Prophecy Variables in Separation Logic
(Extending Iris with Prophecy Variables)

Ralf Jung, Rodolphe Lepigre, Gaurav Parthasarathy, Marianna Rapoport, Amin Timany, Derek Dreyer, Bart Jacobs
MPI-SWS, KU Leuven, ETH Zürich, University of Waterloo

Iris Workshop – Aarhus, October 2019
Forward reasoning is often easier and more natural:

- Start at the beginning of a program’s execution
- Reason about how it behaves as it executes
Forward reasoning is often easier and more natural:

- Start at the beginning of a program’s execution
- Reason about how it behaves as it executes

Strictly forward reasoning is not always good enough!
Forward reasoning is often easier and more natural:

- Start at the beginning of a program’s execution
- Reason about how it behaves as it executes

Strictly forward reasoning is not always good enough!

Reasoning about the current execution step may require:

- Information about past events (this is usual)
- Knowledge of what will happen later in the execution
Auxiliary/ghost variables store information not present in the program’s physical state.

History variables [*Owicki & Gries 1976*] (past):

- Record what happened in the execution so far
- Introduced in the context of Hoare logic
- Widely used (modern form: user-defined ghost state)
Auxiliary/ghost variables store information not present in the program’s physical state.

History variables [Owicki & Gries 1976] (past):
- Record what happened in the execution so far
- Introduced in the context of Hoare logic
- Widely used (modern form: user-defined ghost state)

Prophecy variables [Abadi & Lamport 1991] (future):
- Predict what will happen later in the execution
- Introduced in the context of state machine refinement
- Fairly exotic, (almost) never used for Hoare logic
Motivating example: eager specification

Let us look at a simple coin implementation:

\[
\begin{align*}
\text{new_coin}() & \triangleq \{ \text{val} = \text{ref}(\text{nondet_bool}()) \} \\
\text{read_coin}(c) & \triangleq !c.\text{val}
\end{align*}
\]
Motivating example: eager specification

Let us look at a simple coin implementation:

\[
\text{new\_coin}() \triangleq \{ val = \text{ref}(\text{nondet\_bool}()) \}
\]
\[
\text{read\_coin}(c) \triangleq !c.val
\]

Used for the sake of presentation
Motivating example: eager specification

Let us look at a simple coin implementation:

\[
\begin{align*}
\text{new_coin}() & \triangleq \{ \text{val} = \text{ref}(\text{nondet_bool}()) \} \\
\text{read_coin}(c) & \triangleq !c.\text{val}
\end{align*}
\]

We consider an “eager” coin specification:

- A coin is only ever **tossed** once
- Reading its value **always** gives the same result

Used for the sake of presentation
Motivating example: eager specification

Let us look at a simple coin implementation:

\[
\text{new\_coin}() \triangleq \{ \text{val} = \text{ref}(\text{nondet\_bool}()) \}
\]
\[
\text{read\_coin}(c) \triangleq !c.\text{val}
\]

We consider an “eager” coin specification:

- A coin is only ever tossed once
- Reading its value always gives the same result

\[
\{ \text{True} \} \text{new\_coin}() \{ c. \exists b. \text{Coin}(c, b) \}
\]
\[
\{ \text{Coin}(c, b) \} \text{read\_coin}(c) \{ x. x = b \land \text{Coin}(c, b) \}
\]
Motivating example: eager specification

Let us look at a simple coin implementation:

\[
\text{new_coin}() \triangleq \{ \text{val} = \text{ref}\left(\text{nondet_bool()}\right) \}
\]

\[
\text{read_coin}(c) \triangleq !c.\text{val}
\]

We consider an “eager” coin specification:

- A coin is only ever \textit{tossed} once
- Reading its value \textit{always} gives the same result

\[
\{\text{True}\}\text{new_coin}()\{c. \exists b. \text{Coin}(c, b)\}
\]

\[
\{\text{Coin}(c, b)\}\text{read_coin}(c)\{x. x = b \land \text{Coin}(c, b)\}
\]

\[
\text{Coin}(c, b) \triangleq c.\text{val} \mapsto b
\]
Motivating example: lazy implementation

What if we want to flip the coin as late as possible?

```
new_coin () ≜ {
  val = ref (None)
}
read_coin (c) ≜
  match !c.val with
  Some (b) ⇒ b |
  None ⇒ let b = nondet _ bool (); c.val ← Some (b); b end
```
Motivating example: lazy implementation

What if we want to flip the coin as late as possible?

“Lazy” coin implementation:

\[
\text{new\_coin}() \triangleq \{ \text{val} = \text{ref}(\text{None}) \}
\]

\[
\text{read\_coin}(c) \triangleq \text{match}!c.\text{val} \text{ with}
\]

\[
\begin{align*}
\text{Some}(b) & \Rightarrow b \\
\text{None} & \Rightarrow \text{let } b = \text{nondet\_bool}(); \\
\end{align*}
\]

\[
\text{c.val} \gets \text{Some}(b); \quad b
\]

end
Motivating example: lazy implementation

What if we want to flip the coin as late as possible?

“Lazy” coin implementation:

\[
\text{new\_coin()} \triangleq \{ \text{val} = \text{ref}(\text{None}) \}
\]

\[
\text{read\_coin(c)} \triangleq \text{match}! \text{c.val with}
\]
\[
\text{Some}(b) \Rightarrow b
\]
\[
| \text{None} \Rightarrow \text{let} b = \text{nondet\_bool}(); \text{c.val} \leftarrow \text{Some}(b); b
\]
\]

To keep the same spec we need prophecy variables!!!
Prophecy variables have been used in:

- Verification tools based on reduction [Sezgin et al. 2010]
- Temporal logic [Cook & Koskinen 2011, Lamport & Merz 2017]
Prophecy variables have been used in:

- Verification tools based on reduction [Sezgin et al. 2010]
- Temporal logic [Cook & Koskinen 2011, Lamport & Merz 2017]

But never formally integrated into Hoare logic before!!!
Prior work on prophecy variables

Prophecy variables have been used in:

- Verification tools based on reduction [Sezgin et al. 2010]
- Temporal logic [Cook & Koskinen 2011, Lamport & Merz 2017]

But never formally integrated into Hoare logic before!!!

Only two previous attempts:

- Vafeiadis’s thesis [Vafeiadis 2007] (informal and flawed)
- Structural approach [Zhang et al. 2012] (too limited)
Our contribution: prophecy variables in Hoare logic

We are the first to give a formal account of prophecy variables in Hoare logic!

• Our results are all formalized in the Iris framework
• We also extended VeriFast with prophecy variables
• Useful to prove logical atomicity (RDCSS, HW Queue)
Our contribution: prophecy variables in Hoare logic

We are the first to give a formal account of prophecy variables in Hoare logic!

- Our results are all formalized in the Iris framework
- We also extended VeriFast with prophecy variables
- Useful to prove logical atomicity (RDCSS, HW Queue)

Presented this morning by Ralf

Prophecies help in case of “future-dependent” LP
Key idea of our approach

We leverage separation logic to easily ensure soundness!!!
Key idea of our approach

We leverage separation logic to easily ensure soundness!!

The high-level idea is to use new instruction for:

- Predicting a future observation \( \text{let } p = \text{NewProph} \)
- Realizing such an observation \( \text{Resolve } p \text{ to } v \)
Key idea of our approach

We leverage separation logic to easily ensure soundness!!!

Principles of prophecy variables in separation logic:

1. **The future is ours**
   - We model the right to resolve a prophecy as a resource
   - Proph$_1$($p$, $b$) gives exclusive right to resolve $p$
We leverage separation logic to easily ensure soundness!!!

Principles of prophecy variables in separation logic:

1. **The future is ours**
   - We model the right to resolve a prophecy as a resource
   - Proph$_1 (p, b)$ gives exclusive right to resolve $p$
Key idea of our approach

We leverage separation logic to easily ensure soundness!!

Principles of prophecy variables in separation logic:

1. **The future is ours**
   - We model the right to resolve a prophecy as a resource
   - $\text{Proph}_1(p, b)$ gives exclusive right to resolve $p$

2. **We must fulfill our destiny**
   - A prophecy can only be resolved to the predicted value
   - A contradiction can be derived if that is not the case
Prophecy variables are manipulated using ghost code
Prophecy variables are manipulated using **ghost code**

\[
\{ \text{True} \} \\
\text{NewProph} \quad \text{(Creates a one-shot prophecy variable } p \text{)} \\
\{ p. \exists b. \text{Proph}_{1}^{\mathbb{B}}(p, b) \} \\
\]
“One-shot” prophecy variable specification

Prophecy variables are manipulated using **ghost code**

\[
\{ \text{True} \}
\]

\textbf{NewProph} \quad \{ p. \exists b. \text{Proph}_1^B(p, b) \}

Provides an **exclusive** resolution token

(Creates a **one-shot** prophecy variable \( p \))

Consumes the resolution token

But we learn that the prophesied and resolved values are equal
Prophecy variables are manipulated using **ghost code**

\[
\{ \text{True} \}
\]

\[\text{NewProph} \quad \{ p. \ \exists b. \ \text{Proph}_B^1(p, b) \} \]

**Provides an exclusive resolution token**

\[\{ \text{Proph}_B^1(p, b) \} \]

\[\text{Resolve } p \text{ to } v \]

\[\{ v = b \} \]

(Resolves the prophecy \( p \) to value \( v \))

But we learn that the prophesied and resolved values are equal

\[\text{eight.osf} \]
Prophecy variables are manipulated using **ghost code**

\[
\begin{align*}
\{ \text{True} \} \\
\text{NewProph} \\
\{ p. \ \exists b. \ \text{Proph}_B^1(p, b) \} \\
\{ \text{Proph}_B^1(p, b) \} \\
\text{Resolve } p \text{ to } v \\
\{ v = b \}
\end{align*}
\]

- Provides an **exclusive** resolution token
- (Creates a **one-shot** prophecy variable \( p \))
- Consumes the resolution token
- (Resolves the prophecy \( p \) to value \( v \))
Prophecy variables are manipulated using **ghost code**

\[
\{ \text{True} \} \quad \text{NewProph} \quad \{ p. \ \exists b. \ \text{Proph}_1^B(p, b) \} \quad \{ \text{Proph}_1^B(p, b) \} \quad \text{Resolve } p \text{ to } v \quad \{ v = b \} 
\]

- **Provides an exclusive resolution token**
  - (Creates a **one-shot** prophecy variable \( p \))
- **Consumes the resolution token**
  - (Resolves the prophecy \( p \) to value \( v \))

But we learn that the prophesied and resolved values are **equal**
With the required ghost code the example becomes:

\[
\text{new_coin}() \triangleq \{ \text{val} = \text{ref}(\text{None}), \text{p} = \text{NewProph} \}
\]

\[
\text{read_coin}(c) \triangleq \text{match}!c.\text{val} \text{ with}
\]

\[
\begin{align*}
\text{Some}(b) & \Rightarrow b \\
| \text{None} & \Rightarrow \text{let } b = \text{nondet_bool}(); \\
& \quad \text{Resolve } c.\text{p} \text{ to } b; \\
& \quad c.\text{val} \leftarrow \text{Some}(b); \ b
\end{align*}
\]

The specification can be proved using:

\[
\text{Coin}(c, b) \triangleq (c.\text{val} \mapsto \text{Some}(b)) \lor (c.\text{val} \mapsto \text{Some}(b) \ast \text{Proph}_B) / \text{onosf}
\]
With the required ghost code the example becomes:

\[
\text{new\_coin}() \triangleq \{ \text{val} = \text{ref}(\text{None}), \text{p} = \text{NewProph} \}
\]

\[
\text{read\_coin}(c) \triangleq \text{match} ! \! c.\text{val} \text{ with}
\]

\[
\begin{align*}
\quad & \text{Some}(b) \Rightarrow b \\
\mid & \text{None} \Rightarrow \text{let } b = \text{nondet\_bool}(); \\
& \quad \text{Resolve } c.\text{p} \text{ to } b; \\
& \quad c.\text{val} \leftarrow \text{Some}(b); \ b
\end{align*}
\]

The specification can be proved using:

\[
\text{Coin}(c, b) \triangleq (c.\text{val} \mapsto \text{Some } b) \lor
\]

\[
(c.\text{val} \mapsto \text{None} \ast \text{Proph}_1^B (c.\text{p}, b))
\]
Is the one-shot prophecy mechanism general enough?

Consider the following coin implementation:

\[
\text{new\_coin}() \triangleq \{ \text{val} = \text{ref} (\text{nondet\_bool}()) \} \\
\text{read\_coin}(c) \triangleq \! c.\text{val} \\
\text{toss\_coin}(c) \triangleq c.\text{val} \leftarrow \text{nondet\_bool}();
\]
Is the one-shot prophecy mechanism general enough?

Consider the following coin implementation:

\[
\text{new_coin}() \triangleq \{ \text{val} = \text{ref}(\text{nondet_bool}()) \} \\
\text{read_coin}(c) \triangleq !c.\text{val} \\
\text{toss_coin}(c) \triangleq c.\text{val} \leftarrow \text{nondet_bool}();
\]

What if we want a “clairvoyant” specification?

\[
\{ \text{True} \} \text{new_coin}() \{ c. \exists bs. \text{Coin}(c, bs) \} \\
\{ \text{Coin}(c, bs) \} \text{read_coin}(c) \{ b. \exists bs'. bs = b :: bs' \land \text{Coin}(c, bs) \} \\
\{ \text{Coin}(c, bs) \} \text{toss_coin}(c) \{ \exists b, bs'. bs = b :: bs' \land \text{Coin}(c, bs') \}
\]
One shot is not enough

Generalization: prophecy a sequence of resolutions!

\{ \text{True} \}

\textbf{NewProph}

\{ p. \exists bs. \text{Proph}^B(p, bs) \}
One shot is not enough

Generalization: prophecy a sequence of resolutions!

\{ \text{True} \}

\text{NewProph}

\{ p. \exists bs. \text{Proph}^B (p, bs) \}

Prophecy assertion now holds a list
One shot is not enough

Generalization: prophecy a sequence of resolutions!

\{
\text{True}
\}

\text{NewProph}

\{p. \exists bs. \text{Proph}^B (p, bs)\}

\text{Prophecy assertion now holds a list}

\{\text{Proph}^B (p, bs)\}

\text{Resolve } p \text{ to } v

\{\exists bs'. bs = v :: bs' \land \text{Proph}^B (p, bs')\}
Generalization: prophecy a sequence of resolutions!

\[ \{ \text{True} \} \]
\[ \text{NewProph} \]
\[ \{ p. \ \exists bs. \ \text{Proph}^B(p, bs) \} \]

Prophecy assertion now holds a list

\[ \{ \text{Proph}^B(p, bs) \} \]
\[ \text{Resolve } p \text{ to } v \]
\[ \{ \exists bs'. \ bs = v :: bs' \land \text{Proph}^B(p, bs') \} \]

Resolving just pops one element

One shot is not enough
Generalization: prophecy a sequence of resolutions!

\{\text{True}\}

\text{NewProph}

\{ p. \exists bs. \text{Proph}^B(p, bs) \}

Prophecy assertion now holds a list

\{ \text{Proph}^B(p, bs) \}

\text{Resolve } p \text{ to } v

\{ \exists bs'. bs = v :: bs' \land \text{Proph}^B(p, bs') \}

Resolving just pops one element

One-shot prophecies can be encoded easily
Back to the clairvoyant coin example

Clairvoyant coin implementation:

\[
\text{new_coin}() \triangleq \text{let } v = \text{ref}(\text{nondet\_bool}());
\{ \text{val} = v, p = \text{NewProph} \}
\]

\[
\text{read_coin}(c) \triangleq !c.\text{val}
\]

\[
\text{toss_coin}(c) \triangleq \text{let } r = \text{nondet\_bool}();
\text{Resolve } c.p \text{ to } r;
\text{let } c.\text{val} \leftarrow r
\]

The specification can be proved using:

\[
\text{Coin}(c, bs) \triangleq \exists b, bs'. c.\text{val} \rightarrow b \land \text{Proph}B(p, bs') \land bs = b :: bs'
\]
Back to the clairvoyant coin example

Clairvoyant coin implementation:

\[
\text{new_coin}() \triangleq \text{let } v = \text{ref}(\text{nondet\_bool}()); \\
\{ \text{val} = v, p = \text{NewProph} \}
\]

\[
\text{read_coin}(c) \triangleq !c.\text{val}
\]

\[
\text{toss_coin}(c) \triangleq \text{let } r = \text{nondet\_bool}(); \\
\text{Resolve } c.p \text{ to } r; \\
\text{c.val } \leftarrow r
\]

The specification can be proved using:

\[
\text{Coin}(c, bs) \triangleq \exists b, bs'.\ c.\text{val} \mapsto b \land \text{Proph}^B(p, bs') \\
\land bs = b :: bs'
\]
A glimpse at the model of weakest pre

Modified model of weakest preconditions (simplified):

\[ \text{wp } e_1 \{ \Phi \} \triangleq \text{if } e_1 \in \text{Val} \text{ then } \Phi(e_1) \text{ else} \]

\[
\forall \sigma_1, \vec{\kappa}_1, \vec{\kappa}_2. S(\sigma_1, \vec{\kappa}_1 \uplus \vec{\kappa}_2) \Rightarrow \star \\
\text{reducible}(e_1, \sigma_1) \land \\
\forall e_2, \sigma_2, \vec{e}_f. ((e_1, \sigma_1) \rightarrow (e_2, \sigma_2, \vec{e}_f, \vec{\kappa}_1)) \Rightarrow \star \\
S(\sigma_2, \vec{\kappa}_2) \ast \text{wp } e_2 \{ \Phi \} \ast \star_{e \in \vec{e}_f} \text{wp } e \{ \text{True} \}
\]

\[
S(\sigma, \vec{\kappa}) \triangleq \begin{array}{c}
\bullet \sigma.1 \\
\end{array}^{\text{HEAP}} \ast \exists \Pi. \begin{array}{c}
\bullet \Pi, \begin{array}{c}
\end{array}^{\text{PROPH}} \land \text{dom}(\Pi) = \sigma.2 \land \\
\forall \{p \leftarrow \text{vs}\} \in \Pi. \text{vs} = \text{filter}(p, \vec{\kappa})
\end{array}
\]

\begin{align*}
\text{(return value)} \\
\text{(progress)} \\
\text{(preservation)} \\
\text{(state interp.)}
\end{align*}
A glimpse at the model of weakest pre

Modified model of weakest preconditions (simplified):

\[ \text{wp } e_1 \{ \Phi \} \triangleq \text{if } e_1 \in \text{Val} \text{ then } \Phi(e_1) \text{ else } \]

\[ \forall \sigma_1, \vec{\overline{k}}_1, \vec{\overline{k}}_2. S(\sigma_1, \vec{\overline{k}}_1 \oplus \vec{\overline{k}}_2) \Rightarrow^* \]

reducible\( (e_1, \sigma_1) \land \)

\[ \forall e_2, \sigma_2, \vec{e}_f. ((e_1, \sigma_1) \rightarrow (e_2, \sigma_2, \vec{e}_f, \vec{\overline{k}}_1)) \Rightarrow^* \]

\[ S(\sigma_2, \vec{\overline{k}}_2) \ast \text{wp } e_2 \{ \Phi \} \ast \ast_{e \in \vec{e}_f} \text{wp } e \{ \text{True} \} \] (progress)

\[ S(\sigma, \vec{\overline{k}}) \triangleq [\bullet \sigma.1]^{\text{HEAP}} \ast \exists \Pi. [\bullet \Pi]^{\text{PROPH}} \land \text{dom}(\Pi) = \sigma.2 \land \]

\[ \forall \{ p \leftarrow \text{vs} \} \in \Pi. \text{vs} = \text{filter}(p, \vec{\overline{k}}) \] (preservation)

\[ S(\sigma, \vec{\overline{k}}) \]

Reduction now collects “observations”
A glimpse at the model of weakest pre

Modified model of weakest preconditions (simplified):

\[
wp \, e_1 \{\Phi\} \triangleq \text{if } e_1 \in Val \text{ then } \Phi(e_1) \text{ else (return value)}
\]

\[
\forall \sigma_1, \vec{\kappa}_1, \vec{\kappa}_2. \ S(\sigma_1, \vec{\kappa}_1 \oplus \vec{\kappa}_2) \Rightarrow^{\ast} \text{(progress)}
\]

\[
\forall e_2, \sigma_2, \vec{e}_f. \ ((e_1, \sigma_1) \rightarrow (e_2, \sigma_2, \vec{e}_f, \vec{\kappa}_1)) \Rightarrow^{\ast} \text{(progress)}
\]

\[
S(\sigma_2, \vec{\kappa}_2) \ast wp \, e_2 \{\Phi\} \ast \ast_{e \in \vec{e}_f} \ wp \, e \{\text{True}\} \text{ (preservation)}
\]

\[
S(\sigma, \vec{\kappa}) \triangleq \left[\bullet \sigma.1\right]^\gamma_{\text{HEAP}} \ast \exists \Pi.\left[\bullet \Pi\right]^\gamma_{\text{PROPH}} \land \text{dom}(\Pi) = \sigma.2 \land
\]

\[
\forall\{p \leftarrow \text{vs}\} \in \Pi. \text{vs} = \text{filter}(p, \vec{\kappa}) \text{ (state interp.)}
\]
Iris now has support for prophecy variables:

- First formal integration into a program logic
- Useful for logically atomic specifications (Ralf’s talk)
- But that’s not the only application (see François’s talk)
Wrapping up!

Iris now has support for prophecy variables:

- First formal integration into a program logic
- Useful for logically atomic specifications (Ralf’s talk)
- But that’s not the only application (see François’s talk)

Things there was no time for:

- Atomic resolution of prophecy variables
- Logically atomic spec for RDCSS and Herlihy-Wing queue
- Erasure theorem (elimination of ghost code)
Wrapping up!

Iris now has support for prophecy variables:

- First formal integration into a program logic
- Useful for logically atomic specifications (Ralf’s talk)
- But that’s not the only application (see François’s talk)

Things there was no time for:

- Atomic resolution of prophecy variables
- Logically atomic spec for RDCSS and Herlihy-Wing queue
- Erasure theorem (elimination of ghost code)

• Erasure theorem (elimination of ghost code)
Thanks! Questions?

(For more details: https://iris-project.org)
Model of weakest preconditions in Iris

Encoding of weakest preconditions (simplified):

\[ \text{wp } e_1 \{ \Phi \} \triangleq \begin{cases} \text{if } e_1 \in Val \text{ then } \Phi(e_1) \text{ else} & \text{(return value)} \\ \forall \sigma_1. S(\sigma_1) \Rightarrow \star & \text{(progress)} \\ \text{reducible}(e_1, \sigma_1) \land \\ \forall e_2, \sigma_2, \vec{e}_f. ((e_1, \sigma_1) \rightarrow (e_2, \sigma_2, \vec{e}_f)) \Rightarrow \star & \text{(preservation)} \\ S(\sigma_2) \ast \text{wp } e_2 \{ \Phi \} \ast \star_{e \in \vec{e}_f} \text{wp } e \{ \text{True} \} & \end{cases} \]

\[ S(\sigma) \triangleq \gamma_{\text{HEAP}}^{\bullet \sigma} \]

Some intuitions about the involved components:

- The state interpretation holds the state of the physical heap
- View shifts \( P \Rightarrow Q \) allow updates to owned resources
- The actual definition uses the \( \triangleright P \) modality to avoid circularity
Operational semantics: head reduction and observations

We extend reduction rules with observations:

\[(\overline{n} + \overline{m}, \sigma) \rightarrow_h (\overline{n} + \overline{m}, \sigma, \epsilon, \epsilon)\]

\[(\text{ref}(v), \sigma) \rightarrow_h (\ell, \sigma \uplus \{\ell \leftarrow v\}, \epsilon, \epsilon)\]

\[(\ell \leftarrow w, \sigma \uplus \{\ell \leftarrow v\}) \rightarrow_h (\ell, \sigma \uplus \{\ell \leftarrow w\}, \epsilon, \epsilon)\]

\[(\text{fork} \{e\}, \sigma) \rightarrow_h ((), \sigma, e :: \epsilon, \epsilon)\]

\[(\text{Resolve } p \text{ to } v, \sigma) \rightarrow_h ((), \sigma, \epsilon, (p, v) :: \epsilon)\]

\[(\text{NewProph}, \sigma) \rightarrow_h (p, \sigma \uplus \{p\}, \epsilon, \epsilon)\]

A couple of remarks:

- Observations are only recorded on resolutions
- State \(\sigma\) now records the prophecy variables in scope
Extension for prophecy variables

Encoding of weakest preconditions (simplified):

\( \text{wp } e_1 \{ \Phi \} \triangleq \text{if } e_1 \in \text{Val} \text{ then } \Phi(e_1) \text{ else} \)

\( \forall \sigma_1, \vec{\kappa}_1, \vec{\kappa}_2. \ S(\sigma_1, \vec{\kappa}_1 ++ \vec{\kappa}_2) \equiv \ast \)

\( \text{reducible}(e_1, \sigma_1) \land \)

\( \forall e_2, \sigma_2, \vec{e}_f. ((e_1, \sigma_1) \rightarrow (e_2, \sigma_2, \vec{e}_f, \vec{\kappa}_1)) \equiv \ast \)

\( S(\sigma_2, \vec{\kappa}_2) \ast \text{wp } e_2 \{ \Phi \} \ast \ast_{e \in \vec{e}_f} \text{wp } e \{ \text{True} \} \)

\( S(\sigma, \vec{\kappa}) \triangleq [\bullet \sigma.1]^{\text{HEAP}} \ast \exists \Pi. [\bullet \Pi]^{\text{PROPH}} \land \text{dom}(\Pi) = \sigma.2 \land \)

\( \forall \{ p \leftarrow \text{vs} \} \in \Pi. \text{vs} = \text{filter}(p, \vec{\kappa}) \)

Some more intuitions about the involved components:

- State interpretation: holds observations yet to be made
- Observations are removed from the list when taking steps
Safety with respect to a (pure) predicate:

\[
Safe_\phi(e_1) \triangleq \forall \vec{e}s, \sigma, \vec{\kappa}. ([e_1], \emptyset) \rightarrow^*_{tp} (e_2 :: \vec{e}s, \sigma, \vec{\kappa})
\]

\[
\Rightarrow proper_\phi(e_2, \sigma) \land \forall e \in \vec{e}s. proper_{\text{True}}(e, \sigma)
\]

\[
proper_\psi(e, \sigma) \triangleq (e \in Val \land \psi(e)) \lor \text{reducible}(e, \sigma)
\]

**Theorem (adequacy).** Let \( e \) be an expression and \( \phi \) be a (pure) predicate. If \( \text{wp } e \{ \phi \} \) is provable then \( Safe_\phi(e) \).