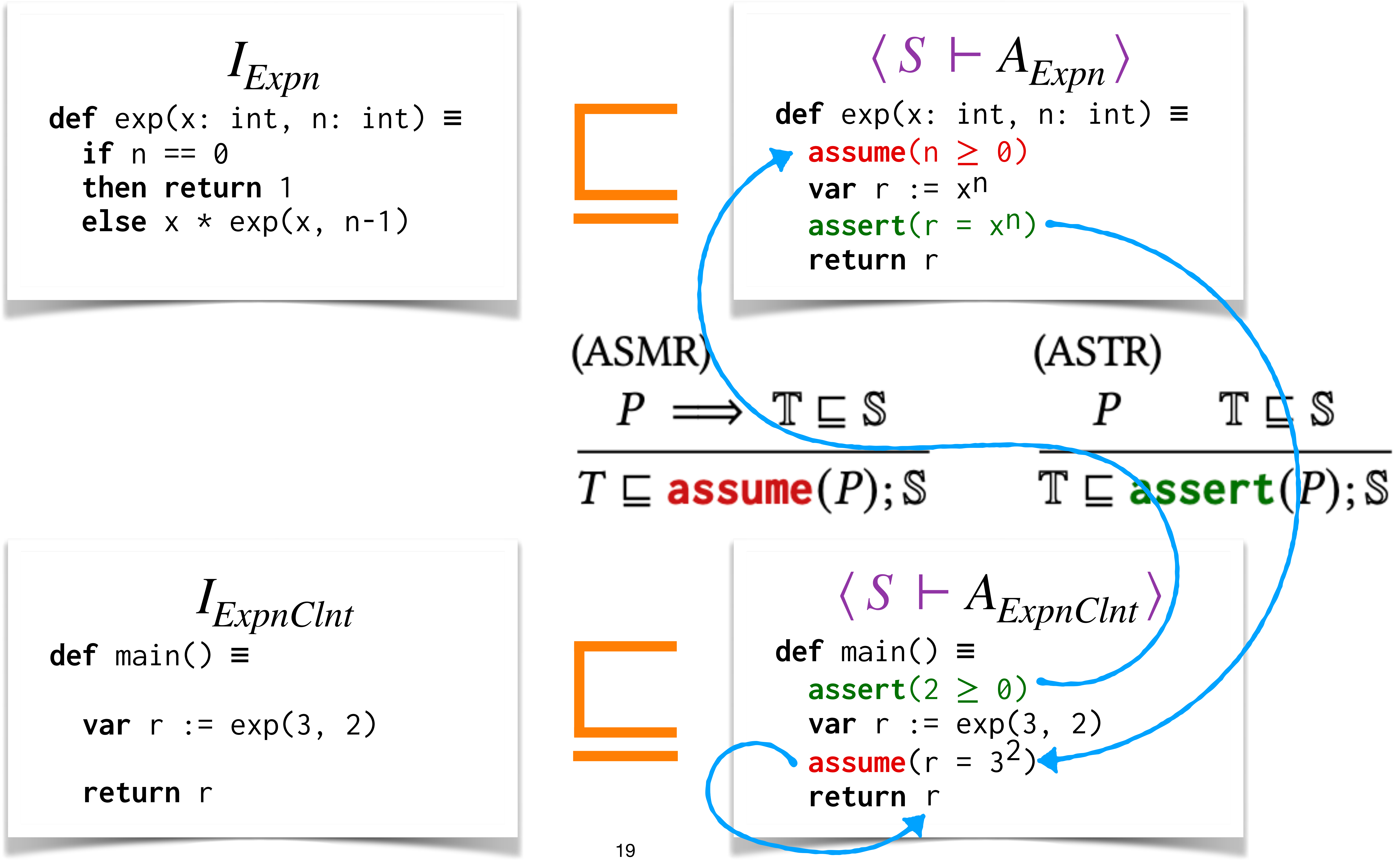
```
def main() ≡

  var r := exp(3, 2)

  return r
```

$\sqsubseteq$

$\langle S \vdash A_{ExpnClnt} \rangle$

```
def main() ≡
  assert(2 ≥ 0)
  var r := exp(3, 2)
  assume(r = 3²)
  return r
```

Stateless Conditional Refinement



I_{Expn}

```
def exp(x: int, n: int) ≡
  if n == 0
  then return 1
  else x * exp(x, n-1)
```

$\sqsubseteq$

$\langle S \vdash A_{Expn} \rangle$

```
def exp(x: int, n: int) ≡
  assume(n ≥ 0)
  var r := xⁿ
  assert(r = xⁿ)
  return r
```

$$\text{(ASMR)}\quad \frac{P \implies \mathbb{T} \sqsubseteq \mathbb{S}}{T \sqsubseteq \texttt{assume}(P); \mathbb{S}}$$

$$\text{(ASTR)}\quad \frac{P \qquad \mathbb{T} \sqsubseteq \mathbb{S}}{\mathbb{T} \sqsubseteq \texttt{assert}(P); \mathbb{S}}$$

$I_{ExpnClnt}$

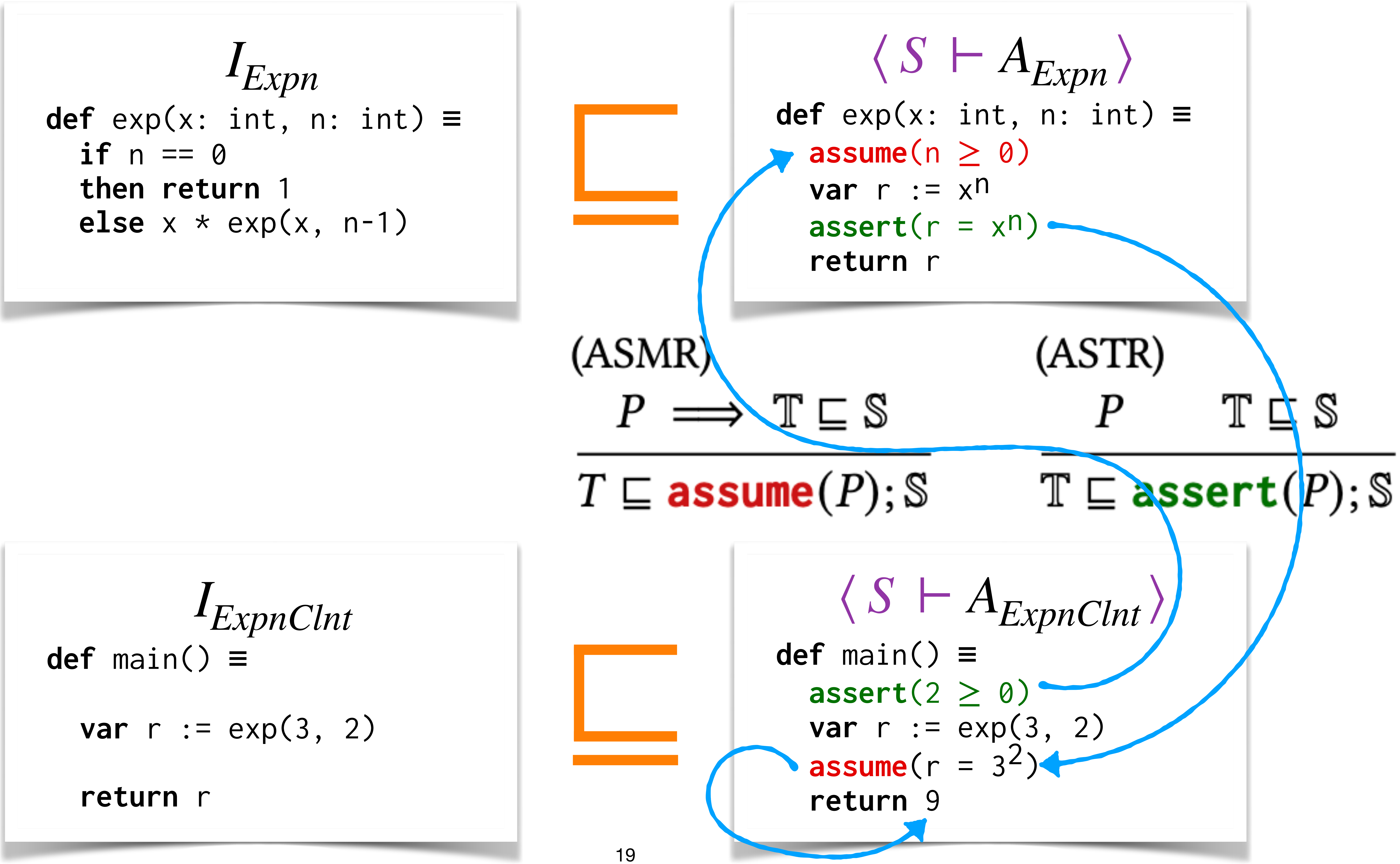
```
def main() ≡

  var r := exp(3, 2)

  return r
```

$\sqsubseteq$

$\langle S \vdash A_{ExpnClnt} \rangle$

```
def main() ≡
  assert(2 ≥ 0)
  var r := exp(3, 2)
  assume(r = 3²)
  return r
```

Stateless Conditional Refinement



I_{Expn}

```
def exp(x: int, n: int) ≡
  if n == 0
  then return 1
  else x * exp(x, n-1)
```

$\sqsubseteq$

$\langle S \vdash A_{Expn} \rangle$

```
def exp(x: int, n: int) ≡
  assume(n ≥ 0)
  var r := x^n
  assert(r = x^n)
  return r
```

(ASMR)

$$\frac{P \implies \mathbb{T} \sqsubseteq \mathbb{S}}{T \sqsubseteq \mathtt{assume}(P);\mathbb{S}}$$

(ASTR)

$$\frac{P \qquad \mathbb{T} \sqsubseteq \mathbb{S}}{\mathbb{T} \sqsubseteq \mathtt{assert}(P);\mathbb{S}}$$

$I_{ExpnClnt}$

```
def main() ≡

  var r := exp(3, 2)

  return r
```

$\sqsubseteq$

$\langle S \vdash A_{ExpnClnt} \rangle$

```
def main() ≡
  assert(2 ≥ 0)
  var r := exp(3, 2)
  assume(r = 3^2)
  return 9
```

Key Technical Pieces

Additional Features (paper/artifact) →

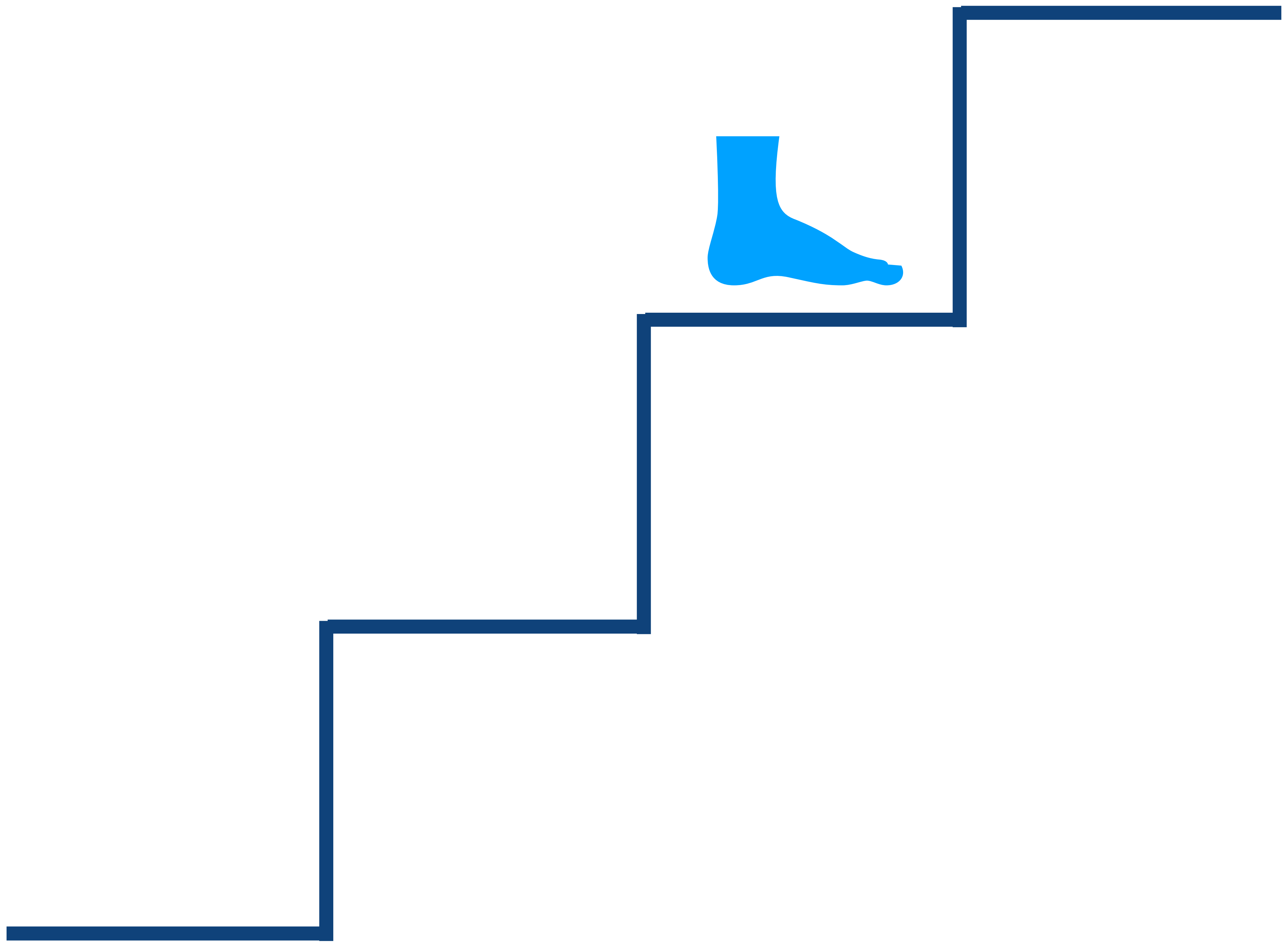
Stateful Conditional Refinement →

Separation logic conditions

Stateless Conditional Refinement →

Hoare logic conditions

Vanilla Refinement →



Stateful ASSUME/ASSERT

A_{Clnt}

```
def main() ≡

  init(2)




  set(0, 42)
```

A_{Map}

```
private map := λ k. 0

def init(sz: int) ≡

  skip



def set(k: int, v: int) ≡

  map := map[k ↤ v]
```

Stateful ASSUME/ASSERT

A_{Clnt}

```
def main() ≡

  init(2)



  set(0, 42)
```

A_{Map}

```
private map := λ k. 0

def init(sz: int) ≡
  skip


def set(k: int, v: int) ≡
  map := map[k ↤ v]
```

$$\forall \mathtt{sz}.\ \{\boxed{\mathit{pending}}\}\quad \mathtt{init(sz)}\ \{\ \ast_{\mathtt{k}\in[0,\mathtt{sz})}\ \mathtt{k} \mapsto_{\mathtt{Map}} \mathtt{0}\ \}$$

$$\forall \mathtt{k}\ \mathtt{v}.\ \{\ \mathtt{k} \mapsto_{\mathtt{Map}} \mathtt{v}\ \}\ \ \mathtt{get(k)}\quad \{\ \mathtt{r}.\ \mathtt{r} = \mathtt{v} \ast \mathtt{k} \mapsto_{\mathtt{Map}} \mathtt{v}\ \}$$

$$\forall \mathtt{k}\ \mathtt{v}.\ \{\ \exists w.\ \mathtt{k} \mapsto_{\mathtt{Map}} \mathtt{w}\ \}\ \ \mathtt{set(k,\ v)}\ \{\ \mathtt{k} \mapsto_{\mathtt{Map}} \mathtt{v}\ \}$$

Stateful ASSUME/ASSERT

A_{Clnt}

```
def main() ≡

  init(2)




  set(0, 42)
```

A_{Map}

```
private map := λ k. 0

def init(sz: int) ≡

  skip



def set(k: int, v: int) ≡

  map := map[k ↤ v]
```

Stateful ASSUME/ASSERT

$$\langle\, S \vdash A_{Clnt} \,\rangle$$

```
def main() ≡

  ASSERT(pending)
  init(2)
  ASSUME($0 \mapsto_{Map} 0 * 1 \mapsto_{Map} 0$)



  ASSERT($(\exists w.0 \mapsto_{Map} w) * 1 \mapsto_{Map} 0$)
  set(0, 42)
  ASSUME($0 \mapsto_{Map} 42 *       1 \mapsto_{Map} 0$)
```

$$\langle\, S \vdash A_{Map} \,\rangle$$

```
private map := λ k. 0

def init(sz: int) ≡
  ASSUME(pending)
  skip
  ASSERT($\ast_{k \in [0,sz)} k \mapsto_{Map} 0$)


def set(k: int, v: int) ≡
  ASSUME($\exists w.k \mapsto_{Map} w$)
  map := map[k ↤ v]
  ASSERT($k \mapsto_{Map} v$)
```

Key Challenge:
Operationalizing Ownership

Stateful ASSUME/ASSERT

$$\langle\, S \vdash A_{Clnt} \,\rangle$$

```
def main() ≡

  ASSERT(pending)
  init(2)
  ASSUME(0 ↦Map 0 * 1 ↦Map 0)



  ASSERT((∃w.0 ↦Map w) * 1 ↦Map 0)
  set(0, 42)
  ASSUME(0 ↦Map 42 *      1 ↦Map 0)
```

$$\langle\, S \vdash A_{Map} \,\rangle$$

```
private map := λ k. 0

def init(sz: int) ≡
  ASSUME(pending)
  skip
  ASSERT(✱k∈[0,sz) k ↦Map 0)


def set(k: int, v: int) ≡
  ASSUME(∃w.k ↦Map w)
  map := map[k ↤ v]
  ASSERT(k ↦Map v)
```

Stateful ASSUME/ASSERT

Step 1: Add Resources

$$\langle\, S \vdash A_{Clnt} \,\rangle$$

```
def main() ≡

  ASSERT(r0 $\in$ $\boxed{pending}$)
  init(2)
  ASSUME(r1 $\in$ 0 $\mapsto_{Map}$ 0 * 1 $\mapsto_{Map}$ 0)



  ASSERT(r2 · fr $\in$ ($\exists w$. 0 $\mapsto_{Map}$ w) * 1 $\mapsto_{Map}$ 0)
  set(0, 42)
  ASSUME(r3 · fr $\in$ 0 $\mapsto_{Map}$ 42 *      1 $\mapsto_{Map}$ 0)
```

$$\langle\, S \vdash A_{Map} \,\rangle$$

```
private map := λ k. 0

def init(sz: int) ≡
  ASSUME(r0 $\in$ $\boxed{pending}$)
  skip
  ASSERT(r1 $\in$ $\ast_{k \in [0,sz)}$ k $\mapsto_{Map}$ 0)


def set(k: int, v: int) ≡
  ASSUME(r2 $\in$ $\exists w$. k $\mapsto_{Map}$ w)
  map := map[k ↤ v]
  ASSERT(r3 $\in$ k $\mapsto_{Map}$ v)
```

Stateful ASSUME/ASSERT

Step 1: Add Resources

$$\langle\, S \vdash A_{Clnt} \,\rangle$$

```
def main() ≡

  assert(r0 ∈ pending)
  init(2)
  assume(r1 ∈ 0 ↦Map 0 * 1 ↦Map 0)



  assert(r2 · fr ∈ (∃w.0 ↦Map w) * 1 ↦Map 0)
  set(0, 42)
  assume(r3 · fr ∈ 0 ↦Map 42 *      1 ↦Map 0)
```

$$\langle\, S \vdash A_{Map} \,\rangle$$

```
private map := λ k. 0

def init(sz: int) ≡
  assume(r0 ∈ pending)
  skip
  assert(r1 ∈ ∗k∈[0,sz) k ↦Map 0)


def set(k: int, v: int) ≡
  assume(r2 ∈ ∃w.k ↦Map w)
  map := map[k ↤ v]
  assert(r3 ∈ k ↦Map v)
```

Stateful ASSUME/ASSERT

Step 1: Add Resources

$$\langle\, S \vdash A_{Clnt} \,\rangle$$

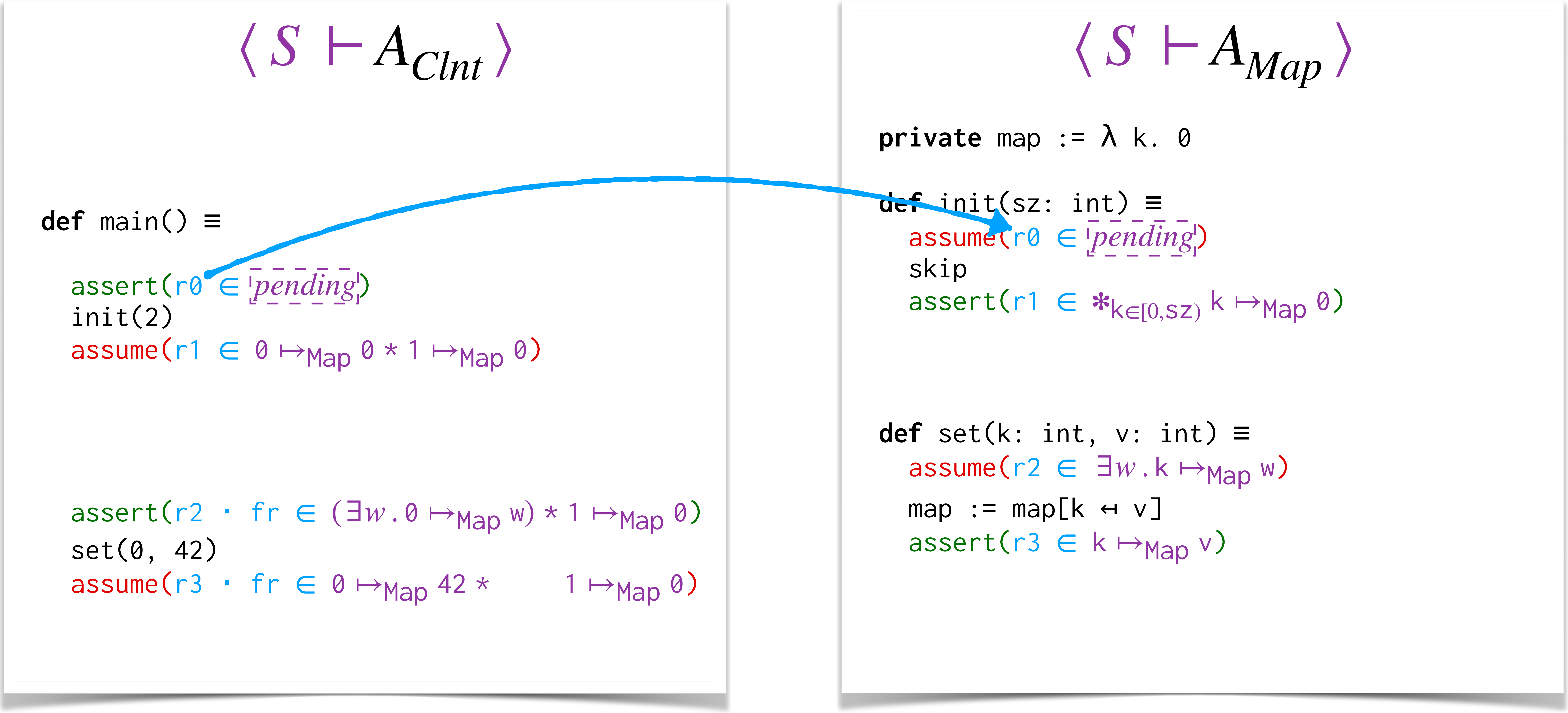
```
def main() ≡

  assert(r0 ∈ ⌜pending⌝)
  init(2)
  assume(r1 ∈ 0 ↦Map 0 * 1 ↦Map 0)



  assert(r2 · fr ∈ (∃w.0 ↦Map w) * 1 ↦Map 0)
  set(0, 42)
  assume(r3 · fr ∈ 0 ↦Map 42 *       1 ↦Map 0)
```

$$\langle\, S \vdash A_{Map} \,\rangle$$

```
private map := λ k. 0

def init(sz: int) ≡
  assume(r0 ∈ ⌜pending⌝)
  skip
  assert(r1 ∈ ∗k∈[0,sz) k ↦Map 0)


def set(k: int, v: int) ≡
  assume(r2 ∈ ∃w.k ↦Map w)
  map := map[k ↤ v]
  assert(r3 ∈ k ↦Map v)
```

Stateful ASSUME/ASSERT

Step 1: Add Resources



$\langle S \vdash A_{Clnt} \rangle$

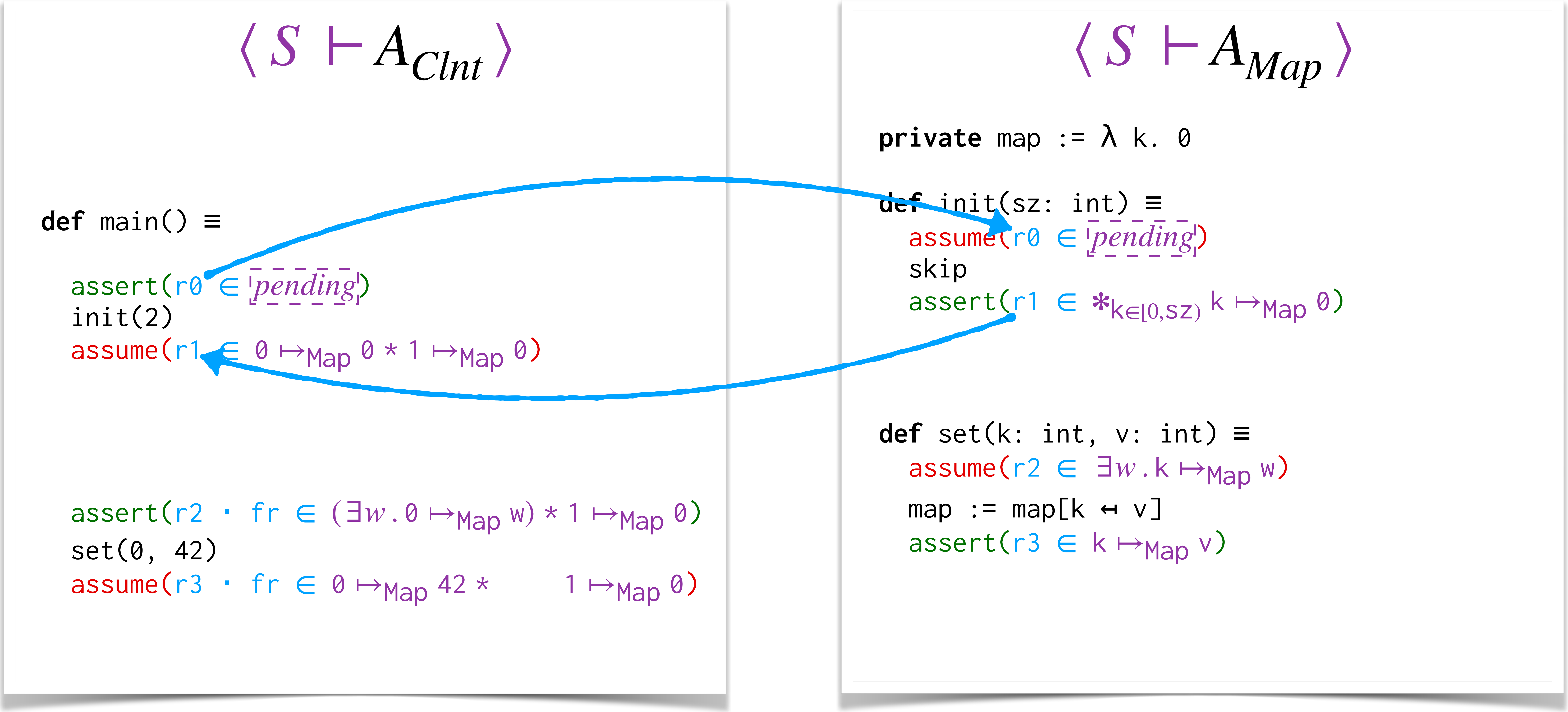
```
def main() ≡

  assert(r0 ∈ pending)
  init(2)
  assume(r1 ∈ 0 ↦Map 0 * 1 ↦Map 0)


  assert(r2 · fr ∈ (∃w.0 ↦Map w) * 1 ↦Map 0)
  set(0, 42)
  assume(r3 · fr ∈ 0 ↦Map 42 *      1 ↦Map 0)
```

$\langle S \vdash A_{Map} \rangle$

```
private map := λ k. 0

def init(sz: int) ≡
  assume(r0 ∈ pending)
  skip
  assert(r1 ∈ ∗k∈[0,sz) k ↦Map 0)


def set(k: int, v: int) ≡
  assume(r2 ∈ ∃w.k ↦Map w)
  map := map[k ↤ v]
  assert(r3 ∈ k ↦Map v)
```

Stateful ASSUME/ASSERT

Step 1: Add Resources



$$\langle\, S \vdash A_{Clnt} \,\rangle$$

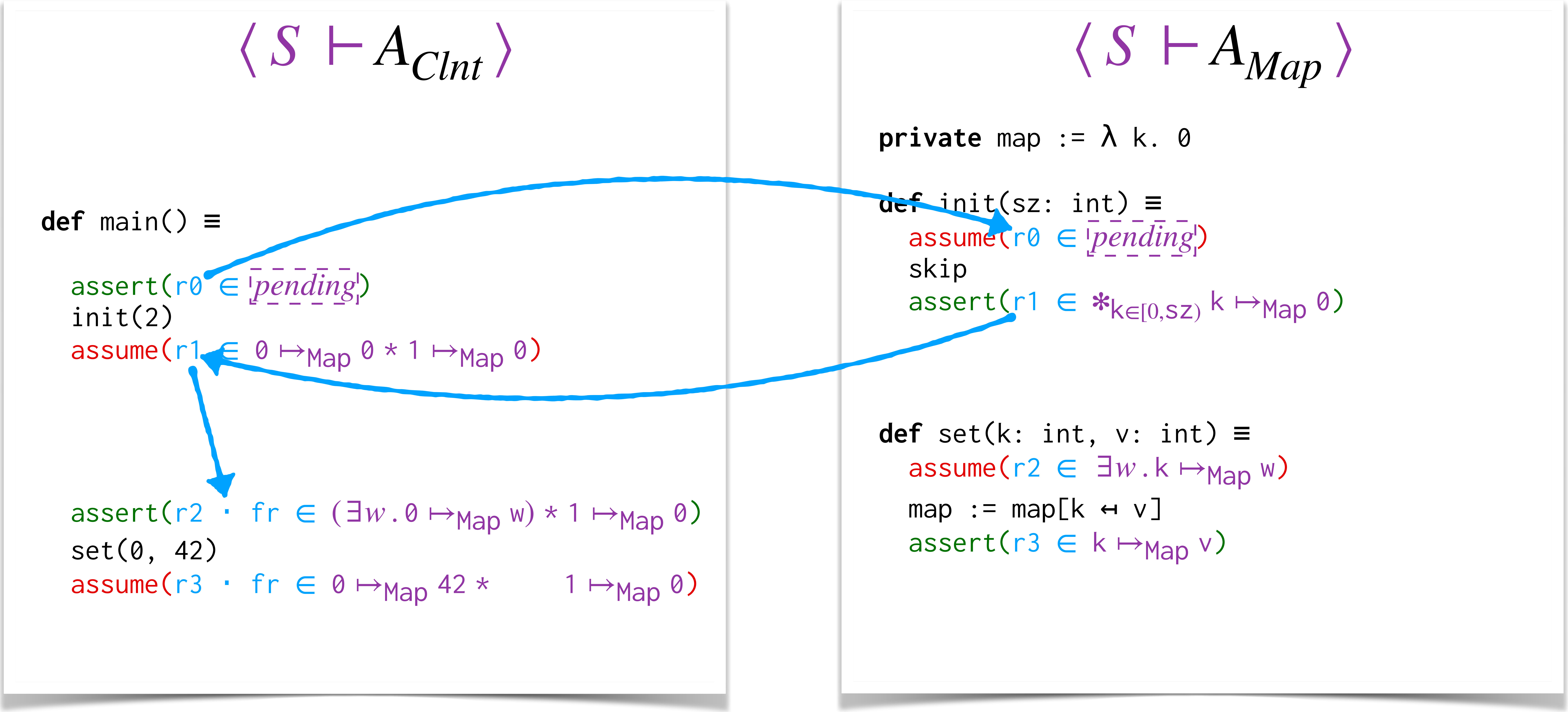
```
def main() ≡
  assert(r0 ∈ $\boxed{pending}$)
  init(2)
  assume(r1 ∈ $0 \mapsto_{Map} 0 * 1 \mapsto_{Map} 0$)

  assert(r2 · fr ∈ $(\exists w.0 \mapsto_{Map} w) * 1 \mapsto_{Map} 0$)
  set(0, 42)
  assume(r3 · fr ∈ $0 \mapsto_{Map} 42 *$      $1 \mapsto_{Map} 0$)
```

$$\langle\, S \vdash A_{Map} \,\rangle$$

```
private map := λ k. 0

def init(sz: int) ≡
  assume(r0 ∈ $\boxed{pending}$)
  skip
  assert(r1 ∈ $\ast_{k \in [0,sz)}\ k \mapsto_{Map} 0$)

def set(k: int, v: int) ≡
  assume(r2 ∈ $\exists w.k \mapsto_{Map} w$)
  map := map[k ↤ v]
  assert(r3 ∈ $k \mapsto_{Map} v$)
```

Stateful ASSUME/ASSERT

Step 1: Add Resources



$\langle S \vdash A_{Clnt} \rangle$

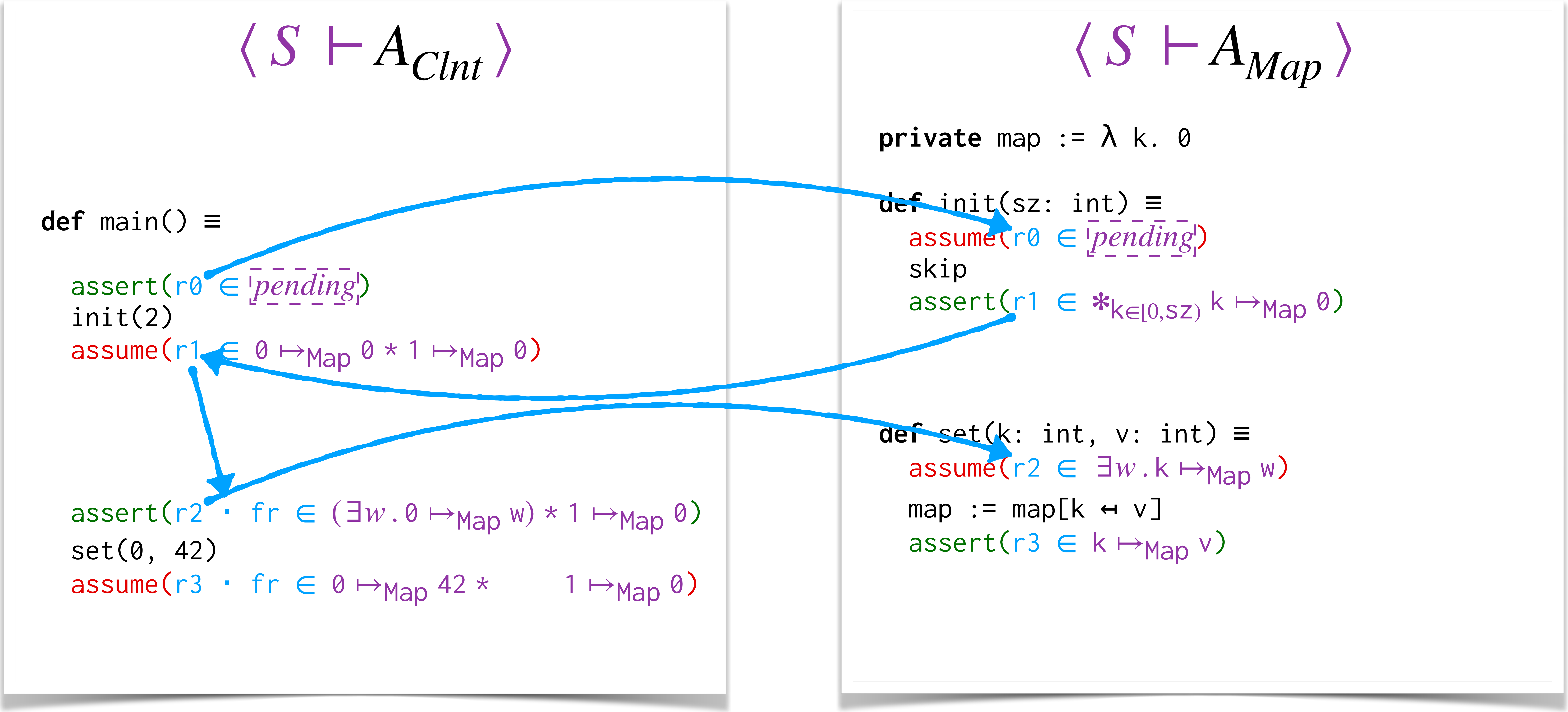
```
def main() ≡
  assert(r0 ∈ $\boxed{\mathit{pending}}$)
  init(2)
  assume(r1 ∈ 0 $\mapsto_{\text{Map}}$ 0 * 1 $\mapsto_{\text{Map}}$ 0)

  assert(r2 · fr ∈ ($\exists w$. 0 $\mapsto_{\text{Map}}$ w) * 1 $\mapsto_{\text{Map}}$ 0)
  set(0, 42)
  assume(r3 · fr ∈ 0 $\mapsto_{\text{Map}}$ 42 *     1 $\mapsto_{\text{Map}}$ 0)
```

$\langle S \vdash A_{Map} \rangle$

```
private map := λ k. 0

def init(sz: int) ≡
  assume(r0 ∈ $\boxed{\mathit{pending}}$)
  skip
  assert(r1 ∈ $\circledast_{k \in [0,sz)}$ k $\mapsto_{\text{Map}}$ 0)

def set(k: int, v: int) ≡
  assume(r2 ∈ $\exists w$.k $\mapsto_{\text{Map}}$ w)
  map := map[k ↤ v]
  assert(r3 ∈ k $\mapsto_{\text{Map}}$ v)
```

Stateful ASSUME/ASSERT

Step 1: Add Resources



$\langle S \vdash A_{Clnt} \rangle$

def main() ≡

  assert(r0 $\in \boxed{pending}$)
  init(2)
  assume(r1 $\in$ 0 $\mapsto_{\text{Map}}$ 0 $*$ 1 $\mapsto_{\text{Map}}$ 0)

  assert(r2 $\cdot$ fr $\in$ ($\exists w$. 0 $\mapsto_{\text{Map}}$ w) $*$ 1 $\mapsto_{\text{Map}}$ 0)
  set(0, 42)
  assume(r3 $\cdot$ fr $\in$ 0 $\mapsto_{\text{Map}}$ 42 $*$ 1 $\mapsto_{\text{Map}}$ 0)

$\langle S \vdash A_{Map} \rangle$

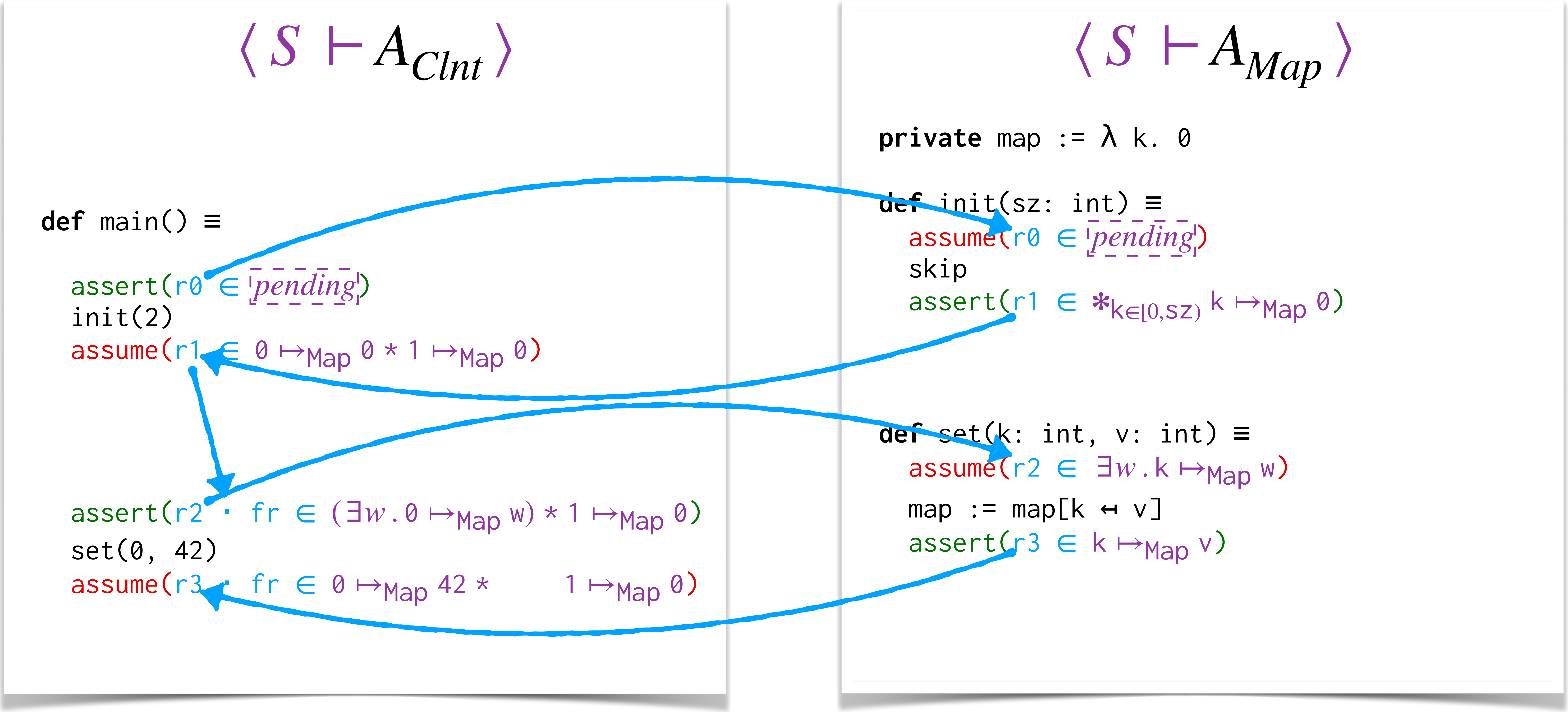
private map := λ k. 0

def init(sz: int) ≡
  assume(r0 $\in \boxed{pending}$)
  skip
  assert(r1 $\in \ast_{k \in [0,sz)}$ k $\mapsto_{\text{Map}}$ 0)

def set(k: int, v: int) ≡
  assume(r2 $\in \exists w$. k $\mapsto_{\text{Map}}$ w)
  map := map[k ↤ v]
  assert(r3 $\in$ k $\mapsto_{\text{Map}}$ v)

Stateful ASSUME/ASSERT

Step 1: Add Resources



$\langle\, S \vdash A_{Clnt} \,\rangle$

```
def main() ≡

  assert(r0 ∈ pending)
  init(2)
  assume(r1 ∈ 0 ↦Map 0 * 1 ↦Map 0)


  assert(r2 · fr ∈ (∃w.0 ↦Map w) * 1 ↦Map 0)
  set(0, 42)
  assume(r3 · fr ∈ 0 ↦Map 42 *     1 ↦Map 0)
```

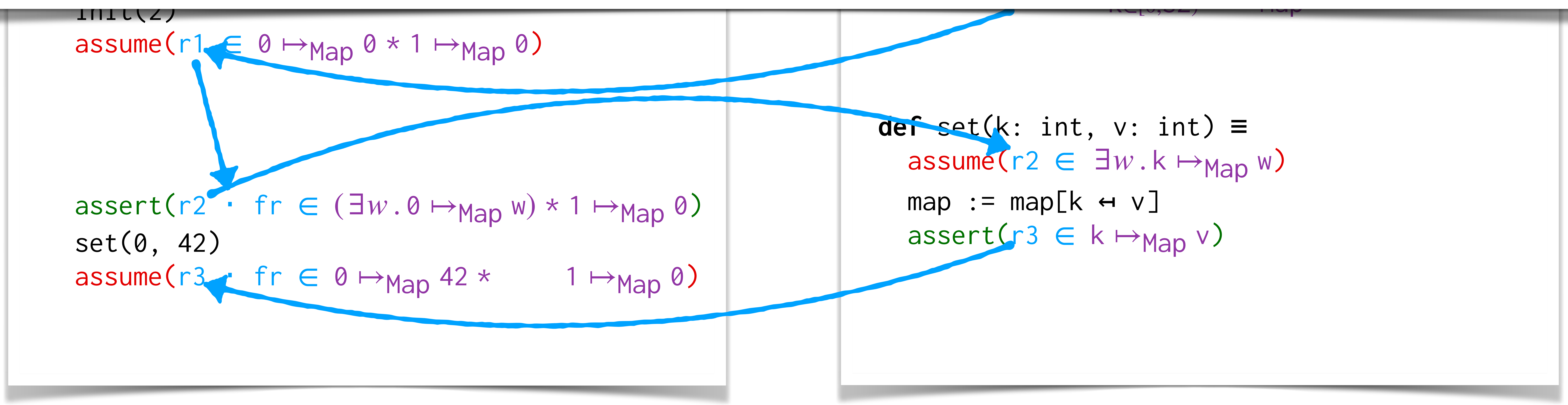
$\langle\, S \vdash A_{Map} \,\rangle$

```
private map := λ k. 0

def init(sz: int) ≡
  assume(r0 ∈ pending)
  skip
  assert(r1 ∈ *k∈[0,sz) k ↦Map 0)


def set(k: int, v: int) ≡
  assume(r2 ∈ ∃w.k ↦Map w)
  map := map[k ↤ v]
  assert(r3 ∈ k ↦Map v)
```

Stateful ASSUME/ASSERT

Step 1: Add Resources

How do we transfer the resources operationally?



```
init(2)
assume(r1 ∈ 0 ↦Map 0 * 1 ↦Map 0)


assert(r2 · fr ∈ (∃w.0 ↦Map w) * 1 ↦Map 0)
set(0, 42)
assume(r3 · fr ∈ 0 ↦Map 42 *      1 ↦Map 0)
```

```
def set(k: int, v: int) ≡
  assume(r2 ∈ ∃w.k ↦Map w)
  map := map[k ↤ v]
  assert(r3 ∈ k ↦Map v)
```

Stateful ASSUME/ASSERT

Step 1: Add Resources

How do we transfer the resources operationally?

First attempt: pass as arguments (returns)

Stateful ASSUME/ASSERT

Step 1: Add Resources

$$\langle\, S \vdash A_{Clnt} \,\rangle$$

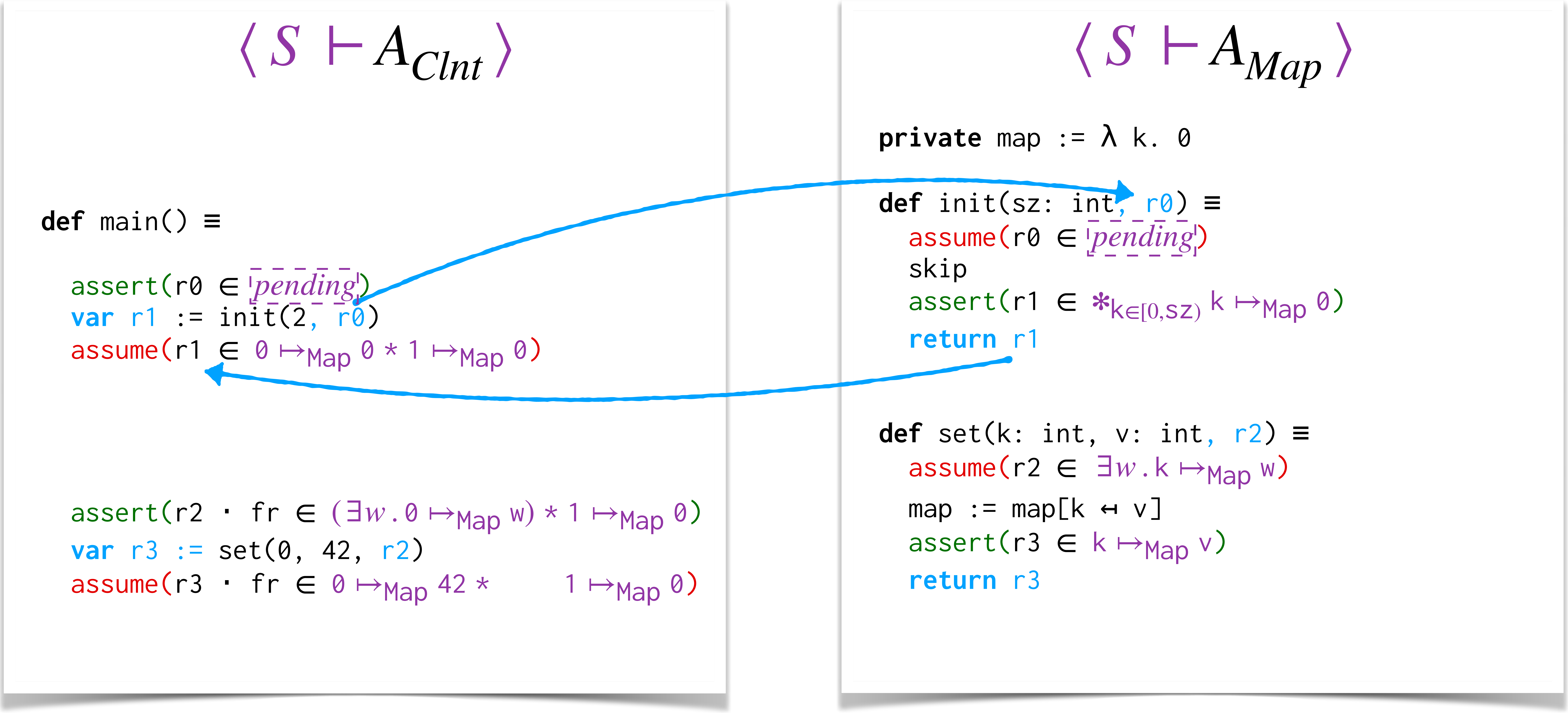
```
def main() ≡

  assert(r0 ∈ $\boxed{pending}$)
  init(2)
  assume(r1 ∈ $0 \mapsto_{Map} 0 * 1 \mapsto_{Map} 0$)



  assert(r2 · fr ∈ $(\exists w.\, 0 \mapsto_{Map} w) * 1 \mapsto_{Map} 0$)
  set(0, 42)
  assume(r3 · fr ∈ $0 \mapsto_{Map} 42 *$       $1 \mapsto_{Map} 0$)
```

$$\langle\, S \vdash A_{Map} \,\rangle$$

```
private map := λ k. 0

def init(sz: int) ≡
  assume(r0 ∈ $\boxed{pending}$)
  skip
  assert(r1 ∈ $\ast_{k \in [0,sz)}\, k \mapsto_{Map} 0$)


def set(k: int, v: int) ≡
  assume(r2 ∈ $\exists w.\, k \mapsto_{Map} w$)
  map := map[k ↤ v]
  assert(r3 ∈ $k \mapsto_{Map} v$)
```

Stateful ASSUME/ASSERT

Step 2: Pass Resources Explicitly...?



$$\langle\, S \vdash A_{Clnt} \,\rangle$$

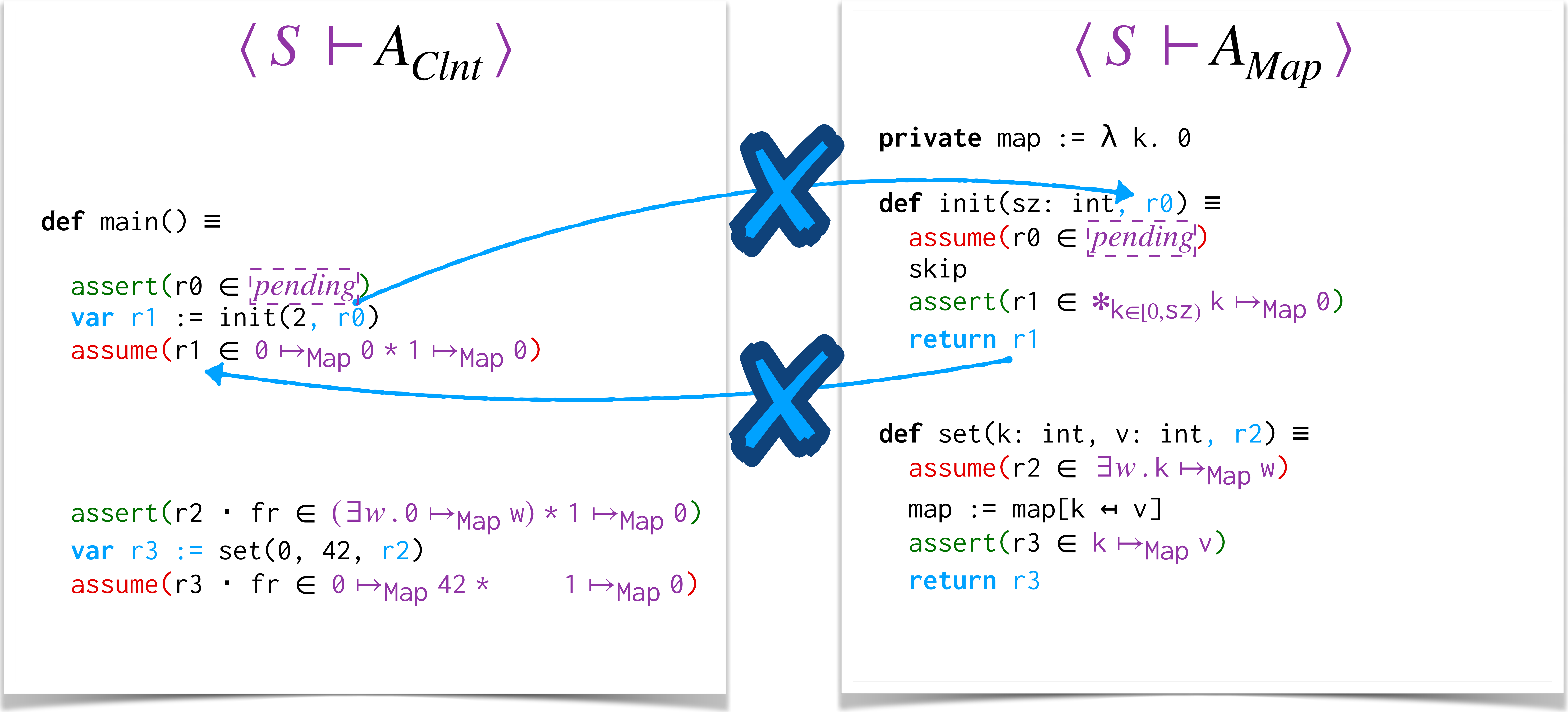
```
def main() ≡

  assert(r0 ∈ pending)
  var r1 := init(2, r0)
  assume(r1 ∈ 0 ↦Map 0 * 1 ↦Map 0)



  assert(r2 · fr ∈ (∃w.0 ↦Map w) * 1 ↦Map 0)
  var r3 := set(0, 42, r2)
  assume(r3 · fr ∈ 0 ↦Map 42 *       1 ↦Map 0)
```

$$\langle\, S \vdash A_{Map} \,\rangle$$

```
private map := λ k. 0

def init(sz: int, r0) ≡
  assume(r0 ∈ pending)
  skip
  assert(r1 ∈ ∗k∈[0,sz) k ↦Map 0)
  return r1

def set(k: int, v: int, r2) ≡
  assume(r2 ∈ ∃w.k ↦Map w)
  map := map[k ↤ v]
  assert(r3 ∈ k ↦Map v)
  return r3
```

Stateful ASSUME/ASSERT

Step 2: Pass Resources Explicitly...?



$\langle\, S \vdash A_{Clnt} \,\rangle$

```
def main() ≡

  assert(r0 ∈ pending)
  var r1 := init(2, r0)
  assume(r1 ∈ 0 ↦Map 0 * 1 ↦Map 0)



  assert(r2 · fr ∈ (∃w.0 ↦Map w) * 1 ↦Map 0)
  var r3 := set(0, 42, r2)
  assume(r3 · fr ∈ 0 ↦Map 42 *      1 ↦Map 0)
```

$\langle\, S \vdash A_{Map} \,\rangle$

```
private map := λ k. 0

def init(sz: int, r0) ≡
  assume(r0 ∈ pending)
  skip
  assert(r1 ∈ ∗k∈[0,sz) k ↦Map 0)
  return r1

def set(k: int, v: int, r2) ≡
  assume(r2 ∈ ∃w.k ↦Map w)
  map := map[k ↤ v]
  assert(r3 ∈ k ↦Map v)
  return r3
```

Motivating Example

With Conditional Contextual Refinement

I_{Map}

```
private data := NULL


def init(sz: int) ≡
  data := calloc(sz)

def get(k: int) ≡
  return *(data + k)


def set(k: int, v: int) ≡
  *(data + k) := v
```

$\sqsubseteq$

M_{Map}

```
private map := λ k. 0
private size := 0

def init(sz: int) ≡
  size := sz

def get(k: int) ≡
  assume(0 ≤ k < size)
  return map[k]

def set(k: int, v: int) ≡
  assume(0 ≤ k < size)
  map := map[k ↤ v]
```

$\sqsubseteq$

A_{Map}

```
private map := λ k. 0


def init(sz: int) ≡
  skip

def get(k: int) ≡

  return map[k]

def set(k: int, v: int) ≡

  map := map[k ↤ v]
```

$\forall sz.\ \{\boxed{\mathit{pending}}\}\ \mathtt{init(sz)}\ \{\top\}$
$\forall k.\ \{\top\}\ \mathtt{get(k)}\ \{\top\}$
$\forall k\ v.\ \{\top\}\ \mathtt{set(k,\ v)}\ \{\top\}$

$\forall sz.\ \{\boxed{\mathit{pending}}\}\ \mathtt{init(sz)}\ \{\ \ast_{k\in[0,sz)}\ k \mapsto_{Map} 0\ \}$
$\forall k\ v.\ \{\ k \mapsto_{Map} v\ \}\ \mathtt{get(k)}\ \{\ r.\ r = v \ast k \mapsto_{Map} v\ \}$
$\forall k\ v.\ \{\ \exists w.\ k \mapsto_{Map} w\ \}\ \mathtt{set(k,\ v)}\ \{\ k \mapsto_{Map} v\ \}$

Motivating Example

With Conditional Contextual Refinement

I_{Map}

```
def get(k: int, r0) ≡
  return (*(data + k), r1)
```

$\sqsubseteq$

M_{Map}

```
def get(k: int, r0) ≡
  assume(0 ≤ k < size)
  return (map[k], r1)
```

$\sqsubseteq$

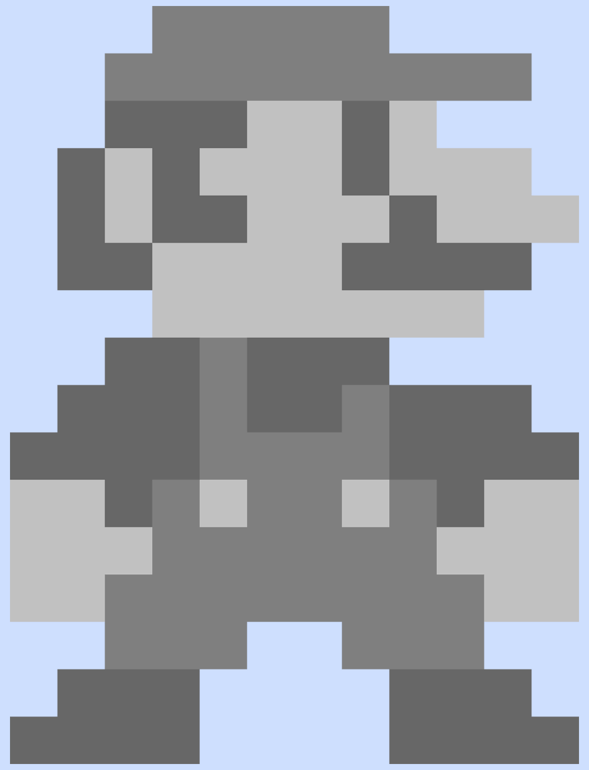
A_{Map}

```
def get(k: int, r0) ≡

  return (map[k], r1)
```

$\forall k.\ \{\top\}\quad \mathtt{get(k)}\quad \{\top\}$

$\forall k\ v. \{k \mapsto_{Map} v\}\ \mathtt{get(k)}\quad \{r. r = v * k \mapsto_{Map} v\}$

$\langle\, S \vdash A_{Clnt} \,\rangle$



$\langle\, S \vdash A_{Map} \,\rangle$

$\langle S \vdash A_{Clnt} \rangle$

$\langle S \vdash A_{Map} \rangle$

Key Idea II: Dual Non-determinism (Combining Demonic and Angelic)

$\langle\, S \vdash A_{Clnt} \,\rangle$

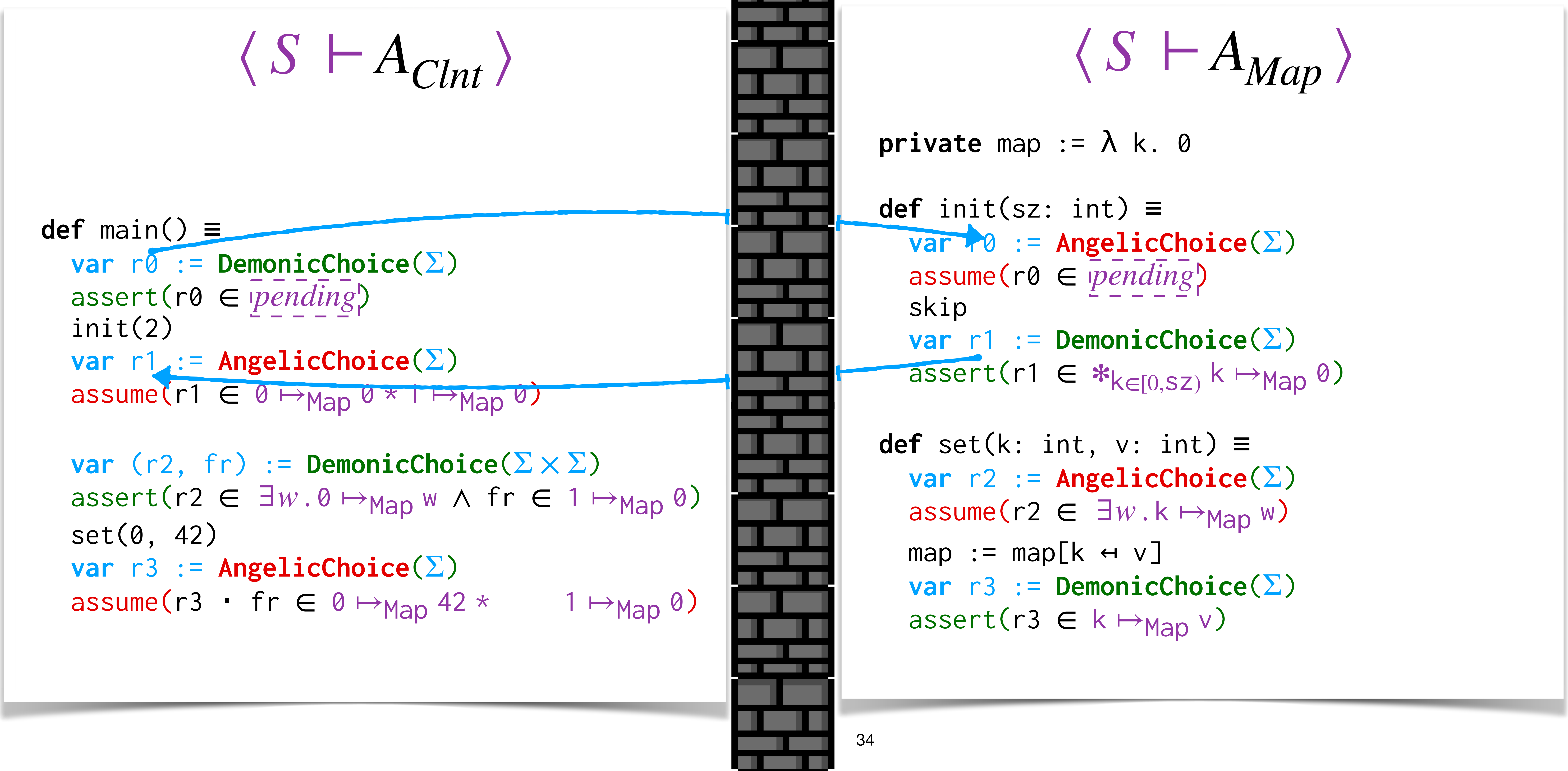
$\langle\, S \vdash A_{Map} \,\rangle$

$\langle\, S \vdash A_{Clnt} \,\rangle$

$\langle\, S \vdash A_{Map} \,\rangle$

Dual Non-Determinism

Pass Resources Implicitly



$\langle S \vdash A_{Clnt} \rangle$

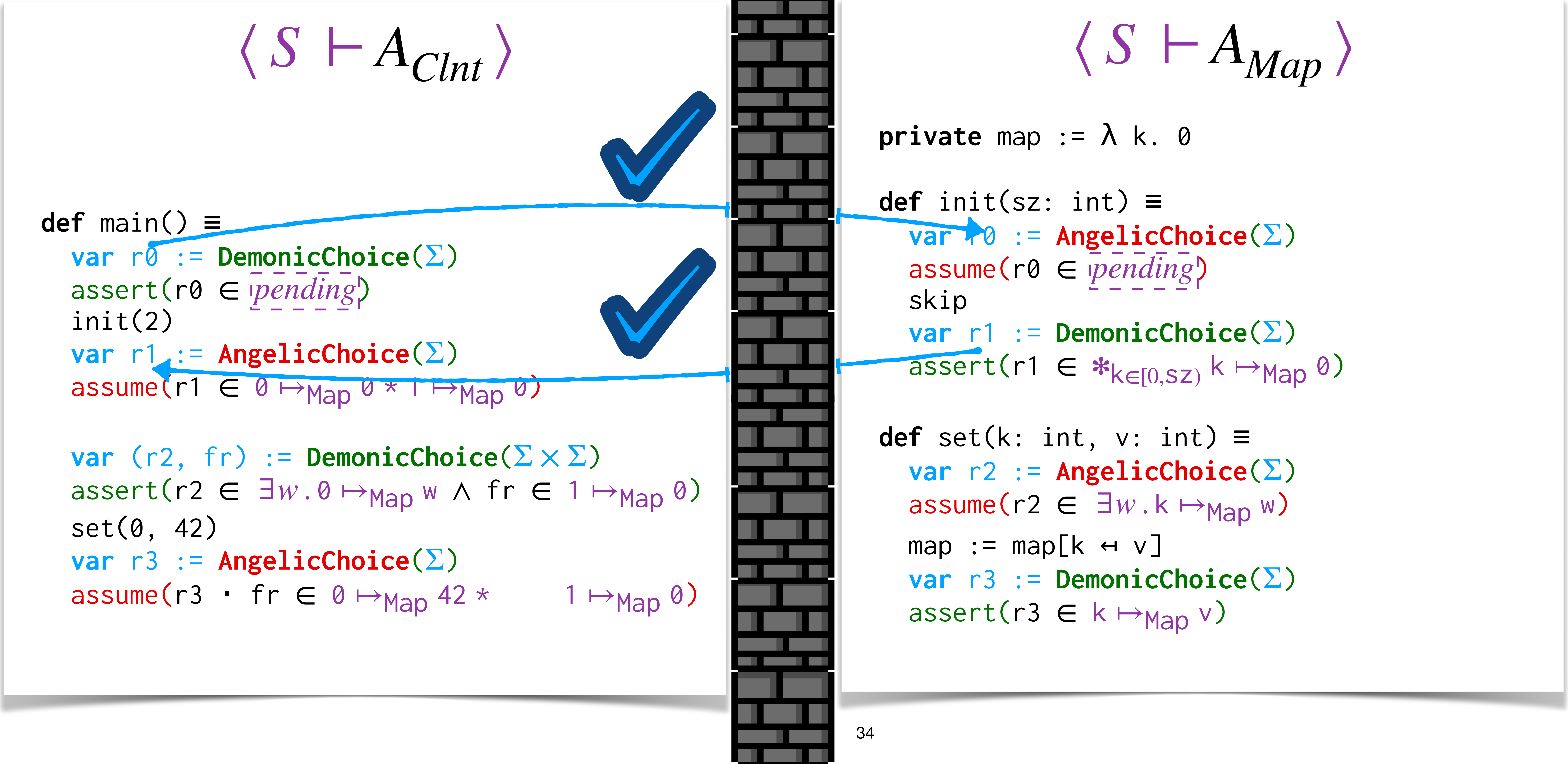
```
def main() ≡
  var r0 := DemonicChoice(Σ)
  assert(r0 ∈ pending)
  init(2)
  var r1 := AngelicChoice(Σ)
  assume(r1 ∈ 0 ↦Map 0 * 1 ↦Map 0)

  var (r2, fr) := DemonicChoice(Σ × Σ)
  assert(r2 ∈ ∃w.0 ↦Map w ∧ fr ∈ 1 ↦Map 0)
  set(0, 42)
  var r3 := AngelicChoice(Σ)
  assume(r3 · fr ∈ 0 ↦Map 42 *        1 ↦Map 0)
```

$\langle S \vdash A_{Map} \rangle$

```
private map := λ k. 0

def init(sz: int) ≡
  var r0 := AngelicChoice(Σ)
  assume(r0 ∈ pending)
  skip
  var r1 := DemonicChoice(Σ)
  assert(r1 ∈ ✱k∈[0,sz) k ↦Map 0)

def set(k: int, v: int) ≡
  var r2 := AngelicChoice(Σ)
  assume(r2 ∈ ∃w.k ↦Map w)
  map := map[k ↤ v]
  var r3 := DemonicChoice(Σ)
  assert(r3 ∈ k ↦Map v)
```

Dual Non-Determinism

Pass Resources Implicitly



$\langle\, S \vdash A_{Clnt} \,\rangle$

```
def main() ≡
  var r0 := DemonicChoice(Σ)
  assert(r0 ∈ pending)
  init(2)
  var r1 := AngelicChoice(Σ)
  assume(r1 ∈ 0 ↦Map 0 * 1 ↦Map 0)

  var (r2, fr) := DemonicChoice(Σ × Σ)
  assert(r2 ∈ ∃w.0 ↦Map w ∧ fr ∈ 1 ↦Map 0)
  set(0, 42)
  var r3 := AngelicChoice(Σ)
  assume(r3 · fr ∈ 0 ↦Map 42 *       1 ↦Map 0)
```

$\langle\, S \vdash A_{Map} \,\rangle$

```
private map := λ k. 0

def init(sz: int) ≡
  var r0 := AngelicChoice(Σ)
  assume(r0 ∈ pending)
  skip
  var r1 := DemonicChoice(Σ)
  assert(r1 ∈ ✻k∈[0,sz) k ↦Map 0)

def set(k: int, v: int) ≡
  var r2 := AngelicChoice(Σ)
  assume(r2 ∈ ∃w.k ↦Map w)
  map := map[k ↤ v]
  var r3 := DemonicChoice(Σ)
  assert(r3 ∈ k ↦Map v)
```

Dual Non-Determinism

Pass Resources Implicitly

$\langle S \vdash A_{Clnt} \rangle$

def main() ≡

  r0 $\in$ *pending* $\Longrightarrow$ 🟩

  init(2)

  🟥 $\Longrightarrow$ r1 $\in$ 0 $\mapsto_{\mathrm{Map}}$ 0 * 1 $\mapsto_{\mathrm{Map}}$ 0

  r2 $\in \exists w.$0 $\mapsto_{\mathrm{Map}}$ w $\Longrightarrow$ 🟩

  var fr $\in$ 1 $\mapsto_{\mathrm{Map}}$ 0

  set(0, 42)

  🟥 $\Longrightarrow$ r3 $\in$ 0 $\mapsto_{\mathrm{Map}}$ 42

$\langle S \vdash A_{Map} \rangle$

private map := λ k. 0

def init(sz: int) ≡

  🟥 $\Longrightarrow$ r0 $\in$ *pending*

  skip

  r1 $\in \ast_{k \in [0, sz)}$ k $\mapsto_{\mathrm{Map}}$ 0 $\Longrightarrow$ 🟩

def set(k: int, v: int) ≡

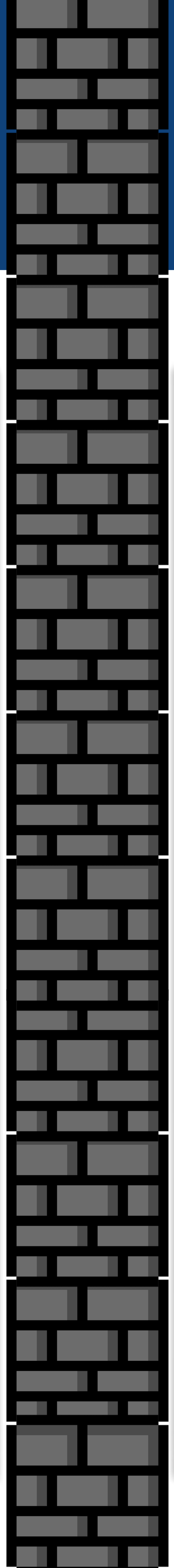
  🟥 $\Longrightarrow$ r2 $\in \exists w.$k $\mapsto_{\mathrm{Map}}$ w

  map := map[k ↤ v]

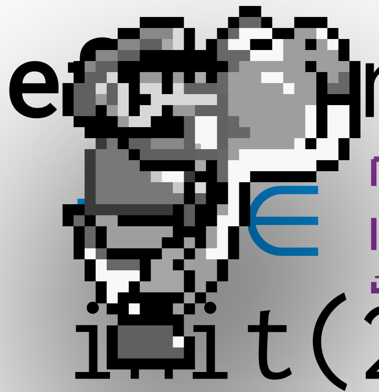
  r3 $\in$ k $\mapsto_{\mathrm{Map}}$ v $\Longrightarrow$ 🟩

Dual Non-Determinism

Pass Resources Implicitly



$\langle S \vdash A_{Clnt} \rangle$



```
def main() ≡
   r0 ∈ pending ⟹🟩
   init(2)
  🟥⟹ r1 ∈ 0 ↦Map 0 * 1 ↦Map 0


   r2 ∈ ∃w.0 ↦Map w ⟹🟩
   var fr ∈ 1 ↦Map 0
   set(0, 42)
  🟥⟹ r3 ∈ 0 ↦Map 42
```

$\langle S \vdash A_{Map} \rangle$

```
private map := λ k. 0
def init(sz: int) ≡
  🟥⟹ r0 ∈ pending
   skip
   r1 ∈ *k∈[0,sz) k ↦Map 0 ⟹🟩

def set(k: int, v: int) ≡
  🟥⟹ r2 ∈ ∃w.k ↦Map w
   map := map[k ↤ v]

   r3 ∈ k ↦Map v ⟹🟩
```

Dual Non-Determinism

Pass Resources Implicitly

$\langle S \vdash A_{Clnt} \rangle$

def main() ≡

r0 ∈ $\mathit{pending}$ ⟹ 🟩

init(2)

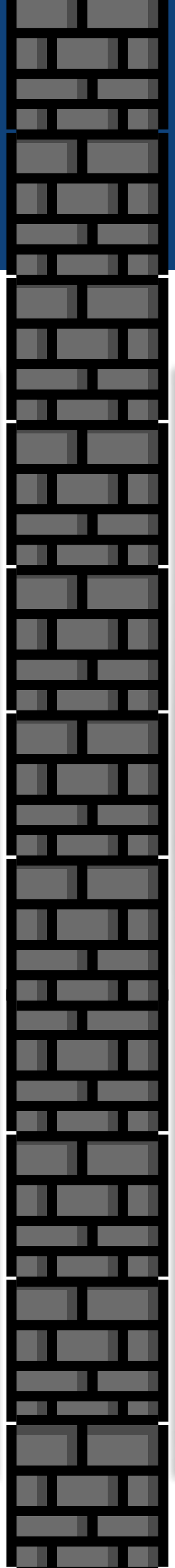
🟥 ⟹ r1 ∈ 0 $\mapsto_{\text{Map}}$ 0 * 1 $\mapsto_{\text{Map}}$ 0

r2 ∈ $\exists w$.0 $\mapsto_{\text{Map}}$ w ⟹ 🟩

var fr ∈ 1 $\mapsto_{\text{Map}}$ 0

set(0, 42)

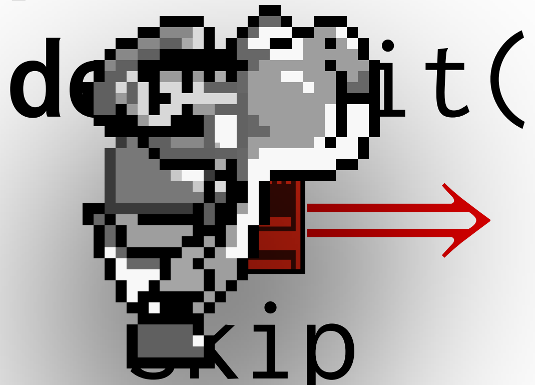
🟥 ⟹ r3 ∈ 0 $\mapsto_{\text{Map}}$ 42



$\langle S \vdash A_{Map} \rangle$

private map := λ k. 0

def init(sz: int) ≡



🟥 ⟹ r0 ∈ $\mathit{pending}$

skip

r1 ∈ $\ast_{k \in [0,sz)}$ k $\mapsto_{\text{Map}}$ 0 ⟹ 🟩

def set(k: int, v: int) ≡

🟥 ⟹ r2 ∈ $\exists w$.k $\mapsto_{\text{Map}}$ w

map := map[k ↤ v]

r3 ∈ k $\mapsto_{\text{Map}}$ v ⟹ 🟩

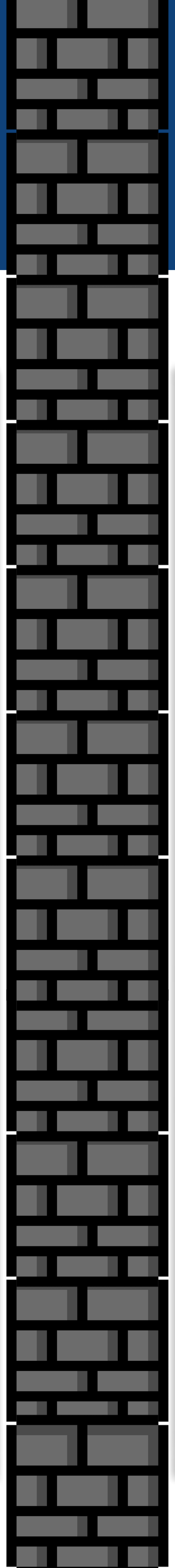
Dual Non-Determinism

Pass Resources Implicitly

$\langle S \vdash A_{Clnt} \rangle$

def main() ≡
  r0 ∈ *pending* ⟹ 🟩
  init(2)
  🟥 ⟹ r1 ∈ $0 \mapsto_{Map} 0 * 1 \mapsto_{Map} 0$

  r2 ∈ $\exists w. 0 \mapsto_{Map} w$ ⟹ 🟩
  var fr ∈ $1 \mapsto_{Map} 0$
  set(0, 42)
  🟥 ⟹ r3 ∈ $0 \mapsto_{Map} 42$



$\langle S \vdash A_{Map} \rangle$

private map := λ k. 0
def init(sz: int) ≡
  🟥 ⟹ r0 ∈ *pending*
  skip
  r1 ∈ $\ast_{k\in[0,sz)} k \mapsto_{Map} 0$ ⟹ 🟩

def set(k: int, v: int) ≡
  🟥 ⟹ r2 ∈ $\exists w. k \mapsto_{Map} w$
  map := map[k ↤ v]

  r3 ∈ $k \mapsto_{Map} v$ ⟹ 🟩

Dual Non-Determinism

Pass Resources Implicitly

$\langle S \vdash A_{Clnt} \rangle$

def main() ≡

  r0 ∈ *pending*

  init(2)

  r1 ∈ $0 \mapsto_{Map} 0 * 1 \mapsto_{Map} 0$

  r2 ∈ $\exists w.\, 0 \mapsto_{Map} w$

  var fr ∈ $1 \mapsto_{Map} 0$

  set(0, 42)

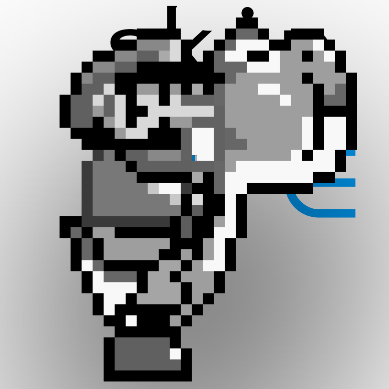
  r3 ∈ $0 \mapsto_{Map} 42$

$\langle S \vdash A_{Map} \rangle$

private map := λ k. 0

def init(sz: int) ≡

  r0 ∈ *pending*



  ∈ $*_{k \in [0,sz)}\, k \mapsto_{Map} 0$

def set(k: int, v: int) ≡

  r2 ∈ $\exists w.\, k \mapsto_{Map} w$

  map := map[k ↤ v]

  r3 ∈ $k \mapsto_{Map} v$

Dual Non-Determinism

Pass Resources Implicitly

$$\langle S \vdash A_{Clnt} \rangle$$

def main() ≡

  r0 ∈ *pending* ⟹ 🟩

  init(2)

  🟥⟹ r1 ∈ $0 \mapsto_{Map} 0 * 1 \mapsto_{Map} 0$

  r2 ∈ $\exists w.\, 0 \mapsto_{Map} w$ ⟹ 🟩

  var fr ∈ $1 \mapsto_{Map} 0$

  set(0, 42)

  🟥⟹ r3 ∈ $0 \mapsto_{Map} 42$

$$\langle S \vdash A_{Map} \rangle$$

private map := λ k. 0

def init(sz: int) ≡

  🟥⟹ r0 ∈ *pending*

  skip

  r1 ∈ $\ast_{k \in [0,sz)}\, k \mapsto_{Map} 0$ ⟹ 🟩

def set(k: int, v: int) ≡

  🟥⟹ r2 ∈ $\exists w.\, k \mapsto_{Map} w$

  map := map[k ↤ v]

  r3 ∈ $k \mapsto_{Map} v$ ⟹ 🟩

Dual Non-Determinism

Pass Resources Implicitly

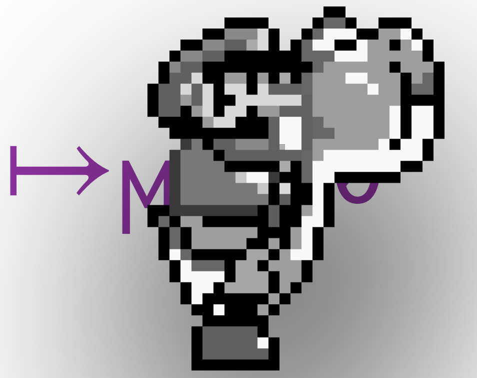
$\langle S \vdash A_{Clnt} \rangle$

def main() ≡

r0 ∈ *pending* ⟹🟩

init(2)

🟥⟹ r1 ∈ $0 \mapsto_{Map} 0 * 1 \mapsto_{M}$



r2 ∈ $\exists w.\, 0 \mapsto_{Map} w$ ⟹🟩

var fr ∈ $1 \mapsto_{Map} 0$

set(0, 42)

🟥⟹ r3 ∈ $0 \mapsto_{Map} 42$

$\langle S \vdash A_{Map} \rangle$

private map := λ k. 0

def init(sz: int) ≡

🟥⟹ r0 ∈ *pending*

skip

r1 ∈ $\ast_{k \in [0, sz)}\, k \mapsto_{Map} 0$ ⟹🟩

def set(k: int, v: int) ≡

🟥⟹ r2 ∈ $\exists w.\, k \mapsto_{Map} w$

map := map[k ↤ v]

r3 ∈ $k \mapsto_{Map} v$ ⟹🟩

Dual Non-Determinism

Pass Resources Implicitly

$$\langle S \vdash A_{Clnt} \rangle$$

def main() ≡

  r0 ∈ $\boxed{\textit{pending}}$ ⟹ 🟩

  init(2)

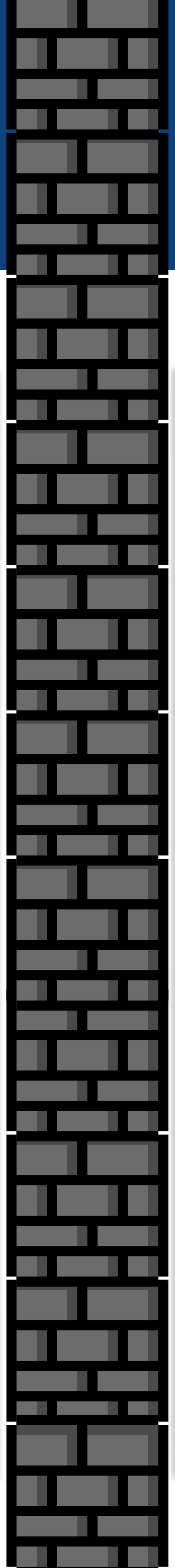
  🟥 ⟹ r1 ∈ $0 \mapsto_{Map} 0 * 1 \mapsto_{Map} 0$

  ∈ $\exists w.\ 0 \mapsto_{Map} w$ ⟹ 🟩

  var fr ∈ $1 \mapsto_{Map} 0$

  set(0, 42)

  🟥 ⟹ r3 ∈ $0 \mapsto_{Map} 42$



$$\langle S \vdash A_{Map} \rangle$$

private map := λ k. 0

def init(sz: int) ≡

  🟥 ⟹ r0 ∈ $\boxed{\textit{pending}}$

  skip

  r1 ∈ $\ast_{k \in [0,sz)}\ k \mapsto_{Map} 0$ ⟹ 🟩

def set(k: int, v: int) ≡

  🟥 ⟹ r2 ∈ $\exists w.\ k \mapsto_{Map} w$

  map := map[k ↤ v]

  r3 ∈ $k \mapsto_{Map} v$ ⟹ 🟩

Dual Non-Determinism

Pass Resources Implicitly

$\langle\, S \vdash A_{Clnt} \,\rangle$

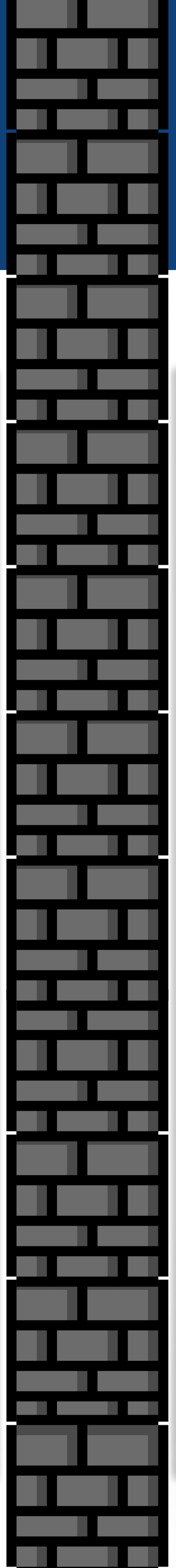
```
def main() ≡
  r0 ∈ pending ⟹
  init(2)
⟹ r1 ∈ 0 ↦Map 0 * 1 ↦Map 0


  r2 ∈ ∃w. 0 ↦Map w ⟹
  var fr ∈ 1 ↦Map 0
  set(0, 42)
⟹ r3 ∈ 0 ↦Map 42
```

$\langle\, S \vdash A_{Map} \,\rangle$

```
private map := λ k. 0
def init(sz: int) ≡
⟹ r0 ∈ pending
  skip
  r1 ∈ ∗k∈[0,sz) k ↦Map 0 ⟹

def set(k: int, v: int) ≡
⟹ r2 ∈ ∃w.k ↦Map w
  map := map[k ↤ v]

  r3 ∈ k ↦Map v ⟹
```

Dual Non-Determinism

Pass Resources Implicitly



$\langle S \vdash A_{Clnt} \rangle$

def main() ≡
  $r0 \in \boxed{\mathit{pending}} \Longrightarrow$
  init(2)
  $\Longrightarrow r1 \in 0 \mapsto_{Map} 0 * 1 \mapsto_{Map} 0$

  $r2 \in \exists w.\ 0 \mapsto_{Map} w \Longrightarrow$
  var fr $\in 1 \mapsto_{Map} 0$
  set(0, 42)
  $\Longrightarrow r3 \in 0 \mapsto_{Map} 42$

$\langle S \vdash A_{Map} \rangle$

private map := λ k. 0
def init(sz: int) ≡
  $\Longrightarrow r0 \in \boxed{\mathit{pending}}$
  skip
  $r1 \in \mathop{\ast}_{k \in [0, sz)} k \mapsto_{Map} 0 \Longrightarrow$

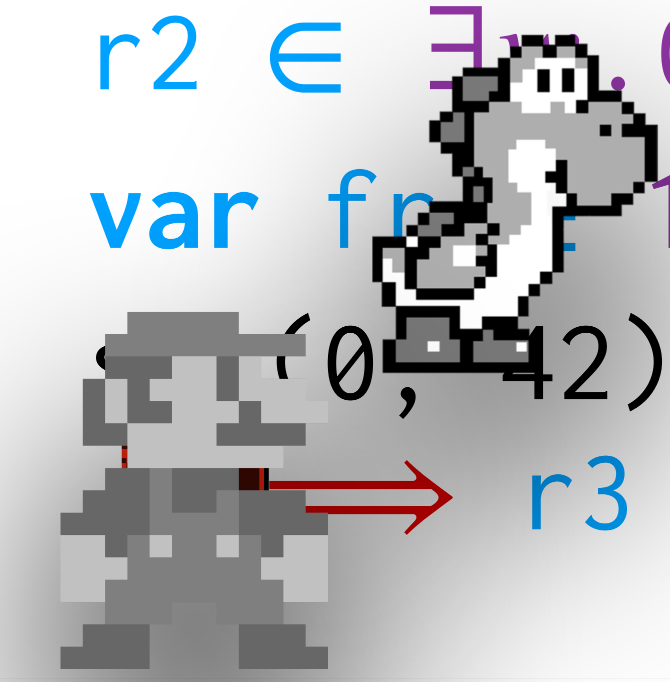
def set(k: int, v: int) ≡
  $\Longrightarrow r2 \in \exists w.k \mapsto_{Map} w$
  map := map[k ↤ v]
  $r \in k \mapsto_{Map} v \Longrightarrow$

Dual Non-Determinism

Pass Resources Implicitly

$\langle S \vdash A_{Clnt} \rangle$

```
def main() ≡
  r0 ∈ pending ⟹
  init(2)
⟹ r1 ∈ 0 ↦Map 0 * 1 ↦Map 0


  r2 ∈ ∃w. 0 ↦Map w ⟹
  var fr ∈ 1 ↦Map 0
  set(0, 42)
⟹ r3 ∈ 0 ↦Map 42
```



$\langle S \vdash A_{Map} \rangle$

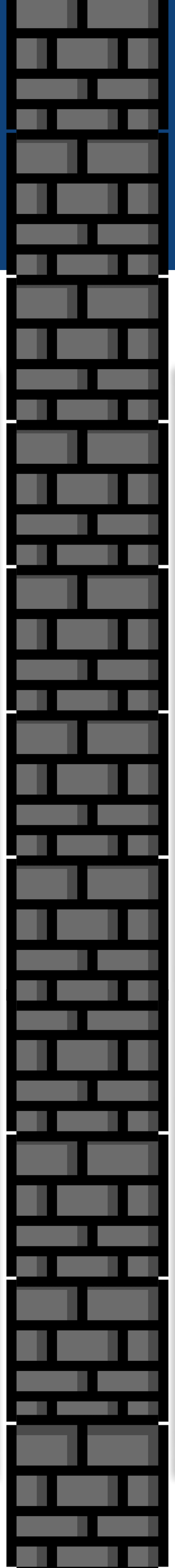
```
private map := λ k. 0
def init(sz: int) ≡
⟹ r0 ∈ pending
  skip
  r1 ∈ *k∈[0,sz) k ↦Map 0 ⟹

def set(k: int, v: int) ≡
⟹ r2 ∈ ∃w.k ↦Map w
  map := map[k ↤ v]

  r3 ∈ k ↦Map v ⟹
```

Dual Non-Determinism

Pass Resources Implicitly

$$\langle\, S \vdash A_{Clnt} \,\rangle$$

```
def main() ≡
   r0 ∈ pending ⟹
   init(2)
   ⟹ r1 ∈ 0 ↦Map 0 * 1 ↦Map 0


   r2 ∈ ∃w. 0 ↦Map w ⟹
   var fr ∈ 1 ↦Map 0
   set(0, 42)
   ⟹ r3 ∈ 0 ↦Map
```



$$\langle\, S \vdash A_{Map} \,\rangle$$

```
private map := λ k. 0
def init(sz: int) ≡
   ⟹ r0 ∈ pending
   skip
   r1 ∈ *k∈[0,sz) k ↦Map 0 ⟹

def set(k: int, v: int) ≡
   ⟹ r2 ∈ ∃w.k ↦Map w
   map := map[k ↤ v]

   r3 ∈ k ↦Map v ⟹
```

Dual Non-Determinism

Pass Resources Implicitly

$\langle S \vdash A_{Clnt} \rangle$

def main() ≡

  r0 ∈ *pending* ⇒ 🟩

  init(2)

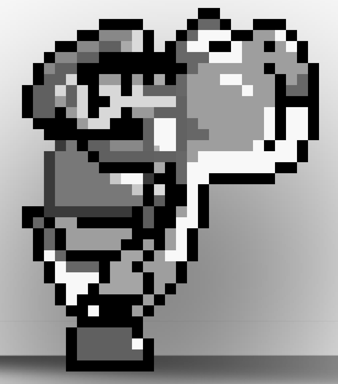
  🟥 ⇒ r1 ∈ $0 \mapsto_{Map} 0 * 1 \mapsto_{Map} 0$

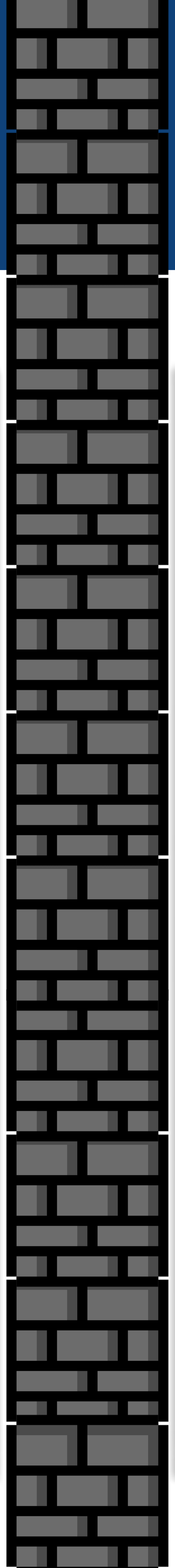
  r2 ∈ $\exists w.\, 0 \mapsto_{Map} w$ ⇒ 🟩

  var fr ∈ $1 \mapsto_{Map} 0$

  set(0, 42)

  🟥 ⇒ r3 ∈ $0 \mapsto_{Map} 42$





$\langle S \vdash A_{Map} \rangle$

private map := λ k. 0

def init(sz: int) ≡

  🟥 ⇒ r0 ∈ *pending*

  skip

  r1 ∈ $\ast_{k \in [0,sz)}\, k \mapsto_{Map} 0$ ⇒ 🟩

def set(k: int, v: int) ≡

  🟥 ⇒ r2 ∈ $\exists w.\, k \mapsto_{Map} w$

  map := map[k ↤ v]

  r3 ∈ $k \mapsto_{Map} v$ ⇒ 🟩

Key Idea III: Wrapper Elimination

Wrapper Elimination

$\langle S \vdash A_{Clnt} \rangle$

```
def main() ≡
  r0 ∈ pending ⟹
  init(2)
⟹ r1 ∈ 0 ↦Map 0 * 1 ↦Map 0


  r2 ∈ ∃w.0 ↦Map w ⟹
  var fr ∈ 1 ↦Map 0
  set(0, 42)
⟹ r3 ∈ 0 ↦Map 42
```

$\langle S \vdash A_{Map} \rangle$

```
private map := λ k. 0
def init(sz: int) ≡
  ⟹ r0 ∈ pending
  skip
  r1 ∈ *k∈[0,sz) k ↦Map 0 ⟹

def set(k: int, v: int) ≡
  ⟹ r2 ∈ ∃w.k ↦Map w
  map := map[k ↤ v]

  r3 ∈ k ↦Map v ⟹
```

Wrapper Elimination

$$\langle S \vdash A_{Clnt} \rangle \quad + \quad \langle S \vdash A_{Map} \rangle$$

```
def main() ≡
   r0 ∈ pending ⟹🟩
   init(2)
 🟥⟹ r1 ∈ 0 ↦Map 0 * 1 ↦Map 0


   r2 ∈ ∃w.0 ↦Map w ⟹🟩
   var fr ∈ 1 ↦Map 0
   set(0, 42)
 🟥⟹ r3 ∈ 0 ↦Map 42
```

+

```
private map := λ k. 0
def init(sz: int) ≡
  🟥⟹ r0 ∈ pending
  skip
  r1 ∈ *k∈[0,sz) k ↦Map 0 ⟹🟩

def set(k: int, v: int) ≡
  🟥⟹ r2 ∈ ∃w.k ↦Map w
  map := map[k ↤ v]

  r3 ∈ k ↦Map v ⟹🟩
```

Wrapper Elimination

$\langle S \vdash A_{Clnt} \rangle$ + $\langle S \vdash A_{Map} \rangle$

def main() ≡

$r0 \in \boxed{pending} \Longrightarrow$

init(2)

$\Longrightarrow r1 \in 0 \mapsto_{Map} 0 * 1 \mapsto_{Map} 0$

$r2 \in \exists w.0 \mapsto_{Map} w \Longrightarrow$

var $fr \in 1 \mapsto_{Map} 0$

set(0, 42)

$\Longrightarrow r3 \in 0 \mapsto_{Map} 42$

+

private map := λ k. 0

def init(sz: int) ≡

$\Longrightarrow r0 \in \boxed{pending}$

skip

$r1 \in \ast_{k \in [0,sz)} k \mapsto_{Map} 0 \Longrightarrow$

def set(k: int, v: int) ≡

$\Longrightarrow r2 \in \exists w.k \mapsto_{Map} w$

map := map[k ↤ v]

$r3 \in k \mapsto_{Map} v \Longrightarrow$

Wrapper Elimination

$\langle \boxed{S} \vdash A_{Clnt} \rangle$ + $\langle \boxed{S} \vdash A_{Map} \rangle$

```
def main() ≡
  r0 ∈ pending ⟹
  init(2)
    ⟹ r1 ∈ $0 \mapsto_{Map} 0 * 1 \mapsto_{Map} 0$


  r2 ∈ $\exists w. 0 \mapsto_{Map} w$ ⟹
  var fr ∈ $1 \mapsto_{Map} 0$
  set(0, 42)
    ⟹ r3 ∈ $0 \mapsto_{Map} 42$
```

+

```
private map := λ k. 0
def init(sz: int) ≡
      ⟹ r0 ∈ pending
  skip
  r1 ∈ $\circledast_{k \in [0,sz)} k \mapsto_{Map} 0$ ⟹

def set(k: int, v: int) ≡
      ⟹ r2 ∈ $\exists w. k \mapsto_{Map} w$
  map := map[k ↤ v]

  r3 ∈ $k \mapsto_{Map} v$ ⟹
```

Wrapper Elimination

$\langle S \vdash A_{Clnt} \rangle \quad + \quad \langle S \vdash A_{Map} \rangle$

```
def main() ≡
  r0 ∈ pending ⟹
  init(2)
     ⟹ r1 ∈ 0 ↦Map 0 * 1 ↦Map 0


  r2 ∈ ∃w.0 ↦Map w ⟹
  var fr ∈ 1 ↦Map 0
  set(0, 42)
     ⟹ r3 ∈ 0 ↦Map 42
```

+

```
private map := λ k. 0
def init(sz: int) ≡
      ⟹ r0 ∈ pending
  skip
  r1 ∈ ∗k∈[0,sz) k ↦Map 0 ⟹

def set(k: int, v: int) ≡
      ⟹ r2 ∈ ∃w.k ↦Map w
  map := map[k ↤ v]

  r3 ∈ k ↦Map v ⟹
```

Wrapper Elimination

A_{Clnt} + A_{Map}

```
def main() ≡

  init(2)


  set(0, 42)
```

+

```
private map := λ k. 0
def init(sz: int) ≡

  skip


def set(k: int, v: int) ≡

  map := map[k ↤ v]
```

Wrapper Elimination

Wrapper Elimination Theorem (WET)

$$\langle S \vdash A_1 \rangle + \langle S \vdash A_2 \rangle \sqsubseteq A_1 + A_2$$

Wrapper Elimination Theorem (WET)

$$\langle S \vdash A_1 \rangle + \langle S \vdash A_2 \rangle \sqsubseteq A_1 + A_2$$

Every module adheres to the same spec

Wrapper Elimination Theorem (WET)

$$\langle\, S \vdash A_1 \,\rangle + \langle\, S \vdash A_2 \,\rangle \sqsubseteq A_1 + A_2$$

Every module adheres to the same spec

(i.e., pipes are installed properly) $\Longrightarrow$

Wrapper Elimination Theorem (WET)

$$\langle\, S \vdash A_1 \,\rangle + \langle\, S \vdash A_2 \,\rangle \sqsubseteq A_1 + A_2$$

Every module adheres to the same spec

(i.e., pipes are installed properly) $\Longrightarrow$

Resources are matched up properly during the execution

Wrapper Elimination Theorem (WET)

$$\langle\, S \vdash A_1 \,\rangle + \langle\, S \vdash A_2 \,\rangle \sqsubseteq A_1 + A_2$$

Every module adheres to the same spec

(i.e., pipes are installed properly) ⟹

Resources are matched up properly during the execution

(i.e., Mario does not die from a pipe accident) ⟹

Wrapper Elimination Theorem (WET)

$$\langle S \vdash A_1 \rangle + \langle S \vdash A_2 \rangle \sqsubseteq A_1 + A_2$$

Every module adheres to the same spec

(i.e., pipes are installed properly) $\Longrightarrow$

Resources are matched up properly during the execution

(i.e., Mario does not die from a pipe accident) $\Longrightarrow$

Wrapper Elimination Theorem!

Wrapper Elimination Theorem (WET)

$$\langle S \vdash A_1 \rangle + \langle S \vdash A_2 \rangle \sqsubseteq A_1 + A_2$$

Every module adheres to the same spec

(i.e., pipes are installed properly) $\Longrightarrow$

Resources are matched up properly during the execution

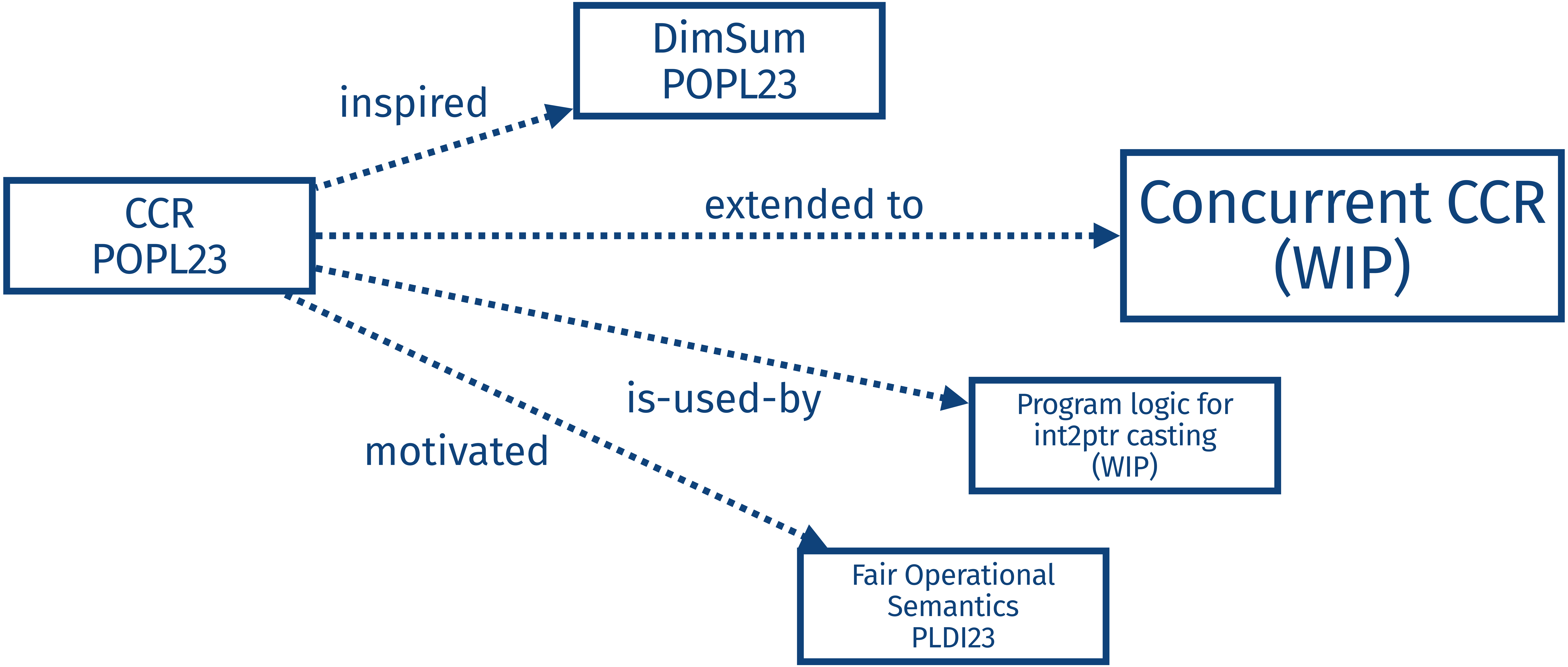
(i.e., Mario does not die from a pipe accident) $\Longrightarrow$

Wrapper Elimination Theorem!

(i.e., can get rid of these pipes!)

Past, Present, and Future

Actively developed, get involved!



Wrap Up of CCR

CCR	marries	refinement & separation logic
Wrapper	operationalizes	separation logic conditions
Dual non-determinism	allows	implicit resource passing

CCR 2.0: Vertical Frame Rule

Youngju Song, Minki Cho, and ?

Limitations of CCR

- CCR developed a model that unifies refinement and separation logic

Limitations of CCR

- CCR developed a model that unifies refinement and separation logic
- But its proof was in a low-level and tedious!

Limitations of CCR

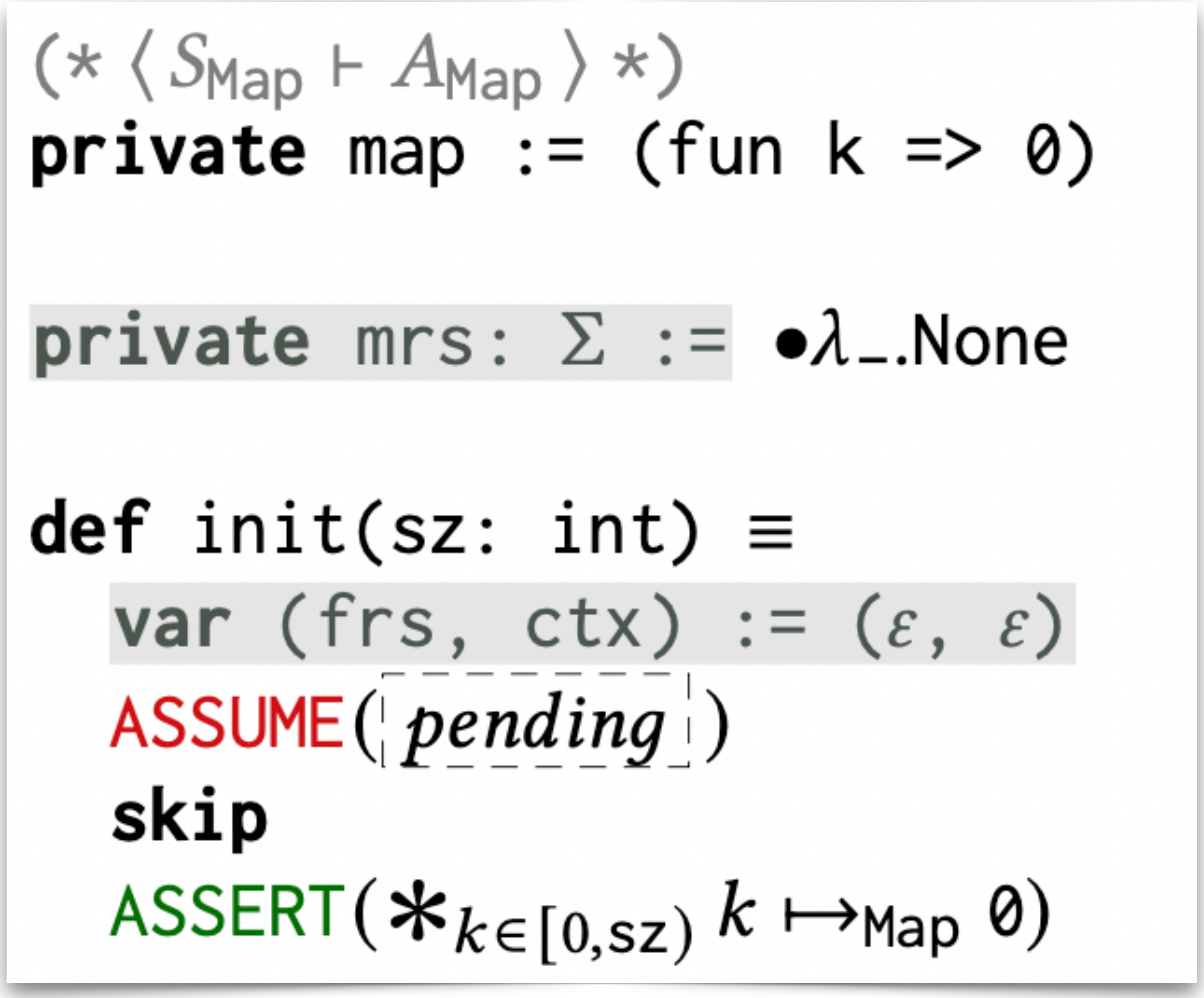
- CCR developed a model that unifies refinement and separation logic
- But its proof was in a low-level and tedious!
  - (i) Resources appeared explicit in the proof

Limitations of CCR

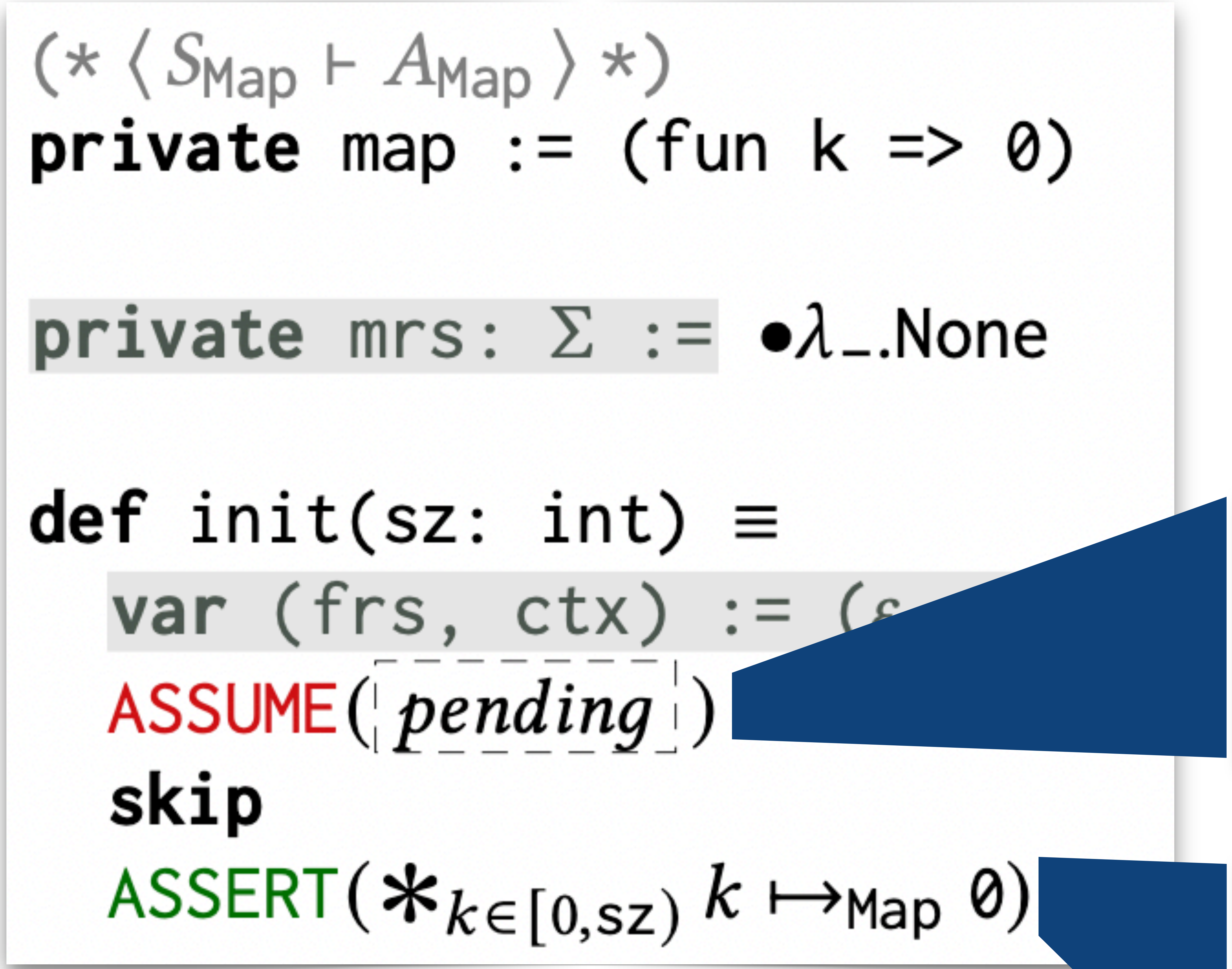
- CCR developed a model that unifies refinement and separation logic
- But its proof was in a low-level and tedious!
  - (i) Resources appeared explicit in the proof
  - (ii) ASSUME/ASSERTs were manually executed

Limitations of CCR

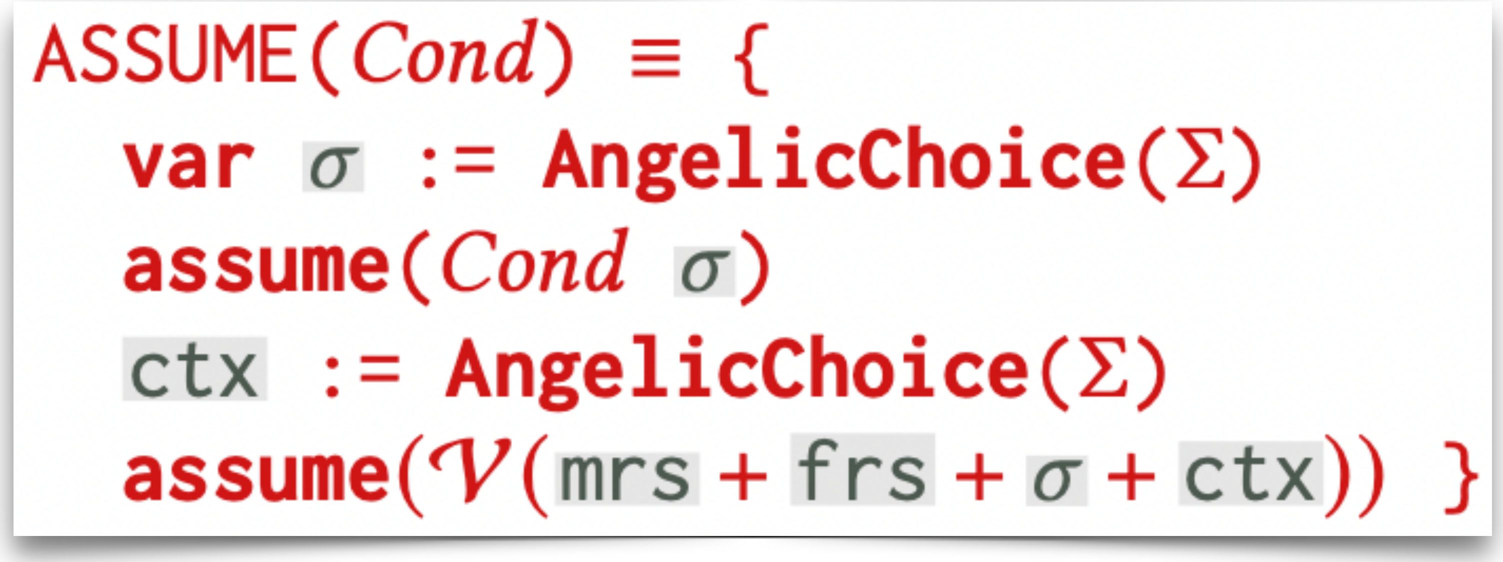
- CCR developed a model that unifies refinement and separation logic
- But its proof was in a low-level and tedious!
  - (i) Resources appeared explicit in the proof
  - (ii) ASSUME/ASSERTs were manually executed
- (iii) Limited compositionality(reusability) of refinement proofs



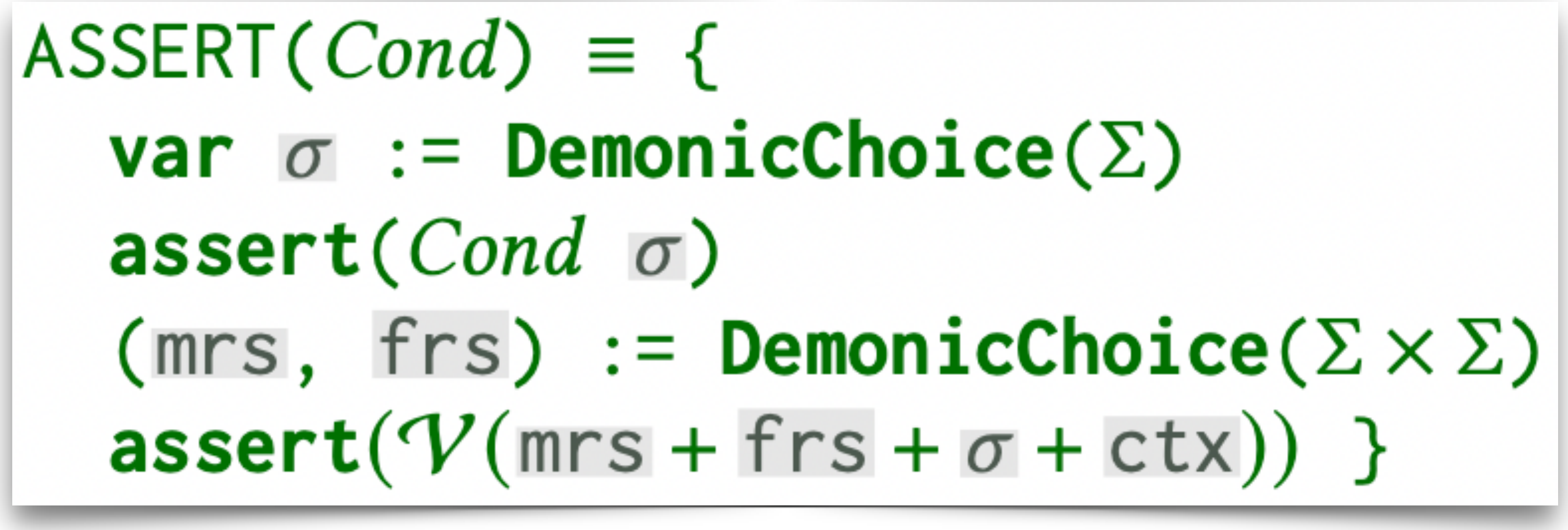
```
(* $\langle S_{\mathsf{Map}} \vdash A_{\mathsf{Map}} \rangle$ *)
private map := (fun k => 0)

private mrs: Σ := $\bullet\lambda\_.\mathsf{None}$

def init(sz: int) ≡
   var (frs, ctx) := ($\varepsilon$, $\varepsilon$)
   ASSUME($\boxed{pending}$)
   skip
   ASSERT($\ast_{k\in[0,sz)}\ k \mapsto_{\mathsf{Map}} 0$)
```



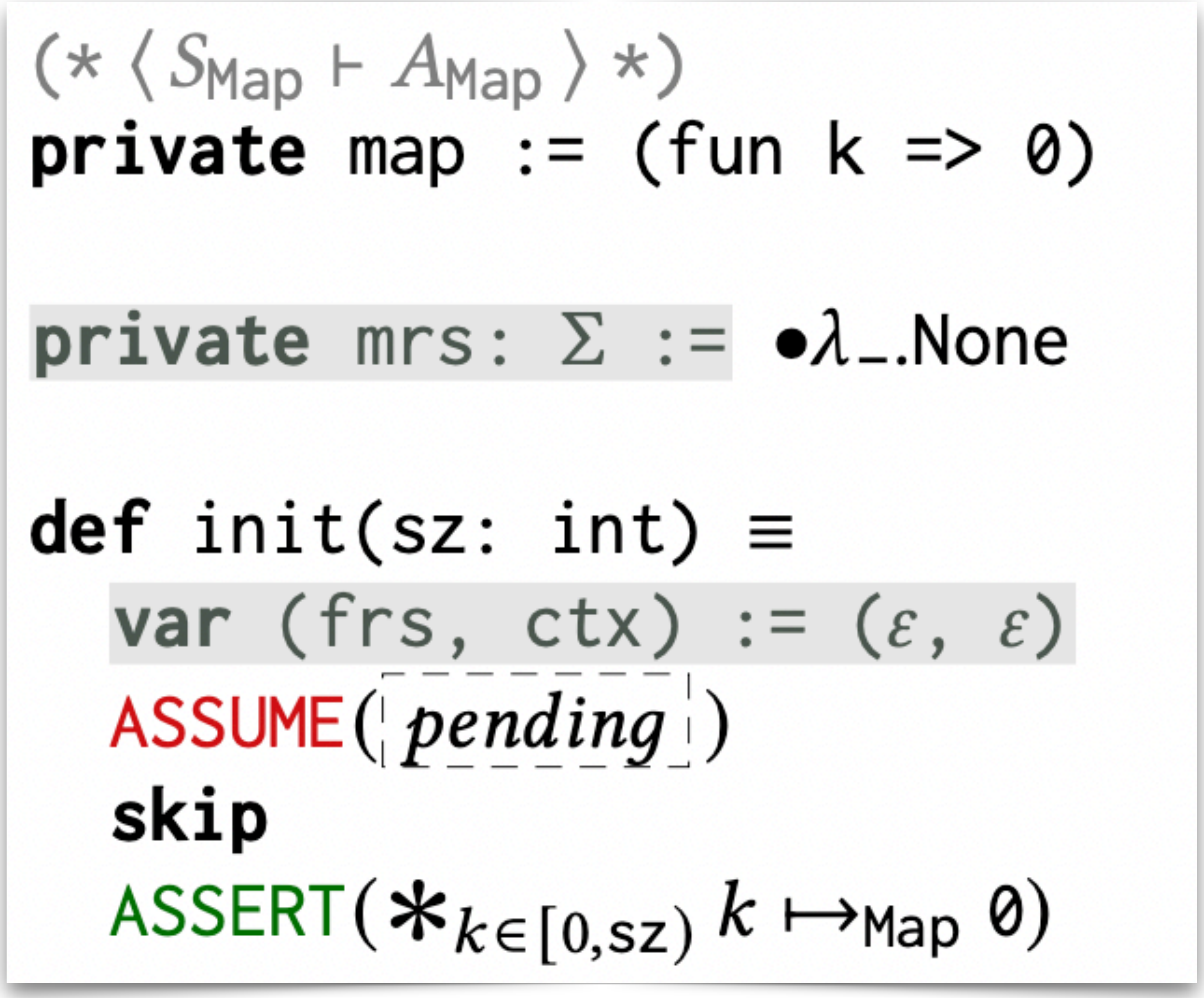
```
(* ⟨ $S_{Map}$ ⊢ $A_{Map}$ ⟩ *)
private map := (fun k => 0)

private mrs: Σ := •λ_.None

def init(sz: int) ≡
   var (frs, ctx) := (ς
   ASSUME( pending )
   skip
   ASSERT($\ast_{k \in [0,sz)}$ k $\mapsto_{Map}$ 0)
```



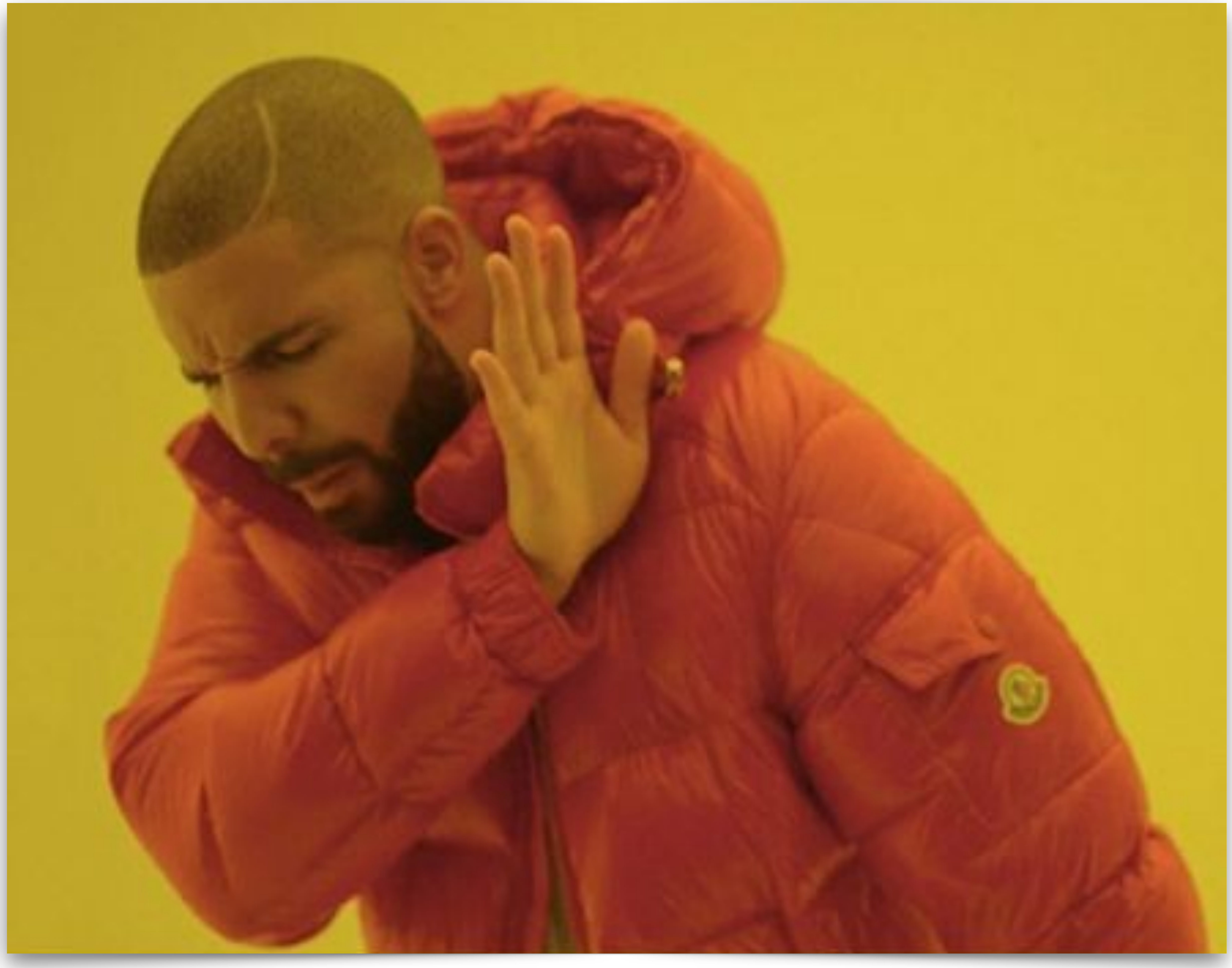
```
ASSUME(Cond) ≡ {
   var σ := AngelicChoice(Σ)
   assume(Cond σ)
   ctx := AngelicChoice(Σ)
   assume($\mathcal{V}$(mrs + frs + σ + ctx)) }
```



```
ASSERT(Cond) ≡ {
   var σ := DemonicChoice(Σ)
   assert(Cond σ)
   (mrs, frs) := DemonicChoice(Σ × Σ)
   assert($\mathcal{V}$(mrs + frs + σ + ctx)) }
```



```
(* ⟨ S_Map ⊢ A_Map ⟩ *)
private map := (fun k => 0)

private mrs: Σ := •λ_.None

def init(sz: int) ≡
   var (frs, ctx) := (ε, ε)
   ASSUME( pending )
   skip
   ASSERT(∗_{k∈[0,sz)} k ↦_Map 0)
```



10 lines of wrapper for
1 line of actual code?!

4 resources floating around?!

CCR 2.0

- But its proof was in a low-level and tedious!
  - (i) Resources appeared explicitly in the proof
  - (ii) ASSUME/ASSERTs were manually executed
- (iii) Limited compositionality(reusability) of refinement proofs

CCR 2.0

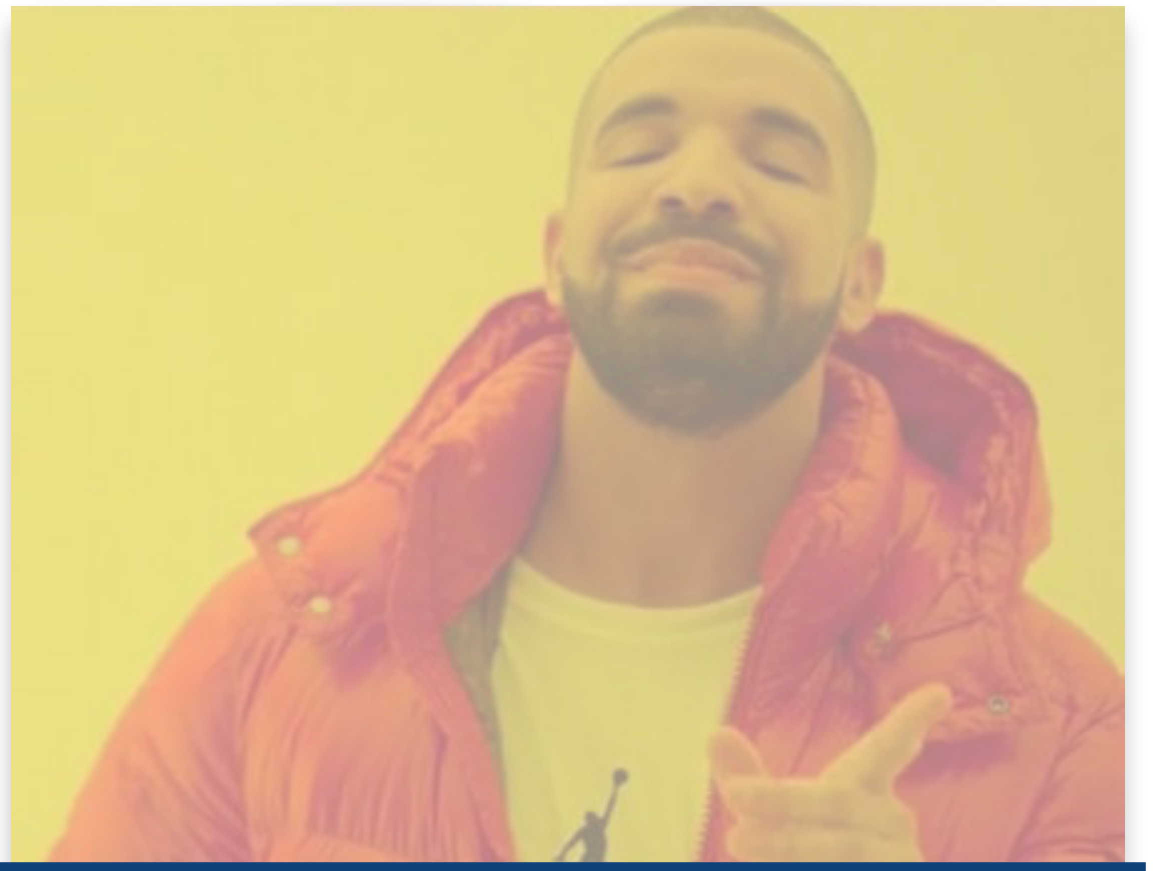
- But its proof was in a low-level and tedious!
  - (i) Resources appeared explicitly in the proof
    - Hide them behind **Iris Proof Mode** Iris
  - (ii) ASSUME/ASSERTs were manually executed
- (iii) Limited compositionality(reusability) of refinement proofs

CCR 2.0

- But its proof was in a low-level and tedious!
  - (i) Resources appeared explicitly in the proof
    - Hide them behind **Iris Proof Mode**
  - (ii) ASSUME/ASSERTs were manually executed
    - Provide an abstract interface like that of **SimulIris**
- (iii) Limited compositionality(reusability) of refinement proofs

CCR 2.0

- But its proof was in a low-level and tedious!
  - (i) Resources appeared explicitly in the proof
    - Hide them behind **Iris Proof Mode**
  - (ii) ASSUME/ASSERTs were manually executed
    - Provide an abstract interface like that of **SimulIris**
- (iii) Limited compositionality(reusability) of refinement proofs
  - **Vertical Frame Rule**



Key Ingredient:
Logical rules for executing ASSUME/ASSERT

Recap: rules for assume/assert

(ASMR)

$$\frac{P \implies \mathbb{T} \precsim \mathbb{S}}{T \precsim \texttt{assume}(P); \mathbb{S}}$$

(ASTR)

$$\frac{P \qquad \mathbb{T} \precsim \mathbb{S}}{\mathbb{T} \precsim \texttt{assert}(P); \mathbb{S}}$$

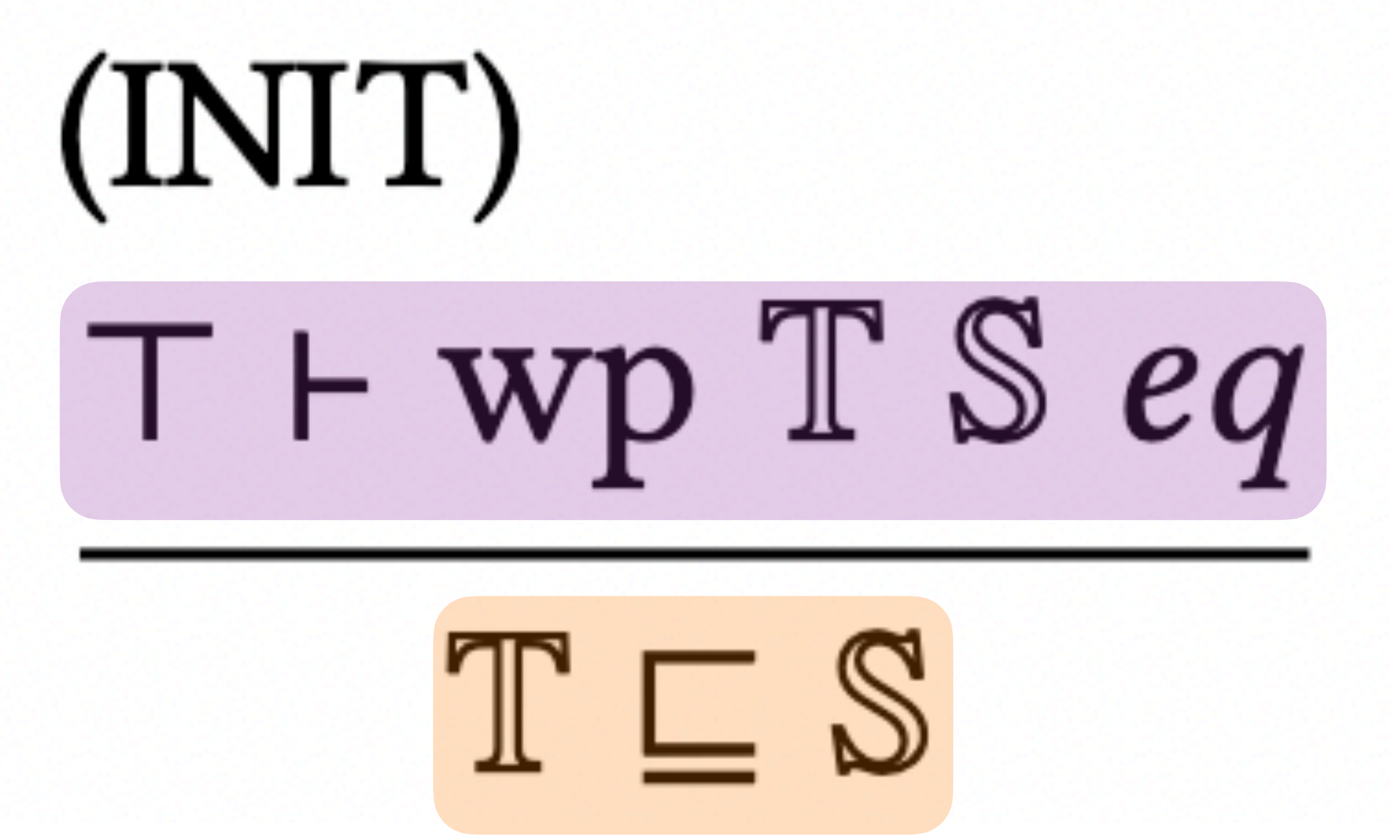
(ASML)

$$\frac{P \qquad \mathbb{T} \precsim \mathbb{S}}{\texttt{assume}(P); \mathbb{T} \precsim \mathbb{S}}$$

(ASTL)

$$\frac{P \implies \mathbb{T} \precsim \mathbb{S}}{\texttt{assert}(P); \mathbb{T} \precsim \mathbb{S}}$$

Now: rules for ASSUME/ASSERT

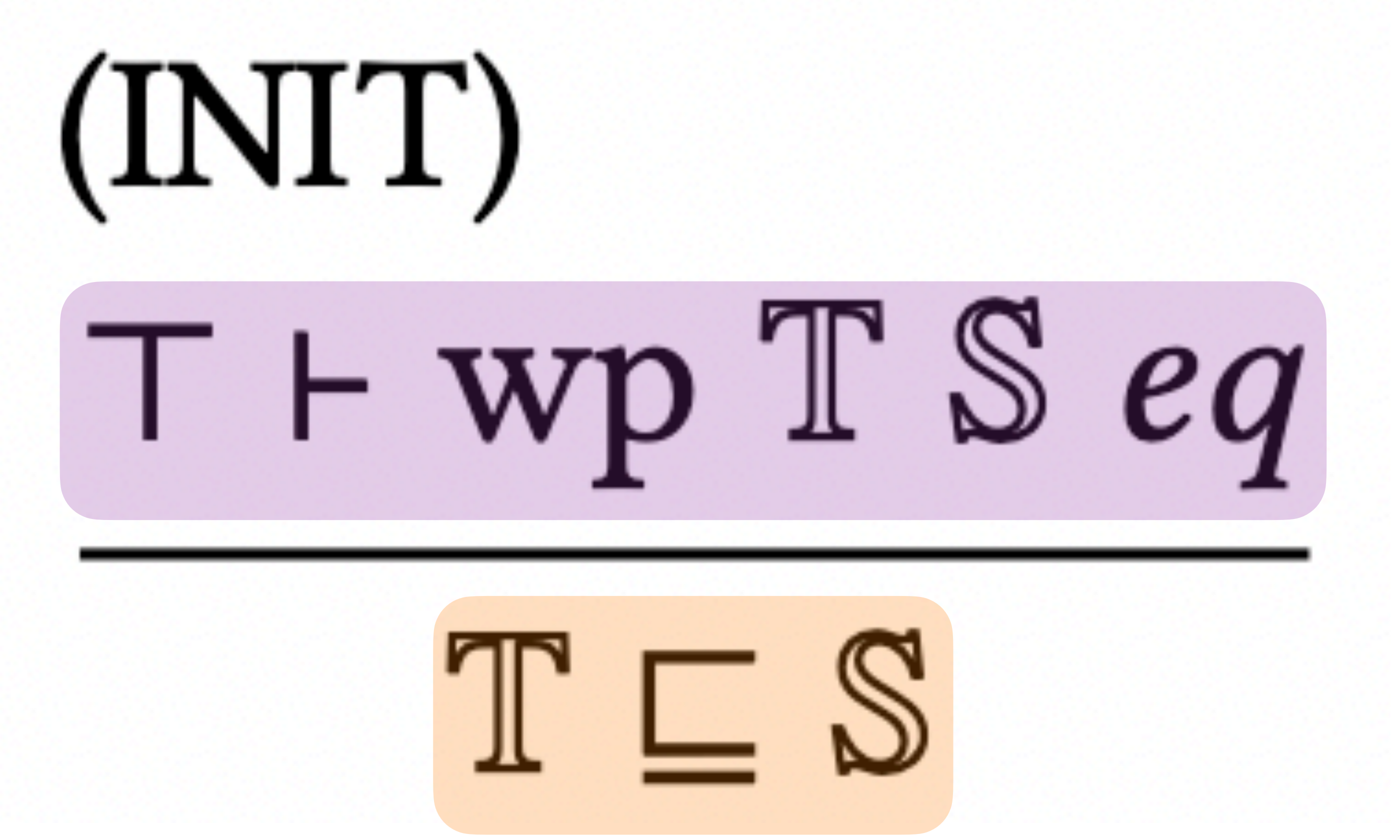


(INIT)

$$\frac{\top \vdash \mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ eq}{\mathbb{T} \sqsubseteq \mathbb{S}}$$

- First,
  turn the refinement goal into a separation logic predicate

Now: rules for ASSUME/ASSERT

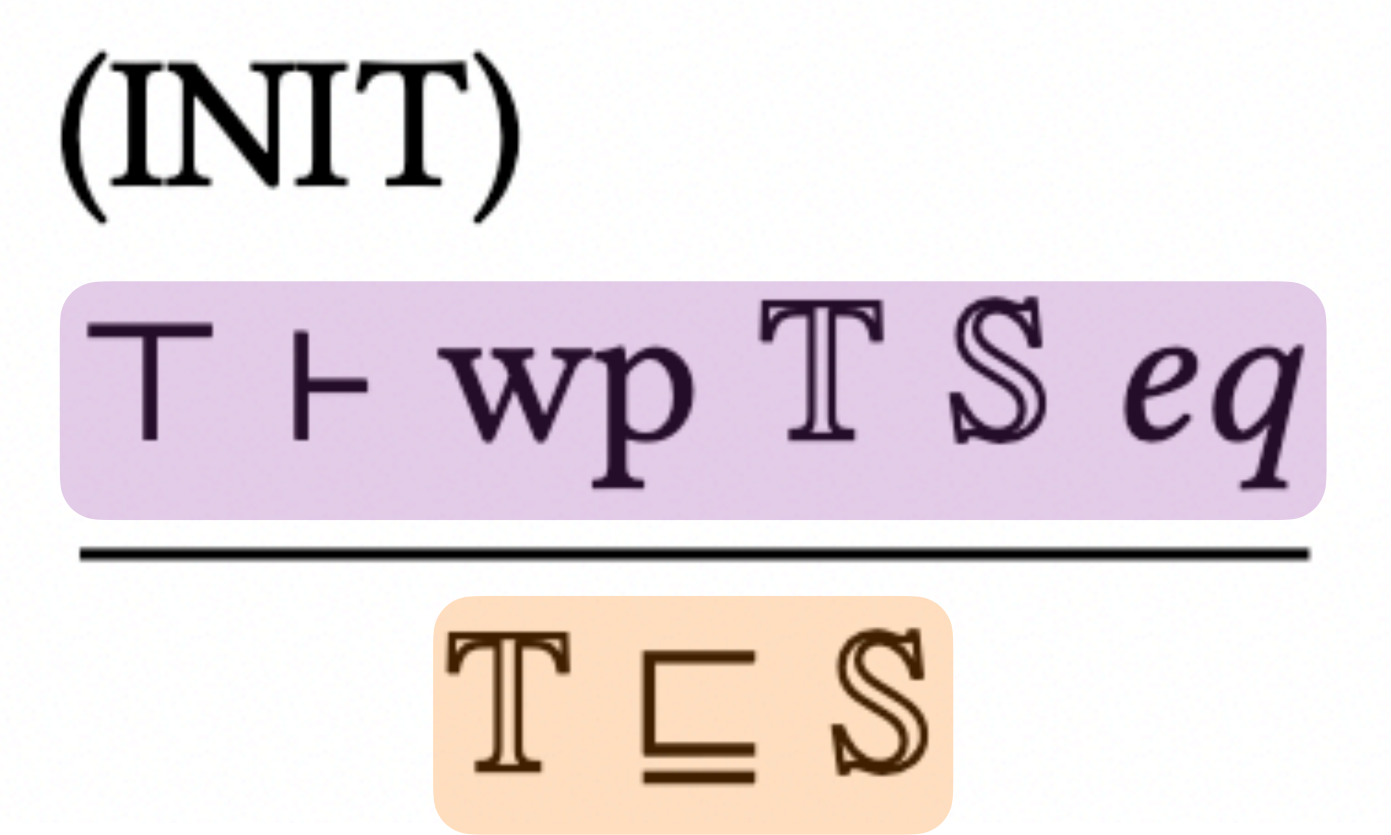


(INIT)

$$\frac{\top \vdash \mathbf{wp}\ \mathbb{T}\ \mathbb{S}\ \mathit{eq}}{\mathbb{T} \sqsubseteq \mathbb{S}}$$

- First,
  turn the refinement goal into a separation logic predicate
- “wp T S Φ” is a simulation WP (following SimulIris)

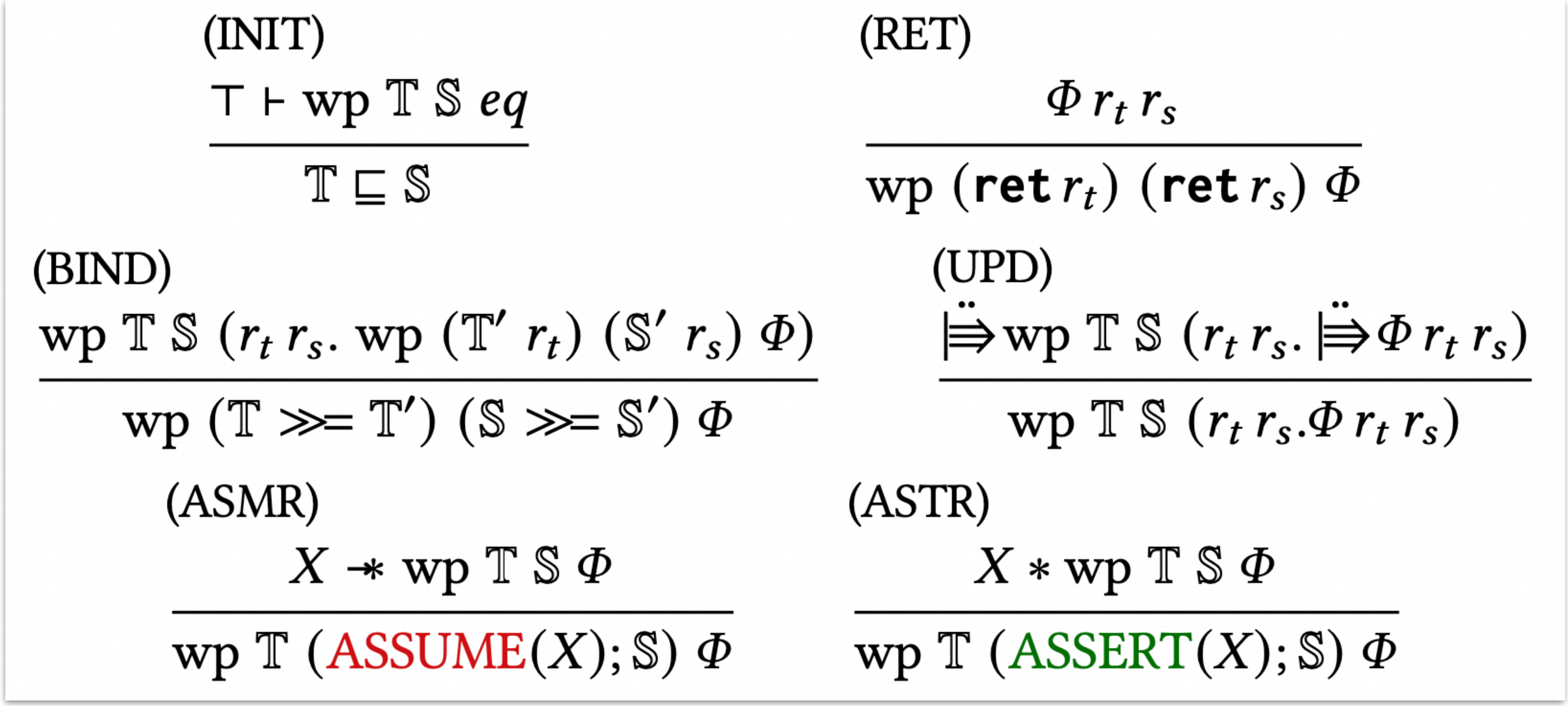
Now: rules for ASSUME/ASSERT



(INIT)

$$\frac{\top \vdash \mathbf{wp}\ \mathbb{T}\ \mathbb{S}\ eq}{\mathbb{T} \sqsubseteq \mathbb{S}}$$

- First,
  turn the refinement goal into a separation logic predicate
- “wp T S Φ” is a simulation WP (following SimulIris)
  - meaning the WP to simulate T against S and end with Φ



(INIT)

$$\frac{\top \vdash \mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ eq}{\mathbb{T} \sqsubseteq \mathbb{S}}$$

(RET)

$$\frac{\Phi\ r_t\ r_s}{\mathrm{wp}\ (\mathbf{ret}\ r_t)\ (\mathbf{ret}\ r_s)\ \Phi}$$

(BIND)

$$\frac{\mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ (r_t\ r_s.\ \mathrm{wp}\ (\mathbb{T}'\ r_t)\ (\mathbb{S}'\ r_s)\ \Phi)}{\mathrm{wp}\ (\mathbb{T} \gg\!= \mathbb{T}')\ (\mathbb{S} \gg\!= \mathbb{S}')\ \Phi}$$

(UPD)

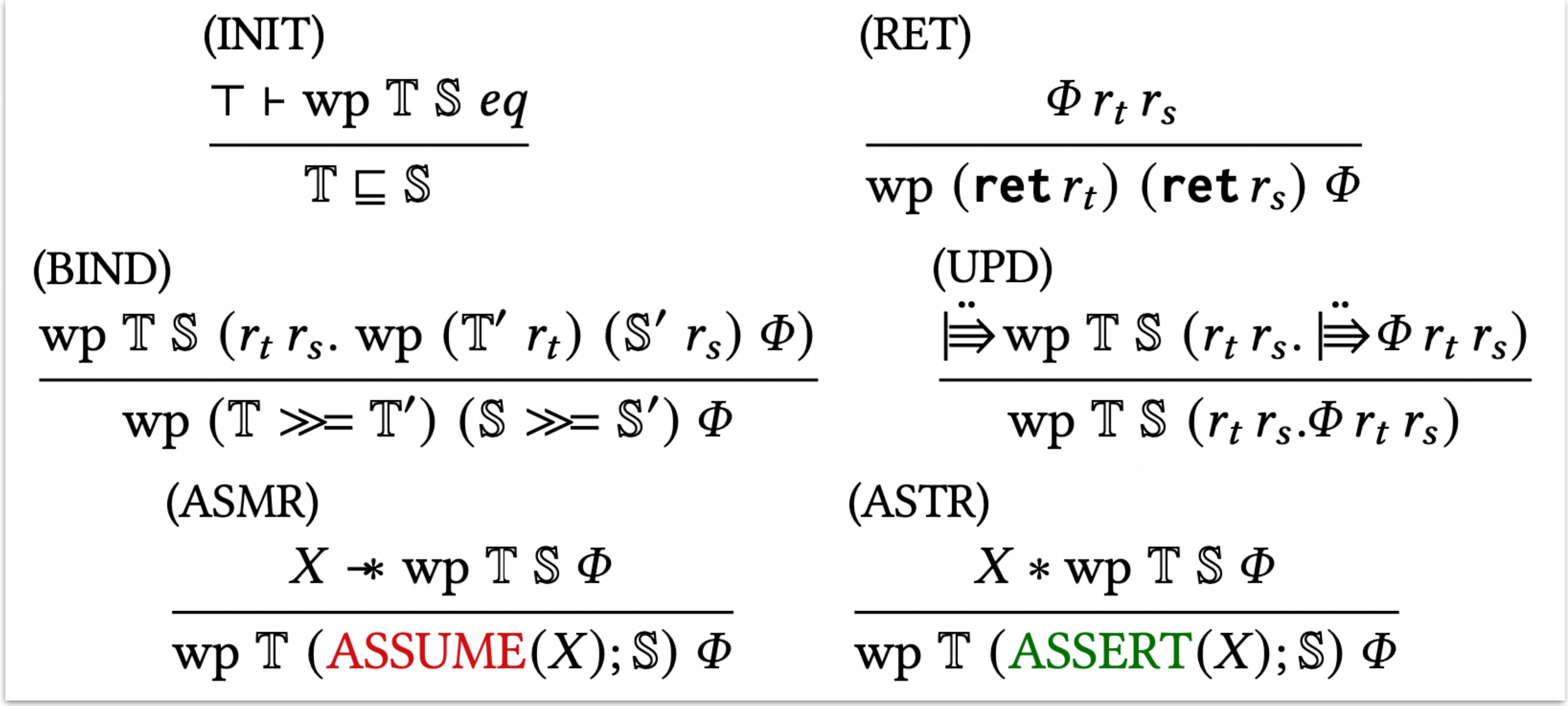
$$\frac{|\!\ddot{\Rrightarrow} \mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ (r_t\ r_s.\ |\!\ddot{\Rrightarrow} \Phi\ r_t\ r_s)}{\mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ (r_t\ r_s.\Phi\ r_t\ r_s)}$$

(ASMR)

$$\frac{X \twoheadrightarrow \mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ \Phi}{\mathrm{wp}\ \mathbb{T}\ (\mathrm{ASSUME}(X); \mathbb{S})\ \Phi}$$

(ASTR)

$$\frac{X * \mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ \Phi}{\mathrm{wp}\ \mathbb{T}\ (\mathrm{ASSERT}(X); \mathbb{S})\ \Phi}$$



(INIT)
$$\frac{\top \vdash \mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ eq}{\mathbb{T} \sqsubseteq \mathbb{S}}$$

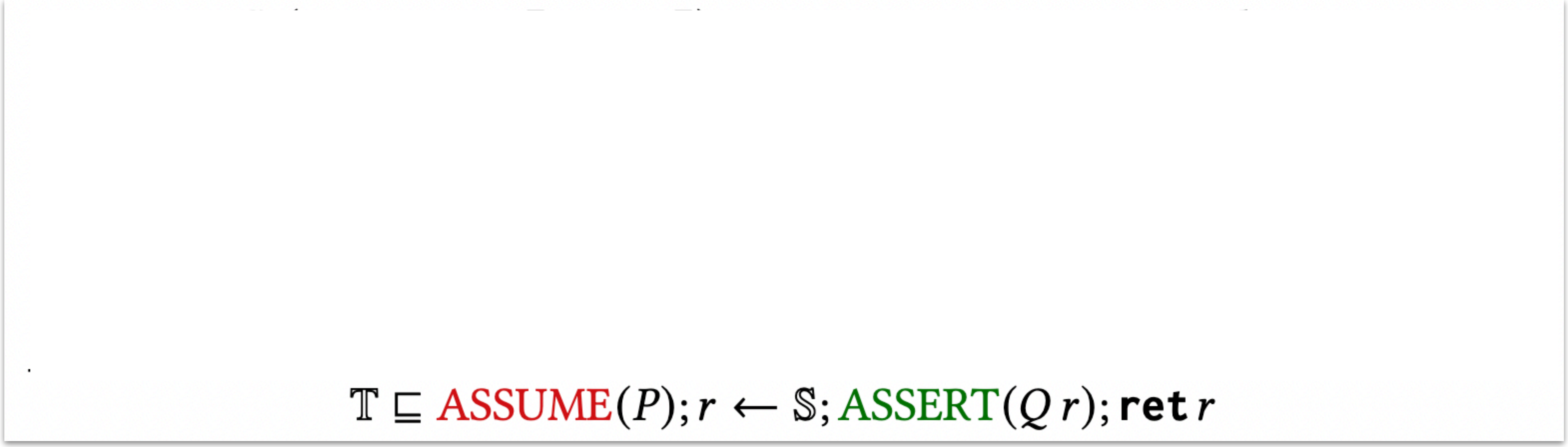
(RET)
$$\frac{\Phi\ r_t\ r_s}{\mathrm{wp}\ (\mathbf{ret}\ r_t)\ (\mathbf{ret}\ r_s)\ \Phi}$$

(BIND)
$$\frac{\mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ (r_t\ r_s.\ \mathrm{wp}\ (\mathbb{T}'\ r_t)\ (\mathbb{S}'\ r_s)\ \Phi)}{\mathrm{wp}\ (\mathbb{T} \gg\!= \mathbb{T}')\ (\mathbb{S} \gg\!= \mathbb{S}')\ \Phi}$$

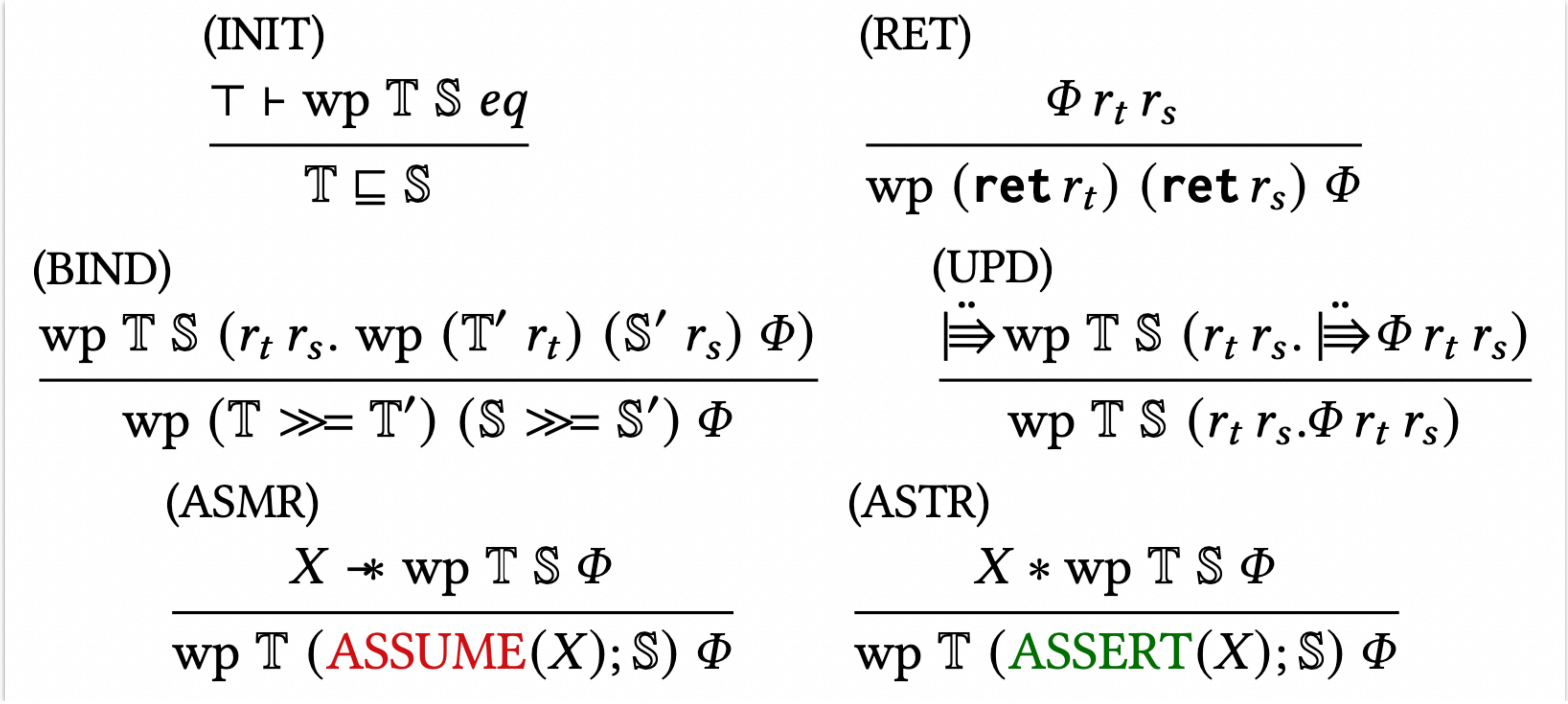
(UPD)
$$\frac{\overset{\cdot\cdot}{\Rrightarrow} \mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ (r_t\ r_s.\ \overset{\cdot\cdot}{\Rrightarrow} \Phi\ r_t\ r_s)}{\mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ (r_t\ r_s. \Phi\ r_t\ r_s)}$$

(ASMR)
$$\frac{X \mathrel{-\!\!*} \mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ \Phi}{\mathrm{wp}\ \mathbb{T}\ (\mathrm{ASSUME}(X); \mathbb{S})\ \Phi}$$

(ASTR)
$$\frac{X * \mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ \Phi}{\mathrm{wp}\ \mathbb{T}\ (\mathrm{ASSERT}(X); \mathbb{S})\ \Phi}$$



$$\mathbb{T} \sqsubseteq \mathrm{ASSUME}(P); r \leftarrow \mathbb{S}; \mathrm{ASSERT}(Q\ r); \mathbf{ret}\ r$$



(INIT)
$$\frac{\top \vdash \mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ eq}{\mathbb{T} \sqsubseteq \mathbb{S}}$$

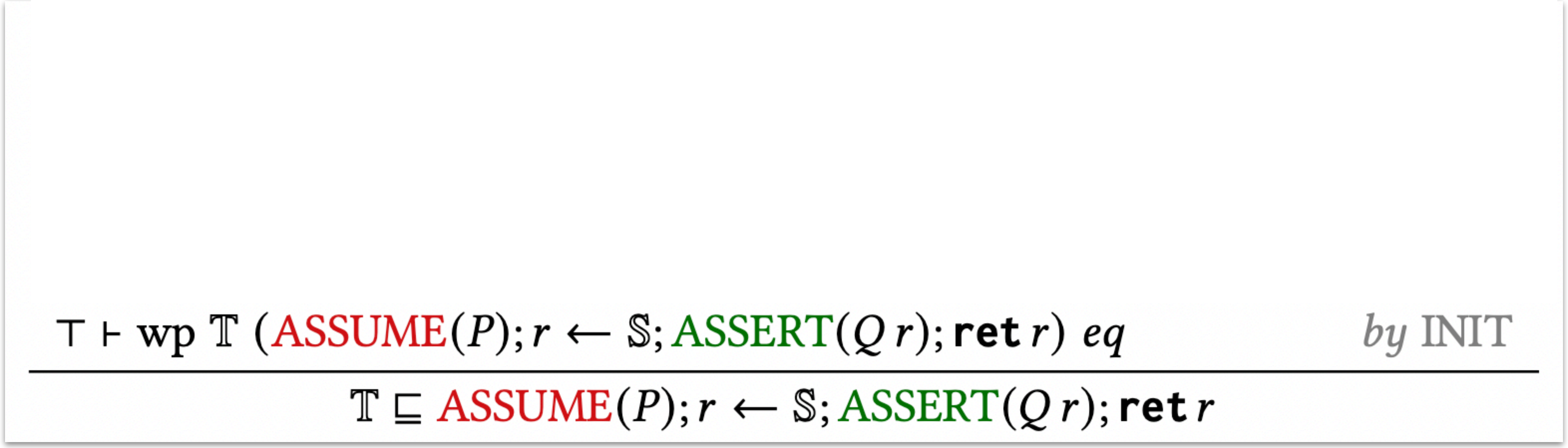
(RET)
$$\frac{\Phi\ r_t\ r_s}{\mathrm{wp}\ (\mathbf{ret}\ r_t)\ (\mathbf{ret}\ r_s)\ \Phi}$$

(BIND)
$$\frac{\mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ (r_t\ r_s.\ \mathrm{wp}\ (\mathbb{T}'\ r_t)\ (\mathbb{S}'\ r_s)\ \Phi)}{\mathrm{wp}\ (\mathbb{T} \gg= \mathbb{T}')\ (\mathbb{S} \gg= \mathbb{S}')\ \Phi}$$

(UPD)
$$\frac{|\ddot{\Rrightarrow}\,\mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ (r_t\ r_s.\ |\ddot{\Rrightarrow}\,\Phi\ r_t\ r_s)}{\mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ (r_t\ r_s.\Phi\ r_t\ r_s)}$$

(ASMR)
$$\frac{X \twoheadrightarrow \mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ \Phi}{\mathrm{wp}\ \mathbb{T}\ (\mathrm{ASSUME}(X); \mathbb{S})\ \Phi}$$

(ASTR)
$$\frac{X * \mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ \Phi}{\mathrm{wp}\ \mathbb{T}\ (\mathrm{ASSERT}(X); \mathbb{S})\ \Phi}$$



$$\frac{\top \vdash \mathrm{wp}\ \mathbb{T}\ (\mathrm{ASSUME}(P); r \leftarrow \mathbb{S}; \mathrm{ASSERT}(Q\ r); \mathbf{ret}\ r)\ eq \qquad \textit{by}\ \mathrm{INIT}}{\mathbb{T} \sqsubseteq \mathrm{ASSUME}(P); r \leftarrow \mathbb{S}; \mathrm{ASSERT}(Q\ r); \mathbf{ret}\ r}$$

(INIT)

$$\frac{\top \vdash \mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ eq}{\mathbb{T} \sqsubseteq \mathbb{S}}$$

(RET)

$$\frac{\Phi\ r_t\ r_s}{\mathrm{wp}\ (\mathbf{ret}\ r_t)\ (\mathbf{ret}\ r_s)\ \Phi}$$

(BIND)

$$\frac{\mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ (r_t\ r_s.\ \mathrm{wp}\ (\mathbb{T}'\ r_t)\ (\mathbb{S}'\ r_s)\ \Phi)}{\mathrm{wp}\ (\mathbb{T} \gg\!= \mathbb{T}')\ (\mathbb{S} \gg\!= \mathbb{S}')\ \Phi}$$

(UPD)

$$\frac{\mathrel{|\!\!\overset{\cdot\cdot}{\Rrightarrow}} \mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ (r_t\ r_s.\ \mathrel{|\!\!\overset{\cdot\cdot}{\Rrightarrow}} \Phi\ r_t\ r_s)}{\mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ (r_t\ r_s.\Phi\ r_t\ r_s)}$$

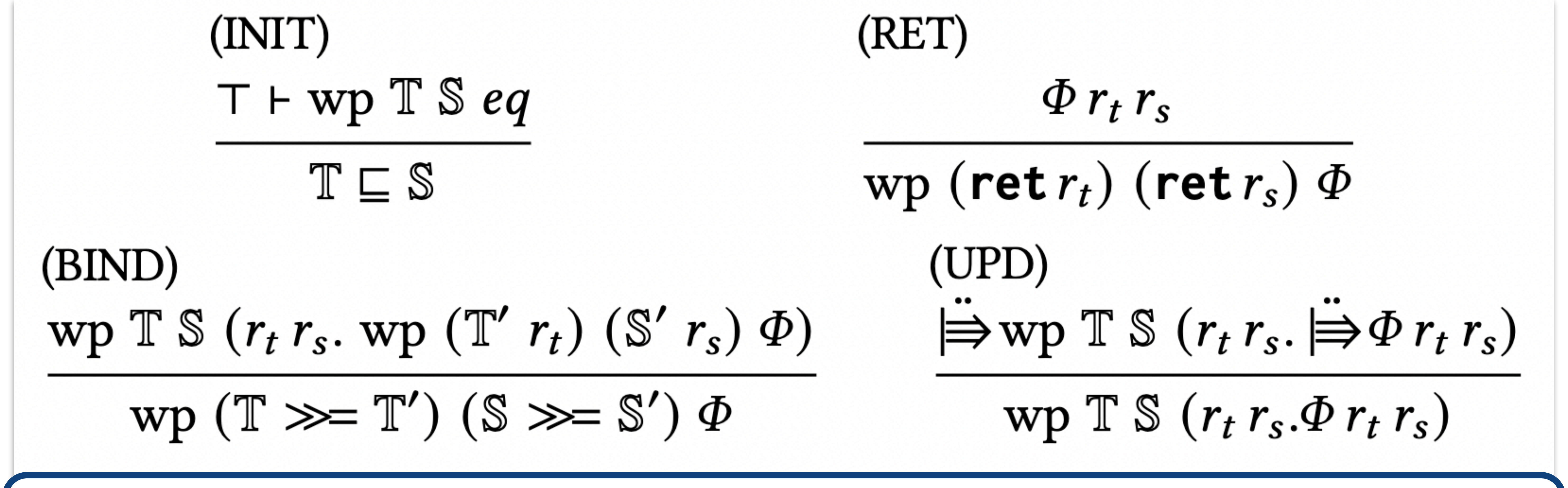
(ASMR)

$$\frac{X \mathrel{-\!\!*} \mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ \Phi}{\mathrm{wp}\ \mathbb{T}\ (\mathrm{ASSUME}(X); \mathbb{S})\ \Phi}$$

(ASTR)

$$\frac{X * \mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ \Phi}{\mathrm{wp}\ \mathbb{T}\ (\mathrm{ASSERT}(X); \mathbb{S})\ \Phi}$$

$$\begin{array}{ll}
P \vdash \mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ (r_t\ r_s.\ Q\ r_s * \ulcorner r_t = r_s \urcorner) & \textit{by}\ \text{UPD \& RET} \\ \hline
P \vdash \mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ (r_t\ r_s.\ Q\ r_s * \mathrm{wp}\ (\mathbf{ret}\ r_t)\ (\mathbf{ret}\ r_s)\ eq) & \textit{by}\ \text{UPD \& ASTR} \\ \hline
P \vdash \mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ (r_t\ r_s.\ \mathrm{wp}\ (\mathbf{ret}\ r_t)\ (\mathrm{ASSERT}(Q\ r_s); \mathbf{ret}\ r_s)\ eq) & \textit{by}\ \text{BIND} \\ \hline
P \vdash \mathrm{wp}\ \mathbb{T}\ (r \leftarrow \mathbb{S}; \mathrm{ASSERT}(Q\ r); \mathbf{ret}\ r)\ eq & \textit{by}\ \text{ASMR} \\ \hline
\top \vdash \mathrm{wp}\ \mathbb{T}\ (\mathrm{ASSUME}(P); r \leftarrow \mathbb{S}; \mathrm{ASSERT}(Q\ r); \mathbf{ret}\ r)\ eq & \textit{by}\ \text{INIT} \\ \hline
\mathbb{T} \sqsubseteq \mathrm{ASSUME}(P); r \leftarrow \mathbb{S}; \mathrm{ASSERT}(Q\ r); \mathbf{ret}\ r &
\end{array}$$



(INIT)

$$\frac{\top \vdash \mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ eq}{\mathbb{T} \sqsubseteq \mathbb{S}}$$

(RET)

$$\frac{\Phi\ r_t\ r_s}{\mathrm{wp}\ (\mathbf{ret}\ r_t)\ (\mathbf{ret}\ r_s)\ \Phi}$$

(BIND)

$$\frac{\mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ (r_t\ r_s.\ \mathrm{wp}\ (\mathbb{T}'\ r_t)\ (\mathbb{S}'\ r_s)\ \Phi)}{\mathrm{wp}\ (\mathbb{T} \gg= \mathbb{T}')\ (\mathbb{S} \gg= \mathbb{S}')\ \Phi}$$

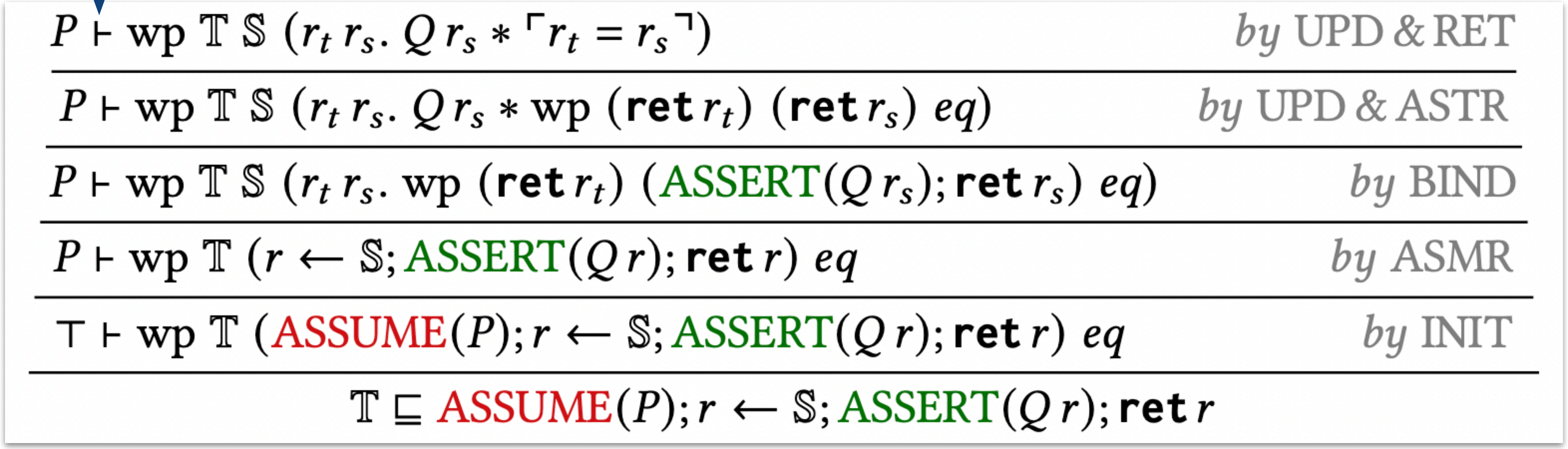
(UPD)

$$\frac{\ddot{\Rrightarrow}\mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ (r_t\ r_s.\ \ddot{\Rrightarrow}\Phi\ r_t\ r_s)}{\mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ (r_t\ r_s.\Phi\ r_t\ r_s)}$$

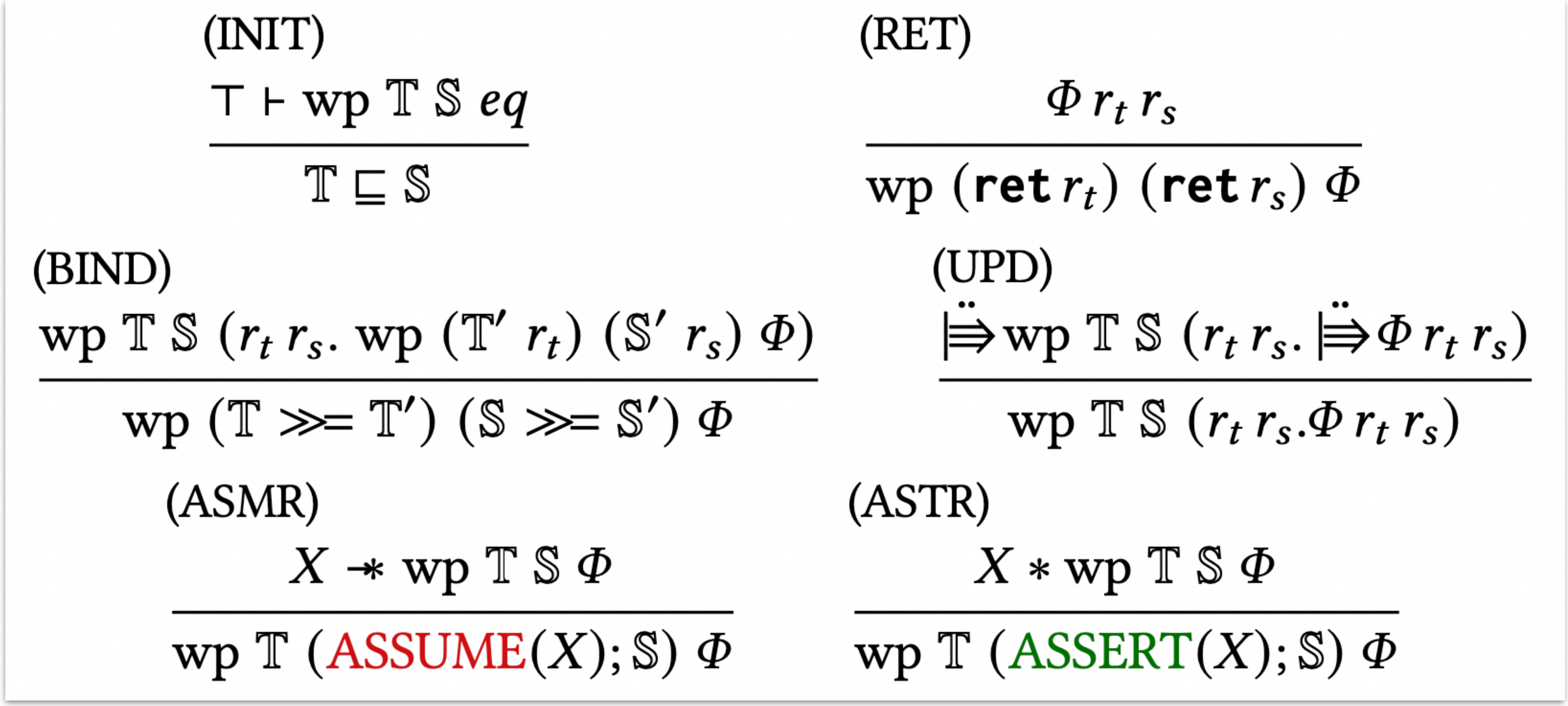
≃ Definition of Hoare Quadruple

{ P } T ≤ S { Q }

in SimulIris



$$\begin{array}{ll}
P \vdash \mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ (r_t\ r_s.\ Q\ r_s * \ulcorner r_t = r_s \urcorner) & \textit{by}\ \text{UPD \& RET} \\ \hline
P \vdash \mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ (r_t\ r_s.\ Q\ r_s * \mathrm{wp}\ (\mathbf{ret}\ r_t)\ (\mathbf{ret}\ r_s)\ eq) & \textit{by}\ \text{UPD \& ASTR} \\ \hline
P \vdash \mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ (r_t\ r_s.\ \mathrm{wp}\ (\mathbf{ret}\ r_t)\ (\mathrm{ASSERT}(Q\ r_s); \mathbf{ret}\ r_s)\ eq) & \textit{by}\ \text{BIND} \\ \hline
P \vdash \mathrm{wp}\ \mathbb{T}\ (r \leftarrow \mathbb{S}; \mathrm{ASSERT}(Q\ r); \mathbf{ret}\ r)\ eq & \textit{by}\ \text{ASMR} \\ \hline
\top \vdash \mathrm{wp}\ \mathbb{T}\ (\mathrm{ASSUME}(P); r \leftarrow \mathbb{S}; \mathrm{ASSERT}(Q\ r); \mathbf{ret}\ r)\ eq & \textit{by}\ \text{INIT} \\ \hline
\mathbb{T} \sqsubseteq \mathrm{ASSUME}(P); r \leftarrow \mathbb{S}; \mathrm{ASSERT}(Q\ r); \mathbf{ret}\ r &
\end{array}$$



(INIT)

$$\frac{\top \vdash \mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ eq}{\mathbb{T} \sqsubseteq \mathbb{S}}$$

(RET)

$$\frac{\Phi\ r_t\ r_s}{\mathrm{wp}\ (\mathbf{ret}\ r_t)\ (\mathbf{ret}\ r_s)\ \Phi}$$

(BIND)

$$\frac{\mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ (r_t\ r_s.\ \mathrm{wp}\ (\mathbb{T}'\ r_t)\ (\mathbb{S}'\ r_s)\ \Phi)}{\mathrm{wp}\ (\mathbb{T} \gg\!= \mathbb{T}')\ (\mathbb{S} \gg\!= \mathbb{S}')\ \Phi}$$

(UPD)

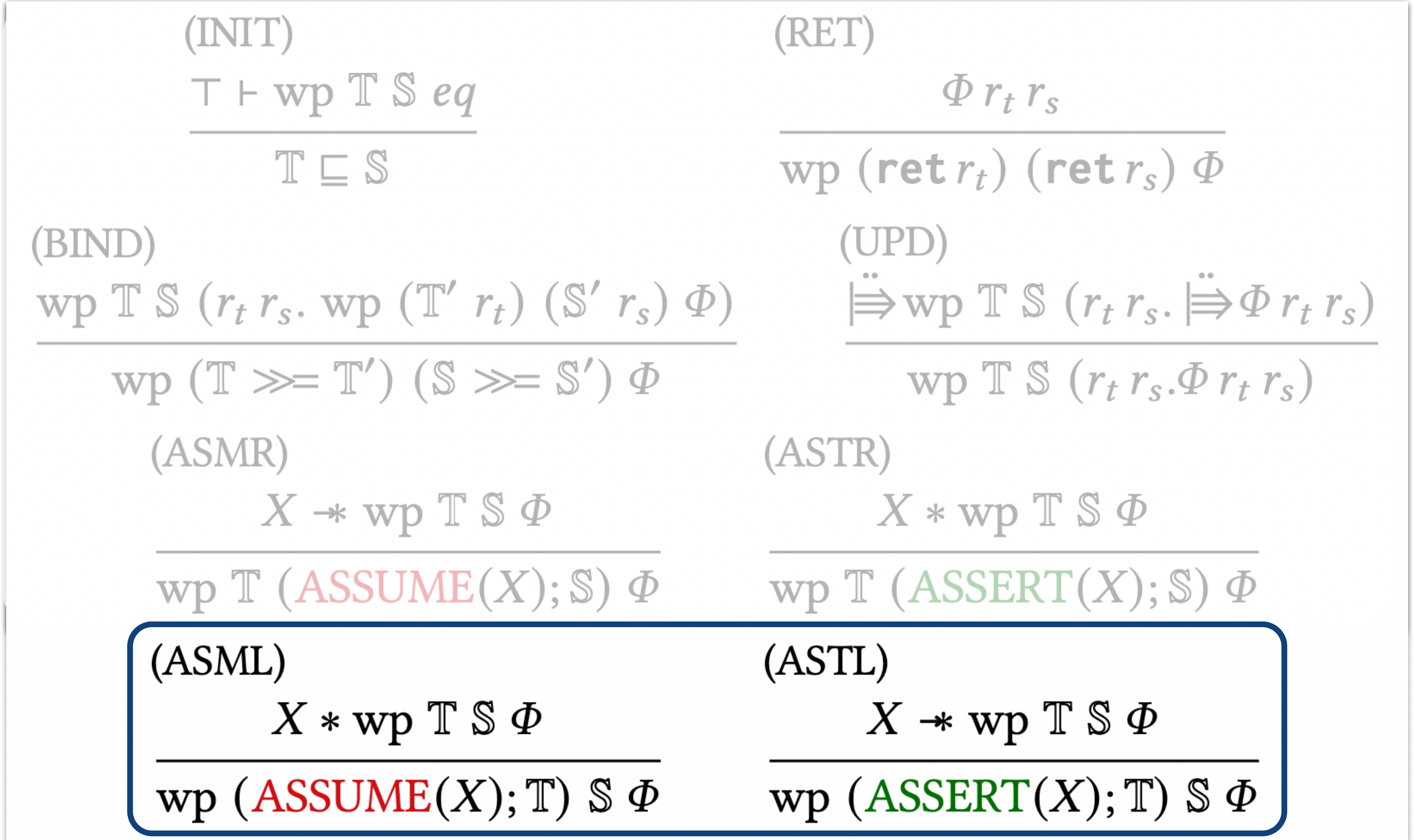
$$\frac{|\!\ddot{\Rrightarrow} \mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ (r_t\ r_s.\ |\!\ddot{\Rrightarrow} \Phi\ r_t\ r_s)}{\mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ (r_t\ r_s.\Phi\ r_t\ r_s)}$$

(ASMR)

$$\frac{X \mathrel{-\!\!*} \mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ \Phi}{\mathrm{wp}\ \mathbb{T}\ (\mathrm{ASSUME}(X); \mathbb{S})\ \Phi}$$

(ASTR)

$$\frac{X * \mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ \Phi}{\mathrm{wp}\ \mathbb{T}\ (\mathrm{ASSERT}(X); \mathbb{S})\ \Phi}$$



(INIT)

$$\frac{\top \vdash \mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ eq}{\mathbb{T} \sqsubseteq \mathbb{S}}$$

(RET)

$$\frac{\Phi\ r_t\ r_s}{\mathrm{wp}\ (\mathbf{ret}\ r_t)\ (\mathbf{ret}\ r_s)\ \Phi}$$

(BIND)

$$\frac{\mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ (r_t\ r_s.\ \mathrm{wp}\ (\mathbb{T}'\ r_t)\ (\mathbb{S}'\ r_s)\ \Phi)}{\mathrm{wp}\ (\mathbb{T} \gg= \mathbb{T}')\ (\mathbb{S} \gg= \mathbb{S}')\ \Phi}$$

(UPD)

$$\frac{\overset{\cdot\cdot}{\Rrightarrow} \mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ (r_t\ r_s.\ \overset{\cdot\cdot}{\Rrightarrow} \Phi\ r_t\ r_s)}{\mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ (r_t\ r_s. \Phi\ r_t\ r_s)}$$

(ASMR)

$$\frac{X \mathrel{-\!\!*} \mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ \Phi}{\mathrm{wp}\ \mathbb{T}\ (\mathrm{ASSUME}(X); \mathbb{S})\ \Phi}$$

(ASTR)

$$\frac{X * \mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ \Phi}{\mathrm{wp}\ \mathbb{T}\ (\mathrm{ASSERT}(X); \mathbb{S})\ \Phi}$$

(ASML)

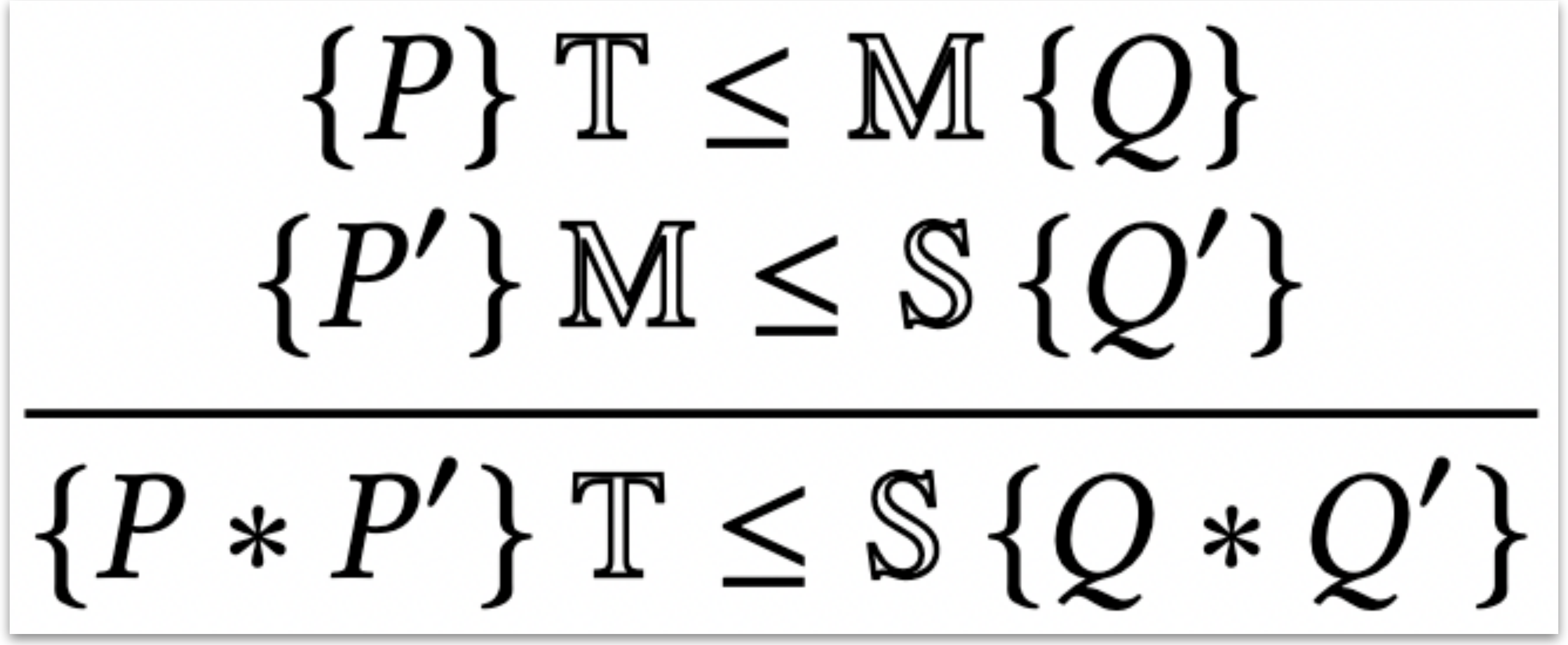
$$\frac{X * \mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ \Phi}{\mathrm{wp}\ (\mathrm{ASSUME}(X); \mathbb{T})\ \mathbb{S}\ \Phi}$$

(ASTL)

$$\frac{X \mathrel{-\!\!*} \mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ \Phi}{\mathrm{wp}\ (\mathrm{ASSERT}(X); \mathbb{T})\ \mathbb{S}\ \Phi}$$

Vertical Frame Rule

Vertical Frame Rule



$$\frac{\begin{array}{c}\{P\}\ \mathbb{T} \le \mathbb{M}\ \{Q\} \\ \{P'\}\ \mathbb{M} \le \mathbb{S}\ \{Q'\}\end{array}}{\{P * P'\}\ \mathbb{T} \le \mathbb{S}\ \{Q * Q'\}}$$

Vertical Frame Rule

Logic

$$\frac{\{P\}\,\mathbb{T} \leq \mathbb{M}\,\{Q\} \qquad \{P'\}\,\mathbb{M} \leq \mathbb{S}\,\{Q'\}}{\{P * P'\}\,\mathbb{T} \leq \mathbb{S}\,\{Q * Q'\}}$$

Model

$$\frac{\mathbb{T} \sqsubseteq \mathrm{ASSUME}(P);\mathbb{M};\mathrm{ASSERT}(Q) \qquad \mathbb{M} \sqsubseteq \mathrm{ASSUME}(P');\mathbb{S};\mathrm{ASSERT}(Q')}{\mathbb{T} \sqsubseteq \mathrm{ASSUME}(P * P');\mathbb{M};\mathrm{ASSERT}(Q * Q')}$$

Vertical Frame Rule

Logic

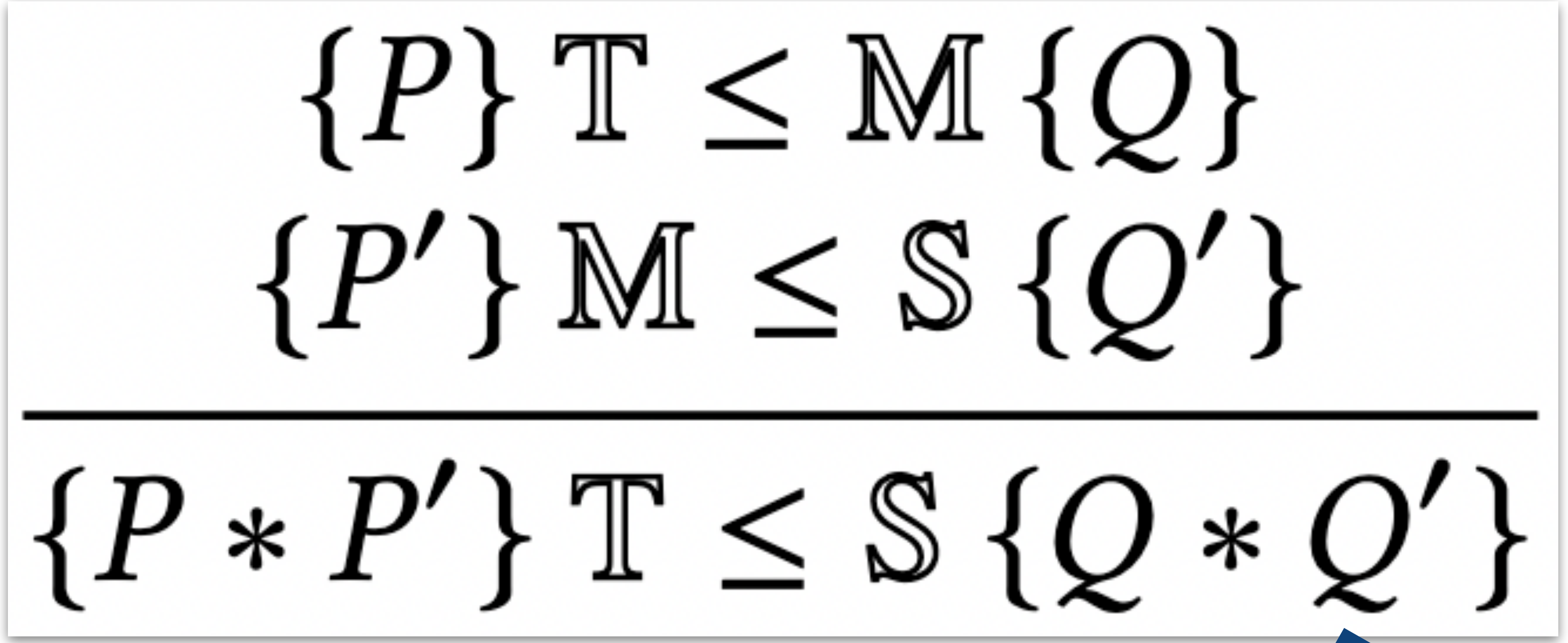
$$\frac{\{P\}\ \mathbb{T} \le \mathbb{M}\ \{Q\} \qquad \{P'\}\ \mathbb{M} \le \mathbb{S}\ \{Q'\}}{\{P * P'\}\ \mathbb{T} \le \mathbb{S}\ \{Q * Q'\}}$$

Model

$$\frac{\mathbb{T} \sqsubseteq \mathrm{ASSUME}(P); \mathbb{M}; \mathrm{ASSERT}(Q) \qquad \mathbb{M} \sqsubseteq \mathrm{ASSUME}(P'); \mathbb{S}; \mathrm{ASSERT}(Q')}{\mathbb{T} \sqsubseteq \mathrm{ASSUME}(P * P'); \mathbb{M}; \mathrm{ASSERT}(Q * Q')}$$

Transitivity does not apply here!

Vertical Frame Rule



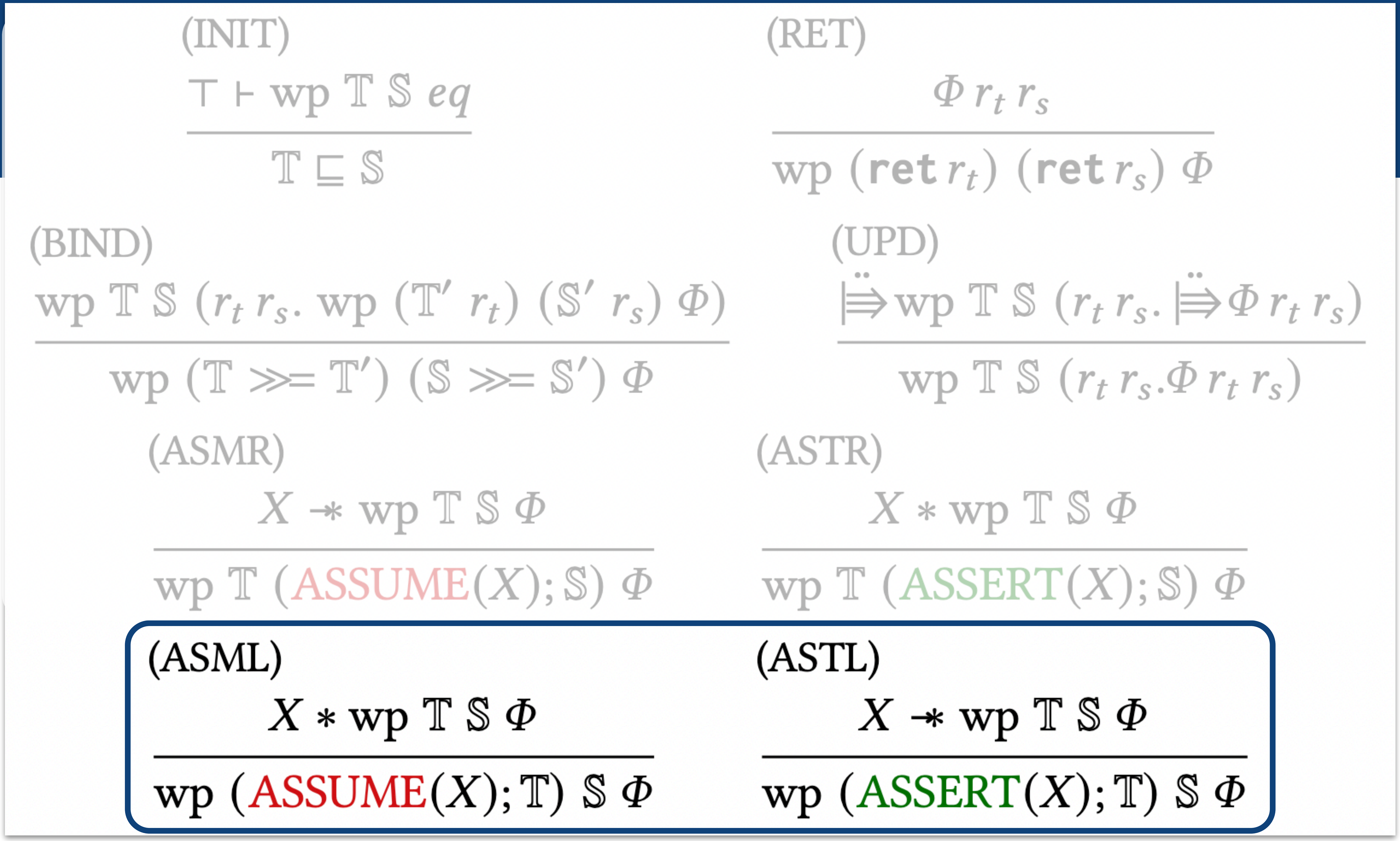
$$\frac{\{P\}\,\mathbb{T} \leq \mathbb{M}\,\{Q\} \qquad \{P'\}\,\mathbb{M} \leq \mathbb{S}\,\{Q'\}}{\{P * P'\}\,\mathbb{T} \leq \mathbb{S}\,\{Q * Q'\}}$$

$$\frac{\begin{array}{c}\forall X\,Y.\,\mathrm{ASSUME}(X);\mathbb{T};\mathrm{ASSERT}(Y) \sqsubseteq \mathrm{ASSUME}(X * P);\mathbb{M};\mathrm{ASSERT}(Y * Q)\\ \forall X\,Y.\,\mathrm{ASSUME}(X);\mathbb{M};\mathrm{ASSERT}(Y) \sqsubseteq \mathrm{ASSUME}(X * P');\mathbb{S};\mathrm{ASSERT}(Y * Q')\end{array}}{\forall X\,Y.\,\mathrm{ASSUME}(X);\mathbb{T};\mathrm{ASSERT}(Y) \sqsubseteq \mathrm{ASSUME}(X * P * P');\mathbb{M};\mathrm{ASSERT}(Y * Q * Q')}$$

Vertical Frame Rule

$$\frac{\{P\}\ \mathbb{T} \le \mathbb{M}\ \{Q\} \qquad \{P'\}\ \mathbb{M} \le \mathbb{S}\ \{Q'\}}{\{P * P'\}\ \mathbb{T} \le \mathbb{S}\ \{Q * Q'\}}$$

$$\frac{\begin{array}{c}\forall X\, Y.\ \mathrm{ASSUME}(X); \mathbb{T}; \mathrm{ASSERT}(Y) \sqsubseteq \mathrm{ASSUME}(X * P); \mathbb{M}; \mathrm{ASSERT}(Y * Q) \\ \forall X\, Y.\ \mathrm{ASSUME}(X); \mathbb{M}; \mathrm{ASSERT}(Y) \sqsubseteq \mathrm{ASSUME}(X * P'); \mathbb{S}; \mathrm{ASSERT}(Y * Q')\end{array}}{\forall X\, Y.\ \mathrm{ASSUME}(X); \mathbb{T}; \mathrm{ASSERT}(Y) \sqsubseteq \mathrm{ASSUME}(X * P * P'); \mathbb{M}; \mathrm{ASSERT}(Y * Q * Q')}$$



(INIT)
$$\frac{\top \vdash \mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ eq}{\mathbb{T} \sqsubseteq \mathbb{S}}$$

(RET)
$$\frac{\Phi\ r_t\ r_s}{\mathrm{wp}\ (\mathtt{ret}\ r_t)\ (\mathtt{ret}\ r_s)\ \Phi}$$

(BIND)
$$\frac{\mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ (r_t\ r_s.\ \mathrm{wp}\ (\mathbb{T}'\ r_t)\ (\mathbb{S}'\ r_s)\ \Phi)}{\mathrm{wp}\ (\mathbb{T} \gg= \mathbb{T}')\ (\mathbb{S} \gg= \mathbb{S}')\ \Phi}$$

(UPD)
$$\frac{\overset{\cdot\cdot}{\models}\!\!\Rrightarrow \mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ (r_t\ r_s.\ \overset{\cdot\cdot}{\models}\!\!\Rrightarrow \Phi\ r_t\ r_s)}{\mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ (r_t\ r_s.\Phi\ r_t\ r_s)}$$

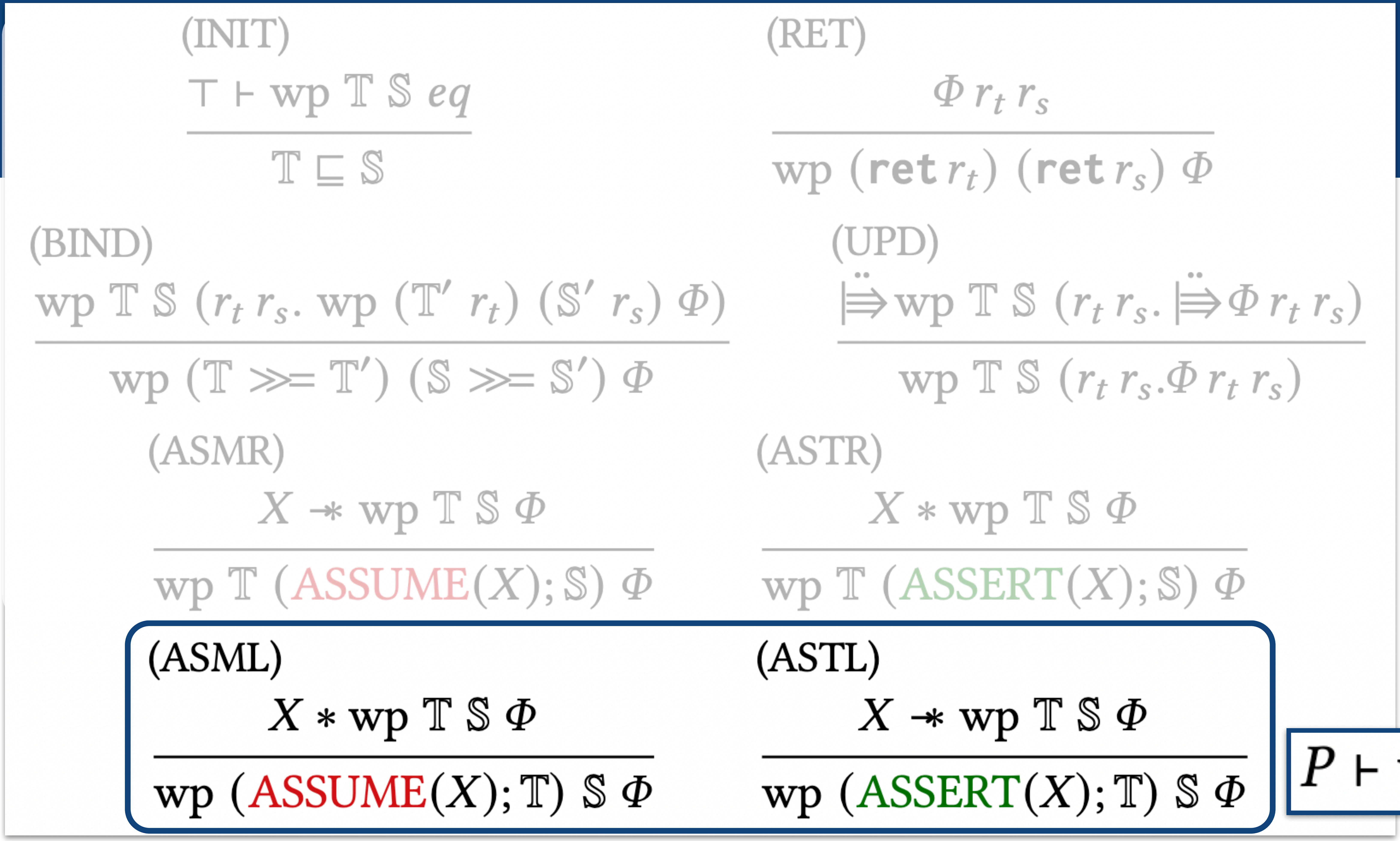
(ASMR)
$$\frac{X \mathrel{-\!\!*} \mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ \Phi}{\mathrm{wp}\ \mathbb{T}\ (\mathrm{ASSUME}(X); \mathbb{S})\ \Phi}$$

(ASTR)
$$\frac{X * \mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ \Phi}{\mathrm{wp}\ \mathbb{T}\ (\mathrm{ASSERT}(X); \mathbb{S})\ \Phi}$$

(ASML)
$$\frac{X * \mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ \Phi}{\mathrm{wp}\ (\mathrm{ASSUME}(X); \mathbb{T})\ \mathbb{S}\ \Phi}$$

(ASTL)
$$\frac{X \mathrel{-\!\!*} \mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ \Phi}{\mathrm{wp}\ (\mathrm{ASSERT}(X); \mathbb{T})\ \mathbb{S}\ \Phi}$$

$$\frac{\begin{array}{c}\forall X\ Y.\ \mathrm{ASSUME}(X); \mathbb{T}; \mathrm{ASSERT}(Y) \sqsubseteq \mathrm{ASSUME}(X * P); \mathbb{M}; \mathrm{ASSERT}(Y * Q)\\ \forall X\ Y.\ \mathrm{ASSUME}(X); \mathbb{M}; \mathrm{ASSERT}(Y) \sqsubseteq \mathrm{ASSUME}(X * P'); \mathbb{S}; \mathrm{ASSERT}(Y * Q')\end{array}}{\forall X\ Y.\ \mathrm{ASSUME}(X); \mathbb{T}; \mathrm{ASSERT}(Y) \sqsubseteq \mathrm{ASSUME}(X * P * P'); \mathbb{M}; \mathrm{ASSERT}(Y * Q * Q')}$$



(INIT)

$$\frac{\top \vdash \mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ eq}{\mathbb{T} \sqsubseteq \mathbb{S}}$$

(RET)

$$\frac{\Phi\ r_t\ r_s}{\mathrm{wp}\ (\mathbf{ret}\ r_t)\ (\mathbf{ret}\ r_s)\ \Phi}$$

(BIND)

$$\frac{\mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ (r_t\ r_s.\ \mathrm{wp}\ (\mathbb{T}'\ r_t)\ (\mathbb{S}'\ r_s)\ \Phi)}{\mathrm{wp}\ (\mathbb{T} \gg\!= \mathbb{T}')\ (\mathbb{S} \gg\!= \mathbb{S}')\ \Phi}$$

(UPD)

$$\frac{\overset{..}{\Rrightarrow} \mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ (r_t\ r_s.\ \overset{..}{\Rrightarrow} \Phi\ r_t\ r_s)}{\mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ (r_t\ r_s.\Phi\ r_t\ r_s)}$$

(ASMR)

$$\frac{X \mathrel{-\!\!*} \mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ \Phi}{\mathrm{wp}\ \mathbb{T}\ (\mathrm{ASSUME}(X);\mathbb{S})\ \Phi}$$

(ASTR)

$$\frac{X * \mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ \Phi}{\mathrm{wp}\ \mathbb{T}\ (\mathrm{ASSERT}(X);\mathbb{S})\ \Phi}$$

(ASML)

$$\frac{X * \mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ \Phi}{\mathrm{wp}\ (\mathrm{ASSUME}(X);\mathbb{T})\ \mathbb{S}\ \Phi}$$

(ASTL)

$$\frac{X \mathrel{-\!\!*} \mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ \Phi}{\mathrm{wp}\ (\mathrm{ASSERT}(X);\mathbb{T})\ \mathbb{S}\ \Phi}$$

$$P \vdash \mathrm{wp}\ \mathbb{T}\ \mathbb{S}\ (r_t\ r_s.\ Q\ r_s * \ulcorner r_t = r_s \urcorner)$$

$$\frac{\begin{array}{l}\forall X\ Y.\ \mathrm{ASSUME}(X);\mathbb{T};\mathrm{ASSERT}(Y) \sqsubseteq \mathrm{ASSUME}(X * P);\mathbb{M};\mathrm{ASSERT}(Y * Q)\\ \forall X\ Y.\ \mathrm{ASSUME}(X);\mathbb{M};\mathrm{ASSERT}(Y) \sqsubseteq \mathrm{ASSUME}(X * P');\mathbb{S};\mathrm{ASSERT}(Y * Q')\end{array}}{\forall X\ Y.\ \mathrm{ASSUME}(X);\mathbb{T};\mathrm{ASSERT}(Y) \sqsubseteq \mathrm{ASSUME}(X * P * P');\mathbb{M};\mathrm{ASSERT}(Y * Q * Q')}$$

One caveat

One caveat

- There is a very subtle **counterexample** when we have

One caveat

- There is a very subtle **counterexample** when we have
    - dual non-determinism

One caveat

- There is a very subtle **counterexample** when we have
  - dual non-determinism
  - aforementioned rules for ASSUME/ASSERT

One caveat

- There is a very subtle **counterexample** when we have
  - dual non-determinism
  - aforementioned rules for ASSUME/ASSERT
  - asynchronous execution of source and target

A Tweak in Update Modality

- The issue is addressed by:
  - a tweak in update modality
  - and a tweak on our model, correspondingly

A Tweak in Update Modality

- The issue is addressed by:
  - a tweak in update modality
  - and a tweak on our model, correspondingly

$$|\!\dot{\Rrightarrow} P \triangleq \lambda\, r.\, \forall\, ctx.\, \mathcal{V}(r \cdot ctx) \Rightarrow \exists\, r'.\, \mathcal{V}(r' \cdot ctx) \wedge P\, r'$$

$$|\!\ddot{\Rrightarrow} P \triangleq \lambda\, r.\, \exists\, r'.\, r \rightsquigarrow r' \wedge P\, r'$$

A Tweak in Update Modality

- The issue is addressed by:
  - a tweak in update modality
  - and a tweak on our model, correspondingly

$$\dot{\models\!\!\Rightarrow} P \triangleq \lambda\, r. \forall\, ctx. \mathcal{V}(r \cdot ctx) \Rightarrow \exists\, r'. \mathcal{V}(r' \cdot ctx) \wedge P\, r'$$

$$\ddot{\models\!\!\Rightarrow} P \triangleq \lambda\, r. \exists\, r'. r \rightsquigarrow r' \wedge P\, r'$$

$$\frac{a \rightsquigarrow B}{\mathrm{Own}\,(a) \vdash \dot{\models\!\!\Rightarrow} \exists b \in B.\, \mathrm{Own}\,(B)}$$

$$\frac{a \rightsquigarrow b}{\mathrm{Own}\,(a) \vdash \ddot{\models\!\!\Rightarrow} \mathrm{Own}\,(b)}$$

Future Works

Future Works

- More features in CCR 2.0 (regarding ITrees)

Future Works

- More features in CCR 2.0 (regarding ITrees)
- CCR-related
    - Prophecy?
    - Concurrency?
    - Specification-preserving compilation?

Future Works

- More features in CCR 2.0 (regarding ITrees)
- CCR-related
  - Prophecy?
  - Concurrency?
  - Specification-preserving compilation?

- + If you are interested in compiler/coinduction/algebraic effects/concurrency
  I am ready to chat!