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

Reasoning about the correctness of a program

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

Reasoning about the correctness of a program

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 correctness of a program

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

Remember the past, know the future

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)

Remember the past, know the future

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:

$$\begin{aligned}\texttt{new\_coin}() &\triangleq \{val = \texttt{ref}(nondet\_bool())\}\\ \texttt{read\_coin}(c) &\triangleq\ !c.val\end{aligned}$$

Motivating example: eager specification

Let us look at a simple coin implementation:

$$\begin{aligned}\texttt{new\_coin}() &\triangleq \{val = \texttt{ref}(nondet\_bool())\}\\ \texttt{read\_coin}(c) &\triangleq\ !c.val\end{aligned}$$

Used for the sake of presentation

Motivating example: eager specification

Let us look at a simple coin implementation:

$$\texttt{new\_coin}() \triangleq \{val = \texttt{ref}(nondet\_bool())\}$$
$$\texttt{read\_coin}(c) \triangleq\ !c.val$$

Used for the sake of presentation

We consider an "eager" coin specification:

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

Motivating example: eager specification

Let us look at a simple coin implementation:

$$\texttt{new\_coin}() \triangleq \{val = \texttt{ref}(nondet\_bool())\}$$
$$\texttt{read\_coin}(c) \triangleq\ !c.val$$

Used for the sake of presentation

We consider an "eager" coin specification:

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

$$\{\mathsf{True}\}\ \texttt{new\_coin}()\ \{c.\ \exists b.\ \mathsf{Coin}(c, b)\}$$
$$\{\mathsf{Coin}(c, b)\}\ \texttt{read\_coin}(c)\ \{x.\ x = b \land \mathsf{Coin}(c, b)\}$$

Motivating example: eager specification

Let us look at a simple coin implementation:

$$\texttt{new\_coin}() \triangleq \{val = \texttt{ref}(nondet\_bool())\}$$

$$\texttt{read\_coin}(c) \triangleq\ !c.val$$

Used for the sake of presentation

We consider an "eager" coin specification:

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

$$\{\mathrm{True}\}\ \texttt{new\_coin}()\ \{c.\ \exists b.\ \mathrm{Coin}(c, b)\}$$

$$\{\mathrm{Coin}(c, b)\}\ \texttt{read\_coin}(c)\ \{x.\ x = b \land \mathrm{Coin}(c, b)\}$$

$$\mathrm{Coin}(c, b) \triangleq c.val \mapsto b$$

Motivating example: lazy implementation

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

Motivating example: lazy implementation

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

"Lazy" coin implementation:

$$\texttt{new\_coin}() \triangleq \{\mathit{val} = \texttt{ref}(\mathrm{None})\}$$

$$\begin{aligned}
\texttt{read_coin}(c) \triangleq\ & \texttt{match}\,!\,c.\mathit{val}\ \texttt{with} \\
& \quad \mathrm{Some}(b) \Rightarrow b \\
& \mid \mathrm{None} \quad\ \ \Rightarrow \texttt{let}\ b = \mathit{nondet_bool}(); \\
& \qquad\qquad\qquad\ \ c.\mathit{val} \leftarrow \mathrm{Some}(b);\ b \\
& \texttt{end}
\end{aligned}$$

Motivating example: lazy implementation

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

"Lazy" coin implementation:

$$\begin{aligned}
\texttt{new_coin}() &\triangleq \{\mathit{val} = \texttt{ref}(\mathrm{None})\} \\
\texttt{read_coin}(c) &\triangleq \texttt{match}\ !\,c.\mathit{val}\ \texttt{with} \\
&\quad\ \ \mathrm{Some}(b) \Rightarrow b \\
&\quad \mid \mathrm{None} \quad\ \Rightarrow \texttt{let}\ b = \mathit{nondet_bool}(); \\
&\qquad\qquad\qquad\quad\ c.\mathit{val} \leftarrow \mathrm{Some}(b);\ b \\
&\quad \texttt{end}
\end{aligned}$$

To keep the same spec we need prophecy variables!!!

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*]

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!!!

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 ($\texttt{let}\ p = \texttt{NewProph}$)
- Realizing such an observation ($\texttt{Resolve}\ p\ \texttt{to}\ v$)

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
   - $\text{Proph}_1^{\mathbb{B}}(p, b)$ 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
   - $\mathrm{Proph}_1^{\mathbb{B}}(p, b)$ gives exclusive right to resolve p

“Assign a value to”

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
   - $\text{Proph}_1^{\mathbb{B}}(p, b)$ gives exclusive right to resolve p

“Assign a value to”

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

“One-shot” prophecy variable specification

Prophecy variables are manipulated using ghost code

"One-shot" prophecy variable specification

Prophecy variables are manipulated using ghost code

$\{\mathsf{True}\}$

`NewProph` (Creates a one-shot prophecy variable p)

$\left\{p.\ \exists b.\ \mathsf{Proph}_1^{\mathbb{B}}(p, b)\right\}$

“One-shot” prophecy variable specification

Prophecy variables are manipulated using ghost code

Provides an exclusive resolution token

$\{\mathrm{True}\}$

`NewProph`

$\{p.\ \exists b.\ \mathrm{Proph}_1^{\mathbb{B}}(p, b)\}$

(Creates a one-shot prophecy variable p)

“One-shot” prophecy variable specification

Prophecy variables are manipulated using ghost code

Provides an exclusive resolution token

$\{\mathrm{True}\}$

`NewProph` (Creates a one-shot prophecy variable p)

$\left\{p.\ \exists b.\ \mathrm{Proph}_1^{\mathbb{B}}(p, b)\right\}$

$\left\{\mathrm{Proph}_1^{\mathbb{B}}(p, b)\right\}$

`Resolve` p `to` v (Resolves the prophecy p to value v)

$\{v = b\}$

"One-shot" prophecy variable specification

Prophecy variables are manipulated using ghost code

$\{\mathsf{True}\}$

`NewProph`

$\{p.\ \exists b.\ \mathsf{Proph}_1^{\mathbb{B}}(p, b)\}$

Provides an exclusive resolution token

(Creates a one-shot prophecy variable p)

$\{\mathsf{Proph}_1^{\mathbb{B}}(p, b)\}$

`Resolve` p `to` v

$\{v = b\}$

Consumes the resolution token

(Resolves the prophecy p to value v)

"One-shot" prophecy variable specification

Prophecy variables are manipulated using ghost code

$\{\mathsf{True}\}$

`NewProph`

$\{p.\ \exists b.\ \mathsf{Proph}_1^{\mathbb{B}}(p, b)\}$

Provides an exclusive resolution token

(Creates a one-shot prophecy variable p)

$\{\mathsf{Proph}_1^{\mathbb{B}}(p, b)\}$

`Resolve` p `to` v

$\{v = b\}$

Consumes the resolution token

(Resolves the prophecy p to value v)

But we learn that the prophesied and resolved values are equal

Back to the lazy coin example

With the required ghost code the example becomes:

$$\texttt{new\_coin}() \triangleq \{val = \texttt{ref}(\mathrm{None}), p = \texttt{NewProph}\}$$

$$\begin{aligned}\texttt{read\_coin}(c) \triangleq\ & \texttt{match}\ !c.val\ \texttt{with} \\ & \quad \mathrm{Some}(b) \Rightarrow b \\ & \mid \mathrm{None} \quad\ \Rightarrow \texttt{let}\ b = nondet\_bool(); \\ & \qquad\qquad\quad \texttt{Resolve}\ c.p\ \texttt{to}\ b; \\ & \qquad\qquad\quad c.val \leftarrow \mathrm{Some}(b);\ b \\ & \texttt{end}\end{aligned}$$

Back to the lazy coin example

With the required ghost code the example becomes:

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

$$\begin{aligned}\texttt{read\_coin}(c) \triangleq\ & \texttt{match}\ !\,c.val\ \texttt{with} \\ & \quad \text{Some}(b) \Rightarrow b \\ & \mid \text{None} \quad\ \Rightarrow \texttt{let}\ b = nondet\_bool(); \\ & \qquad\qquad\qquad \texttt{Resolve}\ c.p\ \texttt{to}\ b; \\ & \qquad\qquad\qquad c.val \leftarrow \text{Some}(b);\ b \\ & \texttt{end}\end{aligned}$$

The specification can be proved using:

$$\begin{aligned}\text{Coin}(c, b) \triangleq\ & (c.val \mapsto \text{Some}\ b) \vee \\ & (c.val \mapsto \text{None} * \text{Proph}_1^{\mathbb{B}}(c.p, b))\end{aligned}$$

Is the one-shot prophecy mechanism general enough?

Consider the following coin implementation:

$$\texttt{new\_coin}() \triangleq \{\mathit{val} = \texttt{ref}(\mathit{nondet\_bool}())\}$$

$$\texttt{read\_coin}(c) \triangleq\ !\,c.\mathit{val}$$

$$\texttt{toss\_coin}(c) \triangleq c.\mathit{val} \leftarrow \mathit{nondet\_bool}();$$

Is the one-shot prophecy mechanism general enough?

Consider the following coin implementation:

$$\texttt{new\_coin}() \triangleq \{val = \texttt{ref}(nondet\_bool())\}$$

$$\texttt{read\_coin}(c) \triangleq\ !\,c.val$$

$$\texttt{toss\_coin}(c) \triangleq c.val \leftarrow nondet\_bool();$$

What if we want a "clairvoyant" specification?

$$\{\mathsf{True}\}\ \texttt{new\_coin}()\ \{c.\ \exists bs.\ \mathsf{Coin}(c, bs)\}$$

$$\{\mathsf{Coin}(c, bs)\}\ \texttt{read\_coin}(c)\ \{b.\ \exists bs'.\ bs = b :: bs' \wedge \mathsf{Coin}(c, bs)\}$$

$$\{\mathsf{Coin}(c, bs)\}\ \texttt{toss\_coin}(c)\ \{\exists b, bs'.\ bs = b :: bs' \wedge \mathsf{Coin}(c, bs')\}$$

One shot is not enough

Generalization: prophecy a sequence of resolutions!

$\{\mathsf{True}\}$

`NewProph`

$\{p.\ \exists bs.\ \mathsf{Proph}^{\mathbb{B}}(p, bs)\}$

One shot is not enough

Generalization: prophecy a sequence of resolutions!

$\{\mathsf{True}\}$

  `NewProph`

$\{p.\ \exists bs.\ \mathsf{Proph}^{\mathbb{B}}(p, bs)\}$

Prophecy assertion now holds a list

One shot is not enough

Generalization: prophecy a sequence of resolutions!

$\{\text{True}\}$

`NewProph`

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

Prophecy assertion now holds a list

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

`Resolve` p `to` v

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

One shot is not enough

Generalization: prophecy a sequence of resolutions!

$\{\mathrm{True}\}$

`NewProph`

$\{p.\ \exists bs.\ \mathrm{Proph}^{\mathbb{B}}(p, bs)\}$

Prophecy assertion now holds a list

$\{\mathrm{Proph}^{\mathbb{B}}(p, bs)\}$

`Resolve` p `to` v

$\{\exists bs'.\ bs = v :: bs' \wedge \mathrm{Proph}^{\mathbb{B}}(p, bs')\}$

Resolving just pops one element

One shot is not enough

Generalization: prophecy a sequence of resolutions!

$\{\text{True}\}$

`NewProph`

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

Prophecy assertion now holds a list

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

`Resolve` p `to` v

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

Resolving just pops one element

One-shot prophecies can be encoded easily

Back to the clairvoyant coin example

Clairvoyant coin implementation:

$$\begin{aligned}
\texttt{new_coin}() &\triangleq \texttt{let}\, v = \texttt{ref}(\mathit{nondet_bool}());\\
&\quad \{\mathit{val} = v, p = \texttt{NewProph}\}\\[1em]
\texttt{read_coin}(c) &\triangleq\, !\, c.\mathit{val}\\[1em]
\texttt{toss_coin}(c) &\triangleq \texttt{let}\, r = \mathit{nondet_bool}();\\
&\quad \texttt{Resolve}\, c.p\, \texttt{to}\, r;\\
&\quad c.\mathit{val} \leftarrow r
\end{aligned}$$

Back to the clairvoyant coin example

Clairvoyant coin implementation:

$$\begin{aligned}\texttt{new\_coin}() &\triangleq \texttt{let}\ v = \texttt{ref}(nondet\_bool());\\ &\quad\ \{val = v, p = \texttt{NewProph}\}\end{aligned}$$

$$\texttt{read\_coin}(c) \triangleq\ !\,c.val$$

$$\begin{aligned}\texttt{toss\_coin}(c) &\triangleq \texttt{let}\ r = nondet\_bool();\\ &\quad\ \texttt{Resolve}\ c.p\ \texttt{to}\ r;\\ &\quad\ c.val \leftarrow r\end{aligned}$$

The specification can be proved using:

$$\begin{aligned}\mathrm{Coin}(c, bs) \triangleq \exists b, bs'.\ c.val \mapsto b \wedge \mathrm{Proph}^{\mathbb{B}}(p, bs')\\ \wedge\ bs = b :: bs'\end{aligned}$$

A glimpse at the model of weakest pre

Modified model of weakest preconditions (simplified):

$$\begin{aligned}
\mathsf{wp}\; e_1 \,\{\Phi\} \triangleq\; & \text{if } e_1 \in \mathit{Val} \text{ then } \Phi(e_1) \text{ else} && \underline{\text{(return value)}}\\
& \forall \sigma_1, \vec{\kappa}_1, \vec{\kappa}_2.\, S(\sigma_1, \vec{\kappa}_1 +\!\!+ \vec{\kappa}_2) \;{=\!\!\ast}\\
& \quad \mathrm{reducible}(e_1, \sigma_1) \wedge && \underline{\text{(progress)}}\\
& \quad \left.\begin{aligned} & \forall e_2, \sigma_2, \vec{e}_f.\, ((e_1, \sigma_1) \to (e_2, \sigma_2, \vec{e}_f, \vec{\kappa}_1)) \;{=\!\!\ast}\\ & \quad S(\sigma_2, \vec{\kappa}_2) \ast \mathsf{wp}\; e_2 \,\{\Phi\} \ast \textstyle\bigast_{e \in \vec{e}_f} \mathsf{wp}\; e \,\{\mathsf{True}\}\end{aligned}\right\} && \underline{\text{(preservation)}}
\end{aligned}$$

$$\begin{aligned}
S(\sigma, \vec{\kappa}) \triangleq\; & \boxed{\bullet\, \sigma.1}^{\gamma_{\mathrm{HEAP}}} \ast\; \exists \Pi.\, \boxed{\bullet\, \Pi}^{\gamma_{\mathrm{PROPH}}} \wedge \mathrm{dom}(\Pi) = \sigma.2 \wedge && \underline{\text{(state interp.)}}\\
& \forall \{p \leftarrow vs\} \in \Pi.\, vs = \mathrm{filter}(p, \vec{\kappa})
\end{aligned}$$

Modified model of weakest preconditions (simplified):

$$\begin{aligned}
\mathsf{wp}\ e_1\ \{\Phi\} &\triangleq \text{if } e_1 \in \mathit{Val} \text{ then } \Phi(e_1) \text{ else} && \text{(return value)}\\
&\quad \forall \sigma_1, \vec{\kappa}_1, \vec{\kappa}_2.\, S(\sigma_1, \vec{\kappa}_1 \mathbin{+\!\!+} \vec{\kappa}_2) \Rrightarrow\!\!\ast \\
&\qquad \mathrm{reducible}(e_1, \sigma_1) \land && \text{(progress)}\\
&\qquad \left.\begin{aligned}&\forall e_2, \sigma_2, \vec{e}_f.\, ((e_1, \sigma_1) \to (e_2, \sigma_2, \vec{e}_f, \vec{\kappa}_1)) \Rrightarrow\!\!\ast \\ &\quad S(\sigma_2, \vec{\kappa}_2) \ast \mathsf{wp}\ e_2\ \{\Phi\} \ast \mathop{\huge\ast}_{e \in \vec{e}_f} \mathsf{wp}\ e\ \{\mathsf{True}\}\end{aligned}\right\} && \text{(preservation)}
\end{aligned}$$

$$\begin{aligned}
S(\sigma, \vec{\kappa}) &\triangleq \boxed{\bullet\, \sigma.1}^{\gamma_{\mathrm{HEAP}}} \ast\ \exists \Pi.\, \boxed{\bullet\, \Pi}^{\gamma_{\mathrm{PROPH}}} \land \mathrm{dom}(\Pi) = \sigma.2 \land && \text{(state interp.)}\\
&\qquad \forall \{p \leftarrow vs\} \in \Pi.\, vs = \mathrm{filter}(p, \vec{\kappa})
\end{aligned}$$

Reduction now collects "observations"

A glimpse at the model of weakest pre

Modified model of weakest preconditions (simplified):

$$\begin{aligned}
&\mathrm{wp}\ e_1\ \{\Phi\} \triangleq \text{if } e_1 \in \mathit{Val} \text{ then } \Phi(e_1) \text{ else} && \underline{\text{(return value)}}\\
&\qquad \forall \sigma_1, \vec{\kappa}_1, \vec{\kappa}_2.\, S(\sigma_1, \vec{\kappa}_1 \mathbin{+\!\!+} \vec{\kappa}_2) \Rrightarrow\!\!\ast \\
&\qquad\quad \mathrm{reducible}(e_1, \sigma_1) \wedge && \underline{\text{(progress)}}\\
&\qquad\quad \forall e_2, \sigma_2, \vec{e}_f.\, ((e_1, \sigma_1) \to (e_2, \sigma_2, \vec{e}_f, \vec{\kappa}_1)) \Rrightarrow\!\!\ast && \underline{\text{(preservation)}}\\
&\qquad\qquad S(\sigma_2, \vec{\kappa}_2) * \mathrm{wp}\ e_2\ \{\Phi\} * \mathop{\Huge\ast}_{e \in \vec{e}_f} \mathrm{wp}\ e\ \{\mathrm{True}\}
\end{aligned}$$

$$\begin{aligned}
S(\sigma, \vec{\kappa}) \triangleq\ & \boxed{\bullet\, \sigma.1}^{\gamma_{\text{HEAP}}} * \exists \Pi.\, \boxed{\bullet\, \Pi}^{\gamma_{\text{PROPH}}} \wedge \mathrm{dom}(\Pi) = \sigma.2 \wedge && \underline{\text{(state interp.)}}\\
& \forall \{p \leftarrow vs\} \in \Pi.\, vs = \mathrm{filter}(p, \vec{\kappa})
\end{aligned}$$

Observations yet to be made

Reduction now collects "observations"

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)

Wrapping up!

Iris now has support for prophecy variables:

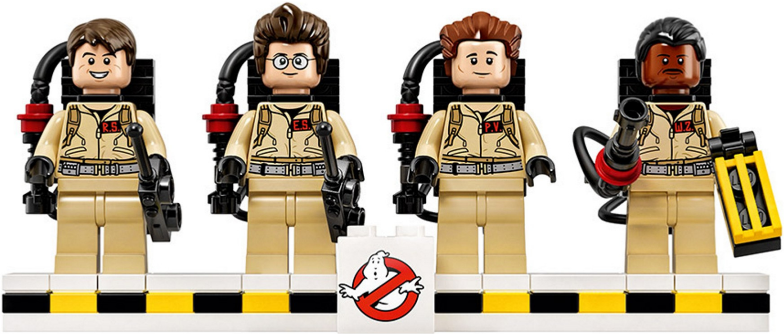
- First formal integration into a program logic
- Useful for logically atomic specifications (Ralf's talk)
- But that's not the only application (see François's talk)

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:



- 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):

$$
\begin{array}{ll}
\mathsf{wp}\ e_1\ \{\Phi\} \triangleq \text{if } e_1 \in \mathit{Val} \text{ then } \Phi(e_1) \text{ else} & \underline{\text{(return value)}} \\
\quad \forall \sigma_1.\, S(\sigma_1) \Rrightarrow\!\!\ast & \\
\quad\quad \mathrm{reducible}(e_1, \sigma_1) \wedge & \underline{\text{(progress)}} \\
\quad\quad \forall e_2, \sigma_2, \vec{e}_f.\, \big((e_1, \sigma_1) \to (e_2, \sigma_2, \vec{e}_f)\big) \Rrightarrow\!\!\ast & \multirow{2}{*}{$\Big\}\ \underline{\text{(preservation)}}$} \\
\quad\quad\quad S(\sigma_2) * \mathsf{wp}\ e_2\ \{\Phi\} * \mathop{\huge\ast}_{e \in \vec{e}_f} \mathsf{wp}\ e\ \{\mathsf{True}\} & \\
S(\sigma) \triangleq \left[\bullet\, \sigma\right]^{\gamma_{\mathrm{HEAP}}} & \underline{\text{(state interp.)}}
\end{array}
$$

Some intuitions about the involved components:

- The state interpretation holds the state of the physical heap
- View shifts $P \Rrightarrow\!\!\ast\ Q$ 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:

$$
\begin{aligned}
(\overline{n} + \overline{m}, \sigma) &\to_{\mathsf{h}} (\overline{n+m}, \sigma, \epsilon, \epsilon) \\
(\mathtt{ref}(v), \sigma) &\to_{\mathsf{h}} (\ell, \sigma \uplus \{\ell \leftarrow v\}, \epsilon, \epsilon) \\
(\ell \leftarrow w, \sigma \uplus \{\ell \leftarrow v\}) &\to_{\mathsf{h}} (\ell, \sigma \uplus \{\ell \leftarrow w\}, \epsilon, \epsilon) \\
(\mathtt{fork}\ \{e\}, \sigma) &\to_{\mathsf{h}} ((), \sigma, e :: \epsilon, \epsilon) \\
(\mathtt{Resolve}\, p\, \mathtt{to}\, v, \sigma) &\to_{\mathsf{h}} ((), \sigma, \epsilon, (p, v) :: \epsilon) \\
(\mathtt{NewProph}, \sigma) &\to_{\mathsf{h}} (p, \sigma \uplus \{p\}, \epsilon, \epsilon)
\end{aligned}
$$

A couple of remarks:

- Observations are only recorded on resolutions
- State σ now records the prophecy variables in scope

Extension for prophecy variables

Encoding of weakest preconditions (simplified):

$$
\begin{aligned}
\mathsf{wp}\ e_1\ \{\Phi\} \triangleq{}& \text{if } e_1 \in \mathit{Val} \text{ then } \Phi(e_1) \text{ else} && \underline{\text{(return value)}}\\
& \forall \sigma_1, \vec{\kappa}_1, \vec{\kappa}_2.\, S(\sigma_1, \vec{\kappa}_1 +\!\!+\, \vec{\kappa}_2) \Rrightarrow\!\!\ast \\
& \quad \mathrm{reducible}(e_1, \sigma_1) \wedge && \underline{\text{(progress)}}\\
& \quad \forall e_2, \sigma_2, \vec{e}_f.\, ((e_1, \sigma_1) \to (e_2, \sigma_2, \vec{e}_f, \vec{\kappa}_1)) \Rrightarrow\!\!\ast && \\
& \qquad S(\sigma_2, \vec{\kappa}_2) * \mathsf{wp}\ e_2\ \{\Phi\} * \mathop{\Large\ast}_{e \in \vec{e}_f} \mathsf{wp}\ e\ \{\mathrm{True}\} && \Big\}\underline{\text{(preservation)}}
\end{aligned}
$$

$$
\begin{aligned}
S(\sigma, \vec{\kappa}) \triangleq{}& \boxed{\bullet\, \sigma.1}^{\gamma_{\text{HEAP}}} * \ \exists \Pi.\, \boxed{\bullet\, \Pi}^{\gamma_{\text{PROPH}}} \wedge \mathrm{dom}(\Pi) = \sigma.2 \wedge && \underline{\text{(state interp.)}}\\
& \forall \{p \leftarrow vs\} \in \Pi.\, vs = \mathrm{filter}(p, \vec{\kappa})
\end{aligned}
$$

Some more intuitions about the involved components:

- State interpretation: holds observations yet to be made
- Observations are removed from the list when taking steps

Statement of safety and adequacy

Safety with respect to a (pure) predicate:

$$\mathit{Safe}_\phi(e_1) \triangleq \forall \vec{es}, \sigma, \vec{\kappa}.\ ([e_1], \emptyset) \to^*_{\mathsf{tp}} (e_2 :: \vec{es}, \sigma, \vec{\kappa})$$
$$\Rightarrow \mathit{proper}_\phi(e_2, \sigma) \wedge \forall e \in \vec{es}.\ \mathit{proper}_{\mathsf{True}}(e, \sigma)$$
$$\mathit{proper}_\psi(e, \sigma) \triangleq (e \in \mathit{Val} \wedge \psi(e)) \vee \mathrm{reducible}(e, \sigma)$$

Theorem (adequacy). Let e be an expression and ϕ be a (pure) predicate. If wp $e\,\{\phi\}$ is provable then $\mathit{Safe}_\phi(e)$.