Completeness of Iris-based Program Logics

Zichen Zhang[*]

with

Johannes Hostert[†], Puming Liu[‡], Simon Gregersen[§], Ralf Jung[†], and Joseph Tassarotti[*]

[*] New York University, [†] ETH Zurich, [‡] NYU Shanghai, [§] CISPA Helmholtz Center for Information Security

June 10, 2026

The Sixth Edition of the Iris Workshop Vienna, Austria

A Missing Piece in Iris

Logic

Truth

1. Let's think about one question: why do we want a program logic? It turns out that we want use it to show some truths about programs via proving assertions in the logic.
2. **[click]** In Iris, the truth is the safety of a program, which means the evaluation of expression e never gets stuck, and the result satisfies a meta-level predicate φ if it terminates.
3. **[click]** and the assertion in the logic that we want to prove is a wp of e of a pure postcondition
4. **[click]** The first thing we want to make sure is soundness, namely, everything proved in the logic is actually true at meta level
5. This is the bottom line of a correctness logic, and indeed **[click]** every Iris-based program logic comes with a soundness theorem, which is often known as adequacy
6. **[click]** But what about the other direction? Namely, completeness, which ensures every truth is provable.
7. **[click]** The significance of completeness is that it directly answers how expressive the logic is, and helps us find out whether there are some missing rules in the logic.
8. **[click]** Indeed, many prior work has shown completeness for other forms of logics
9. Since these prior works considered first-order languages, and used very different form of ghost state, they cannot be directly reused in Iris.
10. **[click]** In fact, it remains unclear whether Iris-based logics are complete or not.

A Missing Piece in Iris

Logic

Truth

$\mathsf{safe}_{\varphi}(e, \sigma)$

Never gets stuck

Result satisfies φ

1. Let's think about one question: why do we want a program logic? It turns out that we want use it to show some truths about programs via proving assertions in the logic.
2. **[click]** In Iris, the truth is the safety of a program, which means the evaluation of expression e never gets stuck, and the result satisfies a meta-level predicate φ if it terminates.
3. **[click]** and the assertion in the logic that we want to prove is a wp of e of a pure postcondition
4. **[click]** The first thing we want to make sure is soundness, namely, everything proved in the logic is actually true at meta level
5. This is the bottom line of a correctness logic, and indeed **[click]** every Iris-based program logic comes with a soundness theorem, which is often known as adequacy
6. **[click]** But what about the other direction? Namely, completeness, which ensures every truth is provable.
7. **[click]** The significance of completeness is that it directly answers how expressive the logic is, and helps us find out whether there are some missing rules in the logic.
8. **[click]** Indeed, many prior work has shown completeness for other forms of logics
9. Since these prior works considered first-order languages, and used very different form of ghost state, they cannot be directly reused in Iris.
10. **[click]** In fact, it remains unclear whether Iris-based logics are complete or not.

A Missing Piece in Iris

$\vdash \mathsf{wp}\ e\ \{v.\ \ulcorner\varphi(v)\urcorner\}$

Logic

$\mathsf{safe}_{\varphi}(e, \sigma)$

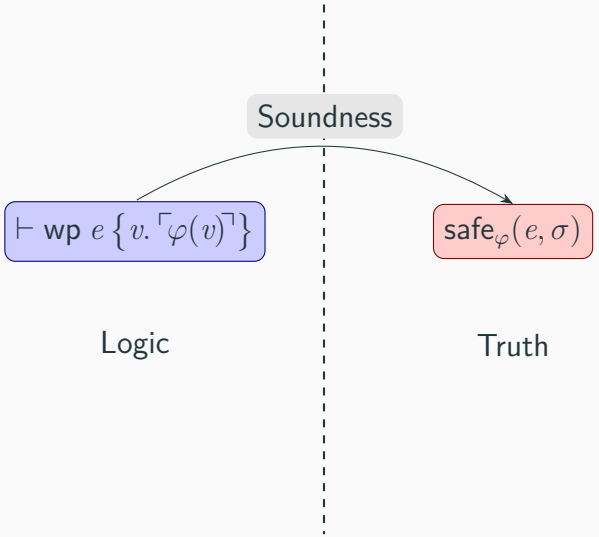
Truth

Never gets stuck

Result satisfies φ

1. Let's think about one question: why do we want a program logic? It turns out that we want use it to show some truths about programs via proving assertions in the logic.
2. **[click]** In Iris, the truth is the safety of a program, which means the evaluation of expression e never gets stuck, and the result satisfies a meta-level predicate φ if it terminates.
3. **[click]** and the assertion in the logic that we want to prove is a wp of e of a pure postcondition
4. **[click]** The first thing we want to make sure is soundness, namely, everything proved in the logic is actually true at meta level
5. This is the bottom line of a correctness logic, and indeed **[click]** every Iris-based program logic comes with a soundness theorem, which is often known as adequacy
6. **[click]** But what about the other direction? Namely, completeness, which ensures every truth is provable.
7. **[click]** The significance of completeness is that it directly answers how expressive the logic is, and helps us find out whether there are some missing rules in the logic.
8. **[click]** Indeed, many prior work has shown completeness for other forms of logics
9. Since these prior works considered first-order languages, and used very different form of ghost state, they cannot be directly reused in Iris.
10. **[click]** In fact, it remains unclear whether Iris-based logics are complete or not.

A Missing Piece in Iris

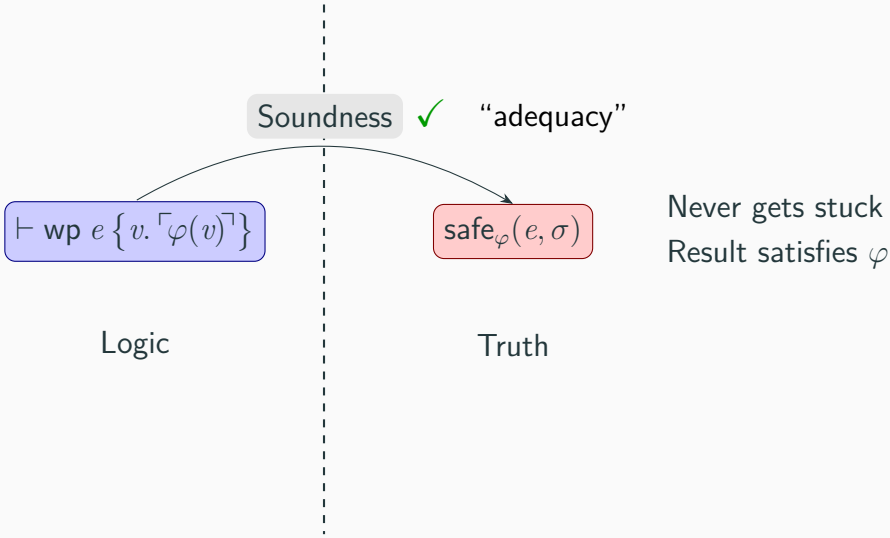


Never gets stuck

Result satisfies φ

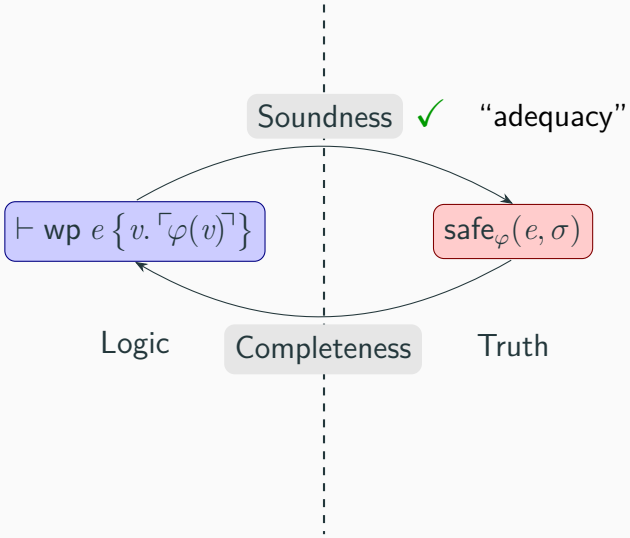
1. Let's think about one question: why do we want a program logic? It turns out that we want use it to show some truths about programs via proving assertions in the logic.
2. **[click]** In Iris, the truth is the safety of a program, which means the evaluation of expression e never gets stuck, and the result satisfies a meta-level predicate φ if it terminates.
3. **[click]** and the assertion in the logic that we want to prove is a wp of e of a pure postcondition
4. **[click]** The first thing we want to make sure is soundness, namely, everything proved in the logic is actually true at meta level
5. This is the bottom line of a correctness logic, and indeed **[click]** every Iris-based program logic comes with a soundness theorem, which is often known as adequacy
6. **[click]** But what about the other direction? Namely, completeness, which ensures every truth is provable.
7. **[click]** The significance of completeness is that it directly answers how expressive the logic is, and helps us find out whether there are some missing rules in the logic.
8. **[click]** Indeed, many prior work has shown completeness for other forms of logics
9. Since these prior works considered first-order languages, and used very different form of ghost state, they cannot be directly reused in Iris.
10. **[click]** In fact, it remains unclear whether Iris-based logics are complete or not.

A Missing Piece in Iris



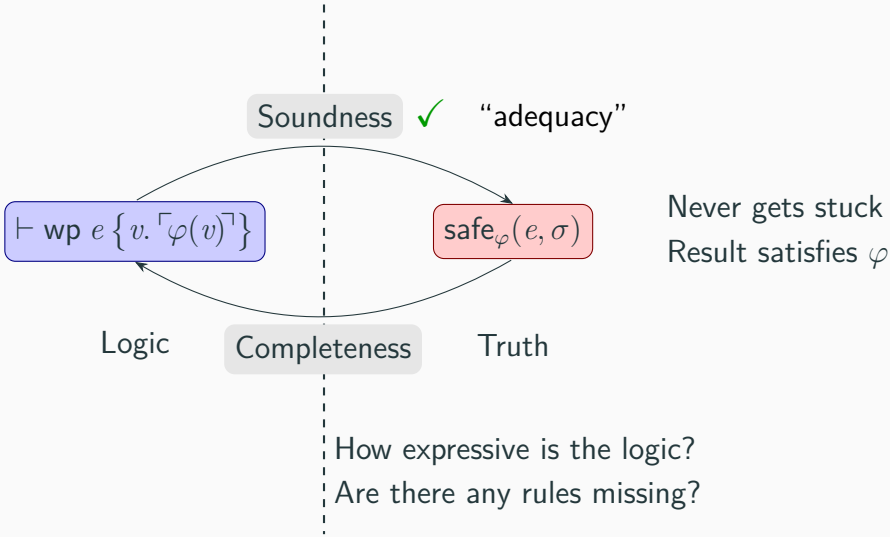
1. Let's think about one question: why do we want a program logic? It turns out that we want use it to show some truths about programs via proving assertions in the logic.
2. **[click]** In Iris, the truth is the safety of a program, which means the evaluation of expression e never gets stuck, and the result satisfies a meta-level predicate φ if it terminates.
3. **[click]** and the assertion in the logic that we want to prove is a wp of e of a pure postcondition
4. **[click]** The first thing we want to make sure is soundness, namely, everything proved in the logic is actually true at meta level
5. This is the bottom line of a correctness logic, and indeed **[click]** every Iris-based program logic comes with a soundness theorem, which is often known as adequacy
6. **[click]** But what about the other direction? Namely, completeness, which ensures every truth is provable.
7. **[click]** The significance of completeness is that it directly answers how expressive the logic is, and helps us find out whether there are some missing rules in the logic.
8. **[click]** Indeed, many prior work has shown completeness for other forms of logics
9. Since these prior works considered first-order languages, and used very different form of ghost state, they cannot be directly reused in Iris.
10. **[click]** In fact, it remains unclear whether Iris-based logics are complete or not.

A Missing Piece in Iris



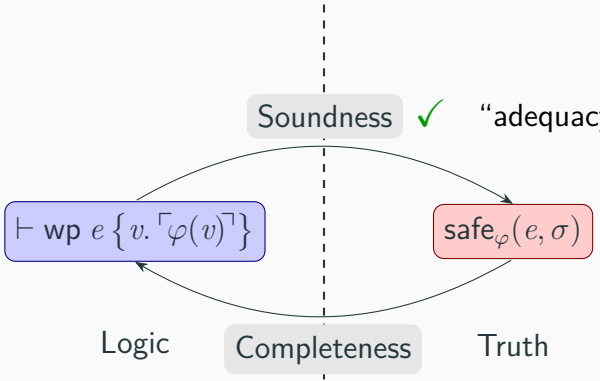
1. Let's think about one question: why do we want a program logic? It turns out that we want use it to show some truths about programs via proving assertions in the logic.
2. **[click]** In Iris, the truth is the safety of a program, which means the evaluation of expression e never gets stuck, and the result satisfies a meta-level predicate φ if it terminates.
3. **[click]** and the assertion in the logic that we want to prove is a wp of e of a pure postcondition
4. **[click]** The first thing we want to make sure is soundness, namely, everything proved in the logic is actually true at meta level
5. This is the bottom line of a correctness logic, and indeed **[click]** every Iris-based program logic comes with a soundness theorem, which is often known as adequacy
6. **[click]** But what about the other direction? Namely, completeness, which ensures every truth is provable.
7. **[click]** The significance of completeness is that it directly answers how expressive the logic is, and helps us find out whether there are some missing rules in the logic.
8. **[click]** Indeed, many prior work has shown completeness for other forms of logics
9. Since these prior works considered first-order languages, and used very different form of ghost state, they cannot be directly reused in Iris.
10. **[click]** In fact, it remains unclear whether Iris-based logics are complete or not.

A Missing Piece in Iris



1. Let's think about one question: why do we want a program logic? It turns out that we want use it to show some truths about programs via proving assertions in the logic.
2. **[click]** In Iris, the truth is the safety of a program, which means the evaluation of expression e never gets stuck, and the result satisfies a meta-level predicate φ if it terminates.
3. **[click]** and the assertion in the logic that we want to prove is a wp of e of a pure postcondition
4. **[click]** The first thing we want to make sure is soundness, namely, everything proved in the logic is actually true at meta level
5. This is the bottom line of a correctness logic, and indeed **[click]** every Iris-based program logic comes with a soundness theorem, which is often known as adequacy
6. **[click]** But what about the other direction? Namely, completeness, which ensures every truth is provable.
7. **[click]** The significance of completeness is that it directly answers how expressive the logic is, and helps us find out whether there are some missing rules in the logic.
8. **[click]** Indeed, many prior work has shown completeness for other forms of logics
9. Since these prior works considered first-order languages, and used very different form of ghost state, they cannot be directly reused in Iris.
10. **[click]** In fact, it remains unclear whether Iris-based logics are complete or not.

A Missing Piece in Iris



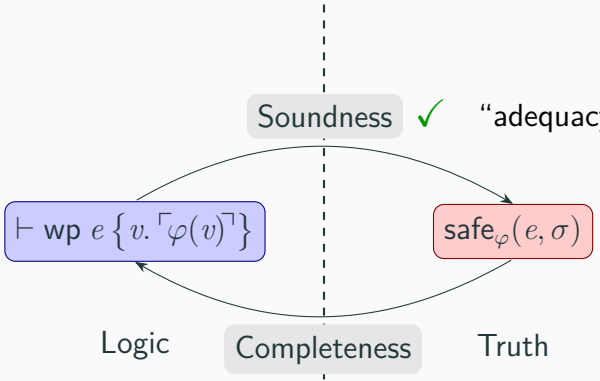
Prior work:

Cook (1978), Owicki (1975),
Jones (1983), Stirling (1988), etc.

How expressive is the logic?
Are there any rules missing?

1. Let's think about one question: why do we want a program logic? It turns out that we want use it to show some truths about programs via proving assertions in the logic.
2. **[click]** In Iris, the truth is the safety of a program, which means the evaluation of expression e never gets stuck, and the result satisfies a meta-level predicate φ if it terminates.
3. **[click]** and the assertion in the logic that we want to prove is a wp of e of a pure postcondition
4. **[click]** The first thing we want to make sure is soundness, namely, everything proved in the logic is actually true at meta level
5. This is the bottom line of a correctness logic, and indeed **[click]** every Iris-based program logic comes with a soundness theorem, which is often known as adequacy
6. **[click]** But what about the other direction? Namely, completeness, which ensures every truth is provable.
7. **[click]** The significance of completeness is that it directly answers how expressive the logic is, and helps us find out whether there are some missing rules in the logic.
8. **[click]** Indeed, many prior work has shown completeness for other forms of logics
9. Since these prior works considered first-order languages, and used very different form of ghost state, they cannot be directly reused in Iris.
10. **[click]** In fact, it remains unclear whether Iris-based logics are complete or not.

A Missing Piece in Iris



Prior work:

Cook (1978), Owicki (1975),
Jones (1983), Stirling (1988), etc.

Iris: ?

How expressive is the logic?

Are there any rules missing?

1. Let's think about one question: why do we want a program logic? It turns out that we want use it to show some truths about programs via proving assertions in the logic.
2. **[click]** In Iris, the truth is the safety of a program, which means the evaluation of expression e never gets stuck, and the result satisfies a meta-level predicate φ if it terminates.
3. **[click]** and the assertion in the logic that we want to prove is a wp of e of a pure postcondition
4. **[click]** The first thing we want to make sure is soundness, namely, everything proved in the logic is actually true at meta level
5. This is the bottom line of a correctness logic, and indeed **[click]** every Iris-based program logic comes with a soundness theorem, which is often known as adequacy
6. **[click]** But what about the other direction? Namely, completeness, which ensures every truth is provable.
7. **[click]** The significance of completeness is that it directly answers how expressive the logic is, and helps us find out whether there are some missing rules in the logic.
8. **[click]** Indeed, many prior work has shown completeness for other forms of logics
9. Since these prior works considered first-order languages, and used very different form of ghost state, they cannot be directly reused in Iris.
10. **[click]** In fact, it remains unclear whether Iris-based logics are complete or not.

Our Contributions

Part 1: completeness of whole program specifications

- Generic completeness theorems for any Iris-based program logics for
  - Partial correctness
  - Total correctness
  - Refinement
- Concrete languages:
  - HeapLang, λ_{Rust}, Time credits, Eris
  - **Probably your favorite program logics!**

1. And this work finally adds this missing piece to Iris.
2. In the first part of the talk, I will explain the completeness of whole program specifications
3. We also developed generic completeness theorems for any Iris-based program logics for partial correctness, total correctness, and refinement.
4. And we have used this framework to prove the completeness of heaplang, lambda rust, time credits, and Eris.
5. We believe our theorems are generic enough, so it probably also works for your favorite program logics!
6. **[click]** In the second part of this talk, I will explain our on going work on the completeness for library specifications.
7. More specifically, the work focuses on the completeness of logically atomic triples, the specifications for concurrent data structures.

Our Contributions

Part 1: completeness of whole program specifications

- Generic completeness theorems for any Iris-based program logics for
  - Partial correctness
  - Total correctness
  - Refinement
- Concrete languages:
  - HeapLang, λ_{Rust}, Time credits, Eris
  - **Probably your favorite program logics!**

Part 2 (WIP): completeness of library specifications

- Completeness of logically atomic triples in Iris

1. And this work finally adds this missing piece to Iris.
2. In the first part of the talk, I will explain the completeness of whole program specifications
3. We also developed generic completeness theorems for any Iris-based program logics for partial correctness, total correctness, and refinement.
4. And we have used this framework to prove the completeness of heaplang, lambda rust, time credits, and Eris.
5. We believe our theorems are generic enough, so it probably also works for your favorite program logics!
6. **[click]** In the second part of this talk, I will explain our on going work on the completeness for library specifications.
7. More specifically, the work focuses on the completeness of logically atomic triples, the specifications for concurrent data structures.

Whole Program Completeness

Johannes Hostert **Zichen Zhang** Puming Liu

Simon Gregersen Ralf Jung Joseph Tassarotti

Warm-up: Sequential Completeness

Language: $\rightarrow_{\mathsf{pure}} \mid \texttt{ref}\ e \mid !\, e \mid e_1 \leftarrow e_2$

HeapLang, but without `fork`, `free`, and prophecy variables

1. Before showing the most generic theorem, let's warm-up by considering a sequential program language with pure steps, allocate, load, and store. This is essentially a subset of HeapLang with no fork, free or prophecy variables. Our goal is to prove this completeness theorem.
2. **[click]** our goal is to prove this completeness statement
3. The idea is to do Löb induction, which requires us to generalize **[click]** the theorem for any intermediate state, as long **[click]** as we have the ownership to the state
4. **[click]** The induction hypothesis look like this
5. **[click]** To make progress, we case distinction on whether e is a value
6. **[click]** if e is already a value, then, by safety, we know $\varphi(v)$ holds, and we can finish the proof by wp-value.
7. **[click]** if e is not a value, then by safety, e must be reducible. We then need to do case analysis on the first step.

Warm-up: Sequential Completeness

Language: $\to_{\mathsf{pure}} \mid \mathbf{ref}\ e \mid !\, e \mid e_1 \leftarrow e_2$

HeapLang, but without **`fork`**, **`free`**, and prophecy variables

$$\forall e \quad .\ \mathsf{safe}_{\varphi}(e, \varnothing) \implies \qquad \vdash \mathsf{wp}\ e \left\{ v. \ulcorner \varphi(v) \urcorner \right\}$$

1. Before showing the most generic theorem, let's warm-up by considering a sequential program language with pure steps, allocate, load, and store. This is essentially a subset of HeapLang with no fork, free or prophecy variables. Our goal is to prove this completeness theorem.
2. **[click]** our goal is to prove this completeness statement
3. The idea is to do Löb induction, which requires us to generalize **[click]** the theorem for any intermediate state, as long **[click]** as we have the ownership to the state
4. **[click]** The induction hypothesis look like this
5. **[click]** To make progress, we case distinction on whether e is a value
6. **[click]** if e is already a value, then, by safety, we know $\varphi(v)$ holds, and we can finish the proof by wp-value.
7. **[click]** if e is not a value, then by safety, e must be reducible. We then need to do case analysis on the first step.

Warm-up: Sequential Completeness

Language: $\rightarrow_{\mathsf{pure}} \mid \mathbf{ref}\ e \mid !\, e \mid e_1 \leftarrow e_2$

HeapLang, but without **`fork`**, **`free`**, and prophecy variables

$$\forall e \quad . \ \mathsf{safe}_{\varphi}(e, \varnothing) \implies \qquad \vdash \mathsf{wp}\ e \left\{ v. \ulcorner \varphi(v) \urcorner \right\}$$

Löb induction

1. Before showing the most generic theorem, let's warm-up by considering a sequential program language with pure steps, allocate, load, and store. This is essentially a subset of HeapLang with no fork, free or prophecy variables. Our goal is to prove this completeness theorem.
2. **[click]** our goal is to prove this completeness statement
3. The idea is to do Löb induction, which requires us to generalize **[click]** the theorem for any intermediate state, as long **[click]** as we have the ownership to the state
4. **[click]** The induction hypothesis look like this
5. **[click]** To make progress, we case distinction on whether e is a value
6. **[click]** if e is already a value, then, by safety, we know $\varphi(v)$ holds, and we can finish the proof by wp-value.
7. **[click]** if e is not a value, then by safety, e must be reducible. We then need to do case analysis on the first step.

Warm-up: Sequential Completeness

Language: $\rightarrow_{\mathsf{pure}} \mid \texttt{ref}\ e \mid !\, e \mid e_1 \leftarrow e_2$

HeapLang, but without **`fork`**, **`free`**, and prophecy variables

$$\forall e, \sigma.\, \mathsf{safe}_{\varphi}(e, \sigma) \implies \qquad \vdash \mathsf{wp}\ e \left\{ v. \ulcorner \varphi(v) \urcorner \right\}$$

Löb induction

1. Before showing the most generic theorem, let's warm-up by considering a sequential program language with pure steps, allocate, load, and store. This is essentially a subset of HeapLang with no fork, free or prophecy variables. Our goal is to prove this completeness theorem.
2. **[click]** our goal is to prove this completeness statement
3. The idea is to do Löb induction, which requires us to generalize **[click]** the theorem for any intermediate state, as long **[click]** as we have the ownership to the state
4. **[click]** The induction hypothesis look like this
5. **[click]** To make progress, we case distinction on whether e is a value
6. **[click]** if e is already a value, then, by safety, we know $\varphi(v)$ holds, and we can finish the proof by wp-value.
7. **[click]** if e is not a value, then by safety, e must be reducible. We then need to do case analysis on the first step.

Warm-up: Sequential Completeness

Language: $\rightarrow_{\mathsf{pure}} \mid \texttt{ref}\ e \mid !\, e \mid e_1 \leftarrow e_2$

HeapLang, but without `fork`, `free`, and prophecy variables

$$\forall e, \sigma.\, \mathsf{safe}_{\varphi}(e, \sigma) \implies \mathsf{S}_{\circ}(\sigma) \vdash \mathsf{wp}\ e \left\{ v.\, \ulcorner \varphi(v) \urcorner \right\}$$

$$\mathsf{S}_{\circ}(\sigma) \triangleq \mathop{\Large *}_{(\ell \leftarrow v) \in \sigma} (\ell \mapsto v) \qquad \text{(the ownership of } \sigma\text{)}$$

Löb induction

1. Before showing the most generic theorem, let's warm-up by considering a sequential program language with pure steps, allocate, load, and store. This is essentially a subset of HeapLang with no fork, free or prophecy variables. Our goal is to prove this completeness theorem.
2. **[click]** our goal is to prove this completeness statement
3. The idea is to do Löb induction, which requires us to generalize **[click]** the theorem for any intermediate state, as long **[click]** as we have the ownership to the state
4. **[click]** The induction hypothesis look like this
5. **[click]** To make progress, we case distinction on whether e is a value
6. **[click]** if e is already a value, then, by safety, we know $\varphi(v)$ holds, and we can finish the proof by wp-value.
7. **[click]** if e is not a value, then by safety, e must be reducible. We then need to do case analysis on the first step.

Warm-up: Sequential Completeness

Language: $\to_{\mathsf{pure}} \mid \texttt{ref}\ e \mid !\, e \mid e_1 \leftarrow e_2$

HeapLang, but without `fork`, `free`, and prophecy variables

$$\forall e, \sigma.\, \mathsf{safe}_\varphi(e, \sigma) \implies \mathsf{S}_\circ(\sigma) \vdash \mathsf{wp}\ e \left\{ v.\, \ulcorner \varphi(v) \urcorner \right\}$$

$$\mathsf{S}_\circ(\sigma) \triangleq \mathop{\Large *}_{(\ell \leftarrow v) \in \sigma} (\ell \mapsto v) \qquad \text{(the ownership of } \sigma\text{)}$$

Löb induction $\quad \triangleright \left(\forall e, \sigma.\, \ulcorner \mathsf{safe}_\varphi(e, \sigma) \urcorner \mathrel{-\!\!*} \mathsf{S}_\circ(\sigma) \mathrel{-\!\!*} \mathsf{wp}\ e \left\{ v.\, \ulcorner \varphi(v) \urcorner \right\} \right)$

1. Before showing the most generic theorem, let's warm-up by considering a sequential program language with pure steps, allocate, load, and store. This is essentially a subset of HeapLang with no fork, free or prophecy variables. Our goal is to prove this completeness theorem.
2. **[click]** our goal is to prove this completeness statement
3. The idea is to do Löb induction, which requires us to generalize **[click]** the theorem for any intermediate state, as long **[click]** as we have the ownership to the state
4. **[click]** The induction hypothesis look like this
5. **[click]** To make progress, we case distinction on whether e is a value
6. **[click]** if e is already a value, then, by safety, we know $\varphi(v)$ holds, and we can finish the proof by wp-value.
7. **[click]** if e is not a value, then by safety, e must be reducible. We then need to do case analysis on the first step.

Warm-up: Sequential Completeness

Language: $\rightarrow_{\mathsf{pure}} \mid \mathbf{ref}\ e \mid !\, e \mid e_1 \leftarrow e_2$

HeapLang, but without **fork**, **free**, and prophecy variables

$$\forall e, \sigma.\, \mathsf{safe}_\varphi(e, \sigma) \implies \mathsf{S}_\circ(\sigma) \vdash \mathsf{wp}\ e \left\{ v.\, \ulcorner \varphi(v) \urcorner \right\}$$

$$\mathsf{S}_\circ(\sigma) \triangleq \mathop{\Large *}_{(\ell \leftarrow v) \in \sigma} (\ell \mapsto v) \qquad \text{(the ownership of } \sigma\text{)}$$

Löb induction $\quad \triangleright \left(\forall e, \sigma.\, \ulcorner \mathsf{safe}_\varphi(e, \sigma) \urcorner \mathrel{-\!\!*} \mathsf{S}_\circ(\sigma) \mathrel{-\!\!*} \mathsf{wp}\ e \left\{ v.\, \ulcorner \varphi(v) \urcorner \right\} \right)$

- If $\mathsf{safe}_\varphi(v, \sigma)$ ($v \in \mathit{Val}$)
- If $\mathsf{safe}_\varphi(e_1, \sigma_1)$ ($e_1 \notin \mathit{Val}$),

1. Before showing the most generic theorem, let's warm-up by considering a sequential program language with pure steps, allocate, load, and store. This is essentially a subset of HeapLang with no fork, free or prophecy variables. Our goal is to prove this completeness theorem.
2. **[click]** our goal is to prove this completeness statement
3. The idea is to do Löb induction, which requires us to generalize **[click]** the theorem for any intermediate state, as long **[click]** as we have the ownership to the state
4. **[click]** The induction hypothesis look like this
5. **[click]** To make progress, we case distinction on whether e is a value
6. **[click]** if e is already a value, then, by safety, we know $\varphi(v)$ holds, and we can finish the proof by wp-value.
7. **[click]** if e is not a value, then by safety, e must be reducible. We then need to do case analysis on the first step.

Warm-up: Sequential Completeness

Language: $\to_{\mathsf{pure}} \mid \mathtt{ref}\ e \mid !\, e \mid e_1 \leftarrow e_2$

HeapLang, but without `fork`, `free`, and prophecy variables

$$\forall e, \sigma.\, \mathsf{safe}_\varphi(e, \sigma) \implies \mathsf{S}_\circ(\sigma) \vdash \mathsf{wp}\ e \left\{ v.\, \ulcorner \varphi(v) \urcorner \right\}$$

$$\mathsf{S}_\circ(\sigma) \triangleq \mathop{\Large *}_{(\ell \leftarrow v) \in \sigma} (\ell \mapsto v) \qquad \text{(the ownership of } \sigma\text{)}$$

Löb induction $\quad \triangleright \left(\forall e, \sigma.\, \ulcorner \mathsf{safe}_\varphi(e, \sigma) \urcorner \mathrel{-\!\!*} \mathsf{S}_\circ(\sigma) \mathrel{-\!\!*} \mathsf{wp}\ e \left\{ v.\, \ulcorner \varphi(v) \urcorner \right\} \right)$

- If $\mathsf{safe}_\varphi(v, \sigma)$ ($v \in \mathit{Val}$) $\implies \ulcorner \varphi(v) \urcorner \vdash \mathsf{wp}\ v \left\{ v.\, \ulcorner \varphi(v) \urcorner \right\}$ (by WpValue)
- If $\mathsf{safe}_\varphi(e_1, \sigma_1)$ ($e_1 \notin \mathit{Val}$),

1. Before showing the most generic theorem, let's warm-up by considering a sequential program language with pure steps, allocate, load, and store. This is essentially a subset of HeapLang with no fork, free or prophecy variables. Our goal is to prove this completeness theorem.
2. **[click]** our goal is to prove this completeness statement
3. The idea is to do Löb induction, which requires us to generalize **[click]** the theorem for any intermediate state, as long **[click]** as we have the ownership to the state
4. **[click]** The induction hypothesis look like this
5. **[click]** To make progress, we case distinction on whether e is a value
6. **[click]** if e is already a value, then, by safety, we know $\varphi(v)$ holds, and we can finish the proof by wp-value.
7. **[click]** if e is not a value, then by safety, e must be reducible. We then need to do case analysis on the first step.

Warm-up: Sequential Completeness

Language: $\to_{\mathsf{pure}} \mid \mathbf{ref}\ e \mid !\, e \mid e_1 \leftarrow e_2$

HeapLang, but without **`fork`**, **`free`**, and prophecy variables

$$\forall e, \sigma.\, \mathsf{safe}_\varphi(e, \sigma) \implies \mathsf{S}_\circ(\sigma) \vdash \mathsf{wp}\ e \left\{ v. \ulcorner \varphi(v) \urcorner \right\}$$

$$\mathsf{S}_\circ(\sigma) \triangleq \mathop{\Large *}_{(\ell \leftarrow v) \in \sigma} (\ell \mapsto v) \qquad \text{(the ownership of } \sigma\text{)}$$

Löb induction $\quad \triangleright \left(\forall e, \sigma. \ulcorner \mathsf{safe}_\varphi(e, \sigma) \urcorner \mathrel{-\!\!*} \mathsf{S}_\circ(\sigma) \mathrel{-\!\!*} \mathsf{wp}\ e \left\{ v. \ulcorner \varphi(v) \urcorner \right\} \right)$

- If $\mathsf{safe}_\varphi(v, \sigma)$ ($v \in \mathit{Val}$) $\implies \ulcorner \varphi(v) \urcorner \vdash \mathsf{wp}\ v \left\{ v. \ulcorner \varphi(v) \urcorner \right\}$ (by WpValue)
- If $\mathsf{safe}_\varphi(e_1, \sigma_1)$ ($e_1 \notin \mathit{Val}$), then e_1 is reducible

case analysis on the first step

1. Before showing the most generic theorem, let's warm-up by considering a sequential program language with pure steps, allocate, load, and store. This is essentially a subset of HeapLang with no fork, free or prophecy variables. Our goal is to prove this completeness theorem.
2. **[click]** our goal is to prove this completeness statement
3. The idea is to do Löb induction, which requires us to generalize **[click]** the theorem for any intermediate state, as long **[click]** as we have the ownership to the state
4. **[click]** The induction hypothesis look like this
5. **[click]** To make progress, we case distinction on whether e is a value
6. **[click]** if e is already a value, then, by safety, we know $\varphi(v)$ holds, and we can finish the proof by wp-value.
7. **[click]** if e is not a value, then by safety, e must be reducible. We then need to do case analysis on the first step.

If $\mathsf{safe}_\varphi(e_1, \sigma_1)$ ($e_1 \notin \mathit{Val}$)

Pure hyp $\quad$ `Hsafe1`: $\mathsf{safe}_\varphi(e_1, \sigma_1)$

Persistent hyp $\quad$ `IH`: $\triangleright \left(\forall e, \sigma.\, \ulcorner \mathsf{safe}_\varphi(e, \sigma) \urcorner \mathrel{-\!\!*} \mathsf{S}_\circ(\sigma) \mathrel{-\!\!*} \mathsf{wp}\ e \left\{ v.\, \ulcorner \varphi(v) \urcorner \right\}\right)$ □

Spatial hyp $\quad$ `H`$\circ$: $\mathsf{S}_\circ(\sigma_1)$

Goal $\quad$ $\mathsf{wp}\quad e_1\ \left\{ v.\, \ulcorner \varphi(v) \urcorner \right\}$ ∗

1. Take a closer look at the non-value case. In Iris proofmode, our goal look like this: Hsafe says the current configuration (e_1, σ_1) is safe, we have the Löb induction hypothesis IH, we own the current state, and we need to prove a wp of e_1
2. **[click]** Because (e_1, σ_1) is safe, **[click]** there must be a prim step from (e_1, σ_1), and the new configuration is also safe **[click]**
3. **[click]** We then unfold the definition of prim step, and **[click]** get a base step under an evaluation context. **[click]**
4. **[click]** we focus on the first base step **[click]** using wp-bind. **[click]**
5. **[click]** Now, we need to do case analysis the first base step.
6. **[click]** This results in four cases: e_1' takes a pure step, e_1' is alloc, e_1' is load, or e_1' store.
7. let's prove these cases one by one

If $\mathsf{safe}_\varphi(e_1, \sigma_1)$ ($e_1 \notin \mathit{Val}$)

Pure hyp — $\texttt{Hsafe1}: \mathsf{safe}_\varphi(e_1, \sigma_1)$

Persistent hyp — $\texttt{IH}: \triangleright \left(\forall e, \sigma. \ulcorner \mathsf{safe}_\varphi(e, \sigma) \urcorner \mathrel{-\!\!*} \mathsf{S}_\circ(\sigma) \mathrel{-\!\!*} \mathsf{wp}\ e \left\{ v. \ulcorner \varphi(v) \urcorner \right\}\right)$

Spatial hyp — $\texttt{H}\circ: \mathsf{S}_\circ(\sigma_1)$

Goal — $\mathsf{wp}\ \ e_1\ \left\{ v. \ulcorner \varphi(v) \urcorner \right\}$

1. Take a closer look at the non-value case. In Iris proofmode, our goal look like this: Hsafe says the current configuration (e_1, σ_1) is safe, we have the Löb induction hypothesis IH, we own the current state, and we need to prove a wp of e_1
2. **[click]** Because (e_1, σ_1) is safe, **[click]** there must be a prim step from (e_1, σ_1), and the new configuration is also safe **[click]**
3. **[click]** We then unfold the definition of prim step, and **[click]** get a base step under an evaluation context. **[click]**
4. **[click]** we focus on the first base step **[click]** using wp-bind. **[click]**
5. **[click]** Now, we need to do case analysis the first base step.
6. **[click]** This results in four cases: e_1' takes a pure step, e_1' is alloc, e_1' is load, or e_1' store.
7. let's prove these cases one by one

If $\mathsf{safe}_\varphi(e_1, \sigma_1)$ ($e_1 \notin \mathit{Val}$)

Pure hyp — $\texttt{Hsafe1}: \mathsf{safe}_\varphi(e_1, \sigma_1)$

$\texttt{Hprim}: (e_1, \sigma_1) \rightarrow_{\mathsf{prim}} (e_2, \sigma_2)$

$\texttt{Hsafe2}: \mathsf{safe}_\varphi(e_2, \sigma_2)$

Persistent hyp — $\texttt{IH}: \triangleright \left(\forall e, \sigma. \ulcorner \mathsf{safe}_\varphi(e, \sigma) \urcorner \mathrel{-\!\!*} \mathsf{S}_\circ(\sigma) \mathrel{-\!\!*} \mathsf{wp}\ e \left\{ v. \ulcorner \varphi(v) \urcorner \right\}\right)$

---□

Spatial hyp — $\texttt{H}\circ: \mathsf{S}_\circ(\sigma_1)$

---∗

Goal — $\mathsf{wp}\quad e_1\ \left\{ v. \ulcorner \varphi(v) \urcorner \right\}$

1. Take a closer look at the non-value case. In Iris proofmode, our goal look like this: Hsafe says the current configuration (e_1, σ_1) is safe, we have the Löb induction hypothesis IH, we own the current state, and we need to prove a wp of e_1
2. **[click]** Because (e_1, σ_1) is safe, **[click]** there must be a prim step from (e_1, σ_1), and the new configuration is also safe **[click]**
3. **[click]** We then unfold the definition of prim step, and **[click]** get a base step under an evaluation context. **[click]**
4. **[click]** we focus on the first base step **[click]** using wp-bind. **[click]**
5. **[click]** Now, we need to do case analysis the first base step.
6. **[click]** This results in four cases: e_1' takes a pure step, e_1' is alloc, e_1' is load, or e_1' store.
7. let's prove these cases one by one

If $\mathsf{safe}_\varphi(e_1, \sigma_1)$ ($e_1 \notin \mathit{Val}$)

Pure hyp — $\texttt{Hsafe1}: \mathsf{safe}_\varphi(e_1, \sigma_1)$

$\texttt{Hprim}: (e_1, \sigma_1) \rightarrow_{\mathsf{prim}} (e_2, \sigma_2)$

$\texttt{Hsafe2}: \mathsf{safe}_\varphi(e_2, \sigma_2)$

Persistent hyp — $\texttt{IH}: \triangleright \left(\forall e, \sigma. \ulcorner \mathsf{safe}_\varphi(e, \sigma) \urcorner \mathrel{-\!\!*} \mathsf{S}_\circ(\sigma) \mathrel{-\!\!*} \mathsf{wp}\ e \left\{ v. \ulcorner \varphi(v) \urcorner \right\}\right)$

---□

Spatial hyp — $\texttt{H}\circ: \mathsf{S}_\circ(\sigma_1)$

---✱

Goal — $\mathsf{wp}\quad e_1\ \left\{ v. \ulcorner \varphi(v) \urcorner \right\}$

1. Take a closer look at the non-value case. In Iris proofmode, our goal look like this: Hsafe says the current configuration (e_1, σ_1) is safe, we have the Löb induction hypothesis IH, we own the current state, and we need to prove a wp of e_1
2. **[click]** Because (e_1, σ_1) is safe, **[click]** there must be a prim step from (e_1, σ_1), and the new configuration is also safe **[click]**
3. **[click]** We then unfold the definition of prim step, and **[click]** get a base step under an evaluation context. **[click]**
4. **[click]** we focus on the first base step **[click]** using wp-bind. **[click]**
5. **[click]** Now, we need to do case analysis the first base step.
6. **[click]** This results in four cases: e_1' takes a pure step, e_1' is alloc, e_1' is load, or e_1' store.
7. let's prove these cases one by one

If $\mathsf{safe}_\varphi(e_1, \sigma_1)$ ($e_1 \notin \mathit{Val}$)

Pure hyp

$\texttt{Hsafe1}: \mathsf{safe}_\varphi(e_1, \sigma_1)$

unfold prim-step $e_1 = K[e_1'] \land e_2 = K[e_2']$

$\texttt{Hprim}: (e_1, \sigma_1) \to_{\mathsf{prim}} (e_2, \sigma_2)$

$\texttt{Hsafe2}: \mathsf{safe}_\varphi(e_2, \sigma_2)$

Persistent hyp

$\texttt{IH}: \triangleright \left(\forall e, \sigma. \ulcorner \mathsf{safe}_\varphi(e, \sigma) \urcorner \mathrel{-\!\!*} \mathsf{S}_\circ(\sigma) \mathrel{-\!\!*} \mathsf{wp}\ e \left\{ v. \ulcorner \varphi(v) \urcorner \right\}\right)$

---□

Spatial hyp

$\texttt{H}\circ: \mathsf{S}_\circ(\sigma_1)$

---∗

Goal

$\mathsf{wp}\quad e_1 \left\{ v. \ulcorner \varphi(v) \urcorner \right\}$

1. Take a closer look at the non-value case. In Iris proofmode, our goal look like this: Hsafe says the current configuration (e_1, σ_1) is safe, we have the Löb induction hypothesis IH, we own the current state, and we need to prove a wp of e_1
2. **[click]** Because (e_1, σ_1) is safe, **[click]** there must be a prim step from (e_1, σ_1), and the new configuration is also safe **[click]**
3. **[click]** We then unfold the definition of prim step, and **[click]** get a base step under an evaluation context. **[click]**
4. **[click]** we focus on the first base step **[click]** using wp-bind. **[click]**
5. **[click]** Now, we need to do case analysis the first base step.
6. **[click]** This results in four cases: e_1' takes a pure step, e_1' is alloc, e_1' is load, or e_1' store.
7. let's prove these cases one by one

If $\mathsf{safe}_\varphi(e_1, \sigma_1)$ ($e_1 \notin \mathit{Val}$)

Pure hyp

$\texttt{Hsafe1}: \mathsf{safe}_\varphi(K[e_1'], \sigma_1)$ unfold prim-step $e_1 = K[e_1'] \land e_2 = K[e_2']$

$\texttt{Hbase}: (e_1', \sigma_1) \to_{\mathsf{base}} (e_2', \sigma_2)$

$\texttt{Hsafe2}: \mathsf{safe}_\varphi(K[e_2'], \sigma_2)$

Persistent hyp $\texttt{IH}: \triangleright \left(\forall e, \sigma. \ulcorner \mathsf{safe}_\varphi(e, \sigma) \urcorner \mathrel{-\!\!*} \mathsf{S}_\circ(\sigma) \mathrel{-\!\!*} \mathsf{wp}\ e \left\{ v. \ulcorner \varphi(v) \urcorner \right\}\right)$

Spatial hyp $\texttt{H}\circ: \mathsf{S}_\circ(\sigma_1)$

Goal $\mathsf{wp}\ K[e_1'] \left\{ v. \ulcorner \varphi(v) \urcorner \right\}$

1. Take a closer look at the non-value case. In Iris proofmode, our goal look like this: Hsafe says the current configuration (e_1, σ_1) is safe, we have the Löb induction hypothesis IH, we own the current state, and we need to prove a wp of e_1
2. **[click]** Because (e_1, σ_1) is safe, **[click]** there must be a prim step from (e_1, σ_1), and the new configuration is also safe **[click]**
3. **[click]** We then unfold the definition of prim step, and **[click]** get a base step under an evaluation context. **[click]**
4. **[click]** we focus on the first base step **[click]** using wp-bind. **[click]**
5. **[click]** Now, we need to do case analysis the first base step.
6. **[click]** This results in four cases: e_1' takes a pure step, e_1' is alloc, e_1' is load, or e_1' store.
7. let's prove these cases one by one

If $\mathsf{safe}_\varphi(e_1, \sigma_1)$ ($e_1 \notin \mathit{Val}$)

Pure hyp

$\texttt{Hsafe1}: \mathsf{safe}_\varphi(K[e_1'], \sigma_1)$

$\texttt{Hbase}: (e_1', \sigma_1) \rightarrow_{\mathsf{base}} (e_2', \sigma_2)$

$\texttt{Hsafe2}: \mathsf{safe}_\varphi(K[e_2'], \sigma_2)$

Persistent hyp

$\texttt{IH}: \triangleright \left(\forall e, \sigma. \ulcorner \mathsf{safe}_\varphi(e, \sigma) \urcorner \mathrel{-\!\!*} \mathsf{S}_\circ(\sigma) \mathrel{-\!\!*} \mathsf{wp}\ e \left\{ v. \ulcorner \varphi(v) \urcorner \right\}\right)$

---□

Spatial hyp

$\texttt{H}\circ: \mathsf{S}_\circ(\sigma_1)$

---∗

Goal

$\mathsf{wp}\ K[e_1'] \left\{ v. \ulcorner \varphi(v) \urcorner \right\}$

1. Take a closer look at the non-value case. In Iris proofmode, our goal look like this: Hsafe says the current configuration (e_1, σ_1) is safe, we have the Löb induction hypothesis IH, we own the current state, and we need to prove a wp of e_1
2. **[click]** Because (e_1, σ_1) is safe, **[click]** there must be a prim step from (e_1, σ_1), and the new configuration is also safe **[click]**
3. **[click]** We then unfold the definition of prim step, and **[click]** get a base step under an evaluation context. **[click]**
4. **[click]** we focus on the first base step **[click]** using wp-bind. **[click]**
5. **[click]** Now, we need to do case analysis the first base step.
6. **[click]** This results in four cases: e_1' takes a pure step, e_1' is alloc, e_1' is load, or e_1' store.
7. let's prove these cases one by one

If $\mathsf{safe}_\varphi(e_1, \sigma_1)$ ($e_1 \notin \mathit{Val}$)

Pure hyp

$\texttt{Hsafe1}: \mathsf{safe}_\varphi(K[e_1'], \sigma_1)$

$\texttt{Hbase}: (e_1', \sigma_1) \rightarrow_{\mathsf{base}} (e_2', \sigma_2)$

$\texttt{Hsafe2}: \mathsf{safe}_\varphi(K[e_2'], \sigma_2)$

Persistent hyp

$\texttt{IH}: \triangleright \left(\forall e, \sigma. \ulcorner \mathsf{safe}_\varphi(e, \sigma) \urcorner \mathrel{-\!\!*} \mathsf{S}_\circ(\sigma) \mathrel{-\!\!*} \mathsf{wp}\ e \left\{ v. \ulcorner \varphi(v) \urcorner \right\}\right)$

Spatial hyp

$\texttt{H}\circ: \mathsf{S}_\circ(\sigma_1)$

Goal

$\mathsf{wp}\ K[e_1'] \left\{ v. \ulcorner \varphi(v) \urcorner \right\}$ by WpBind

1. Take a closer look at the non-value case. In Iris proofmode, our goal look like this: Hsafe says the current configuration (e_1, σ_1) is safe, we have the Löb induction hypothesis IH, we own the current state, and we need to prove a wp of e_1
2. **[click]** Because (e_1, σ_1) is safe, **[click]** there must be a prim step from (e_1, σ_1), and the new configuration is also safe **[click]**
3. **[click]** We then unfold the definition of prim step, and **[click]** get a base step under an evaluation context. **[click]**
4. **[click]** we focus on the first base step **[click]** using wp-bind. **[click]**
5. **[click]** Now, we need to do case analysis the first base step.
6. **[click]** This results in four cases: e_1' takes a pure step, e_1' is alloc, e_1' is load, or e_1' store.
7. let's prove these cases one by one

If $\mathsf{safe}_\varphi(e_1, \sigma_1)$ ($e_1 \notin \mathit{Val}$)

Pure hyp

$\texttt{Hsafe1}: \mathsf{safe}_\varphi(K[e_1'], \sigma_1)$

$\texttt{Hbase}: (e_1', \sigma_1) \rightarrow_{\mathsf{base}} (e_2', \sigma_2)$

$\texttt{Hsafe2}: \mathsf{safe}_\varphi(K[e_2'], \sigma_2)$

Persistent hyp

$\texttt{IH}: \triangleright \left(\forall e, \sigma. \ulcorner \mathsf{safe}_\varphi(e, \sigma) \urcorner \mathrel{-\!\!*} \mathsf{S}_\circ(\sigma) \mathrel{-\!\!*} \mathsf{wp}\ e \left\{ v. \ulcorner \varphi(v) \urcorner \right\}\right)$

Spatial hyp

$\texttt{H}\circ: \mathsf{S}_\circ(\sigma_1)$

Goal

$\mathsf{wp}\ e_1' \left\{ w. \mathsf{wp}\ K[w] \left\{ v. \ulcorner \varphi(v) \urcorner \right\} \right\}$ by WpBind

1. Take a closer look at the non-value case. In Iris proofmode, our goal look like this: Hsafe says the current configuration (e_1, σ_1) is safe, we have the Löb induction hypothesis IH, we own the current state, and we need to prove a wp of e_1
2. **[click]** Because (e_1, σ_1) is safe, **[click]** there must be a prim step from (e_1, σ_1), and the new configuration is also safe **[click]**
3. **[click]** We then unfold the definition of prim step, and **[click]** get a base step under an evaluation context. **[click]**
4. **[click]** we focus on the first base step **[click]** using wp-bind. **[click]**
5. **[click]** Now, we need to do case analysis the first base step.
6. **[click]** This results in four cases: e_1' takes a pure step, e_1' is alloc, e_1' is load, or e_1' store.
7. let's prove these cases one by one

If $\mathsf{safe}_\varphi(e_1, \sigma_1)$ ($e_1 \notin \mathit{Val}$)

Pure hyp

$\texttt{Hsafe1}: \mathsf{safe}_\varphi(K[e_1'], \sigma_1)$

$\texttt{Hbase}: (e_1', \sigma_1) \rightarrow_{\mathsf{base}} (e_2', \sigma_2)$

$\texttt{Hsafe2}: \mathsf{safe}_\varphi(K[e_2'], \sigma_2)$

Persistent hyp

$\texttt{IH}: \triangleright \left(\forall e, \sigma. \ulcorner \mathsf{safe}_\varphi(e, \sigma) \urcorner \mathrel{-\!\!*} \mathsf{S}_\circ(\sigma) \mathrel{-\!\!*} \mathsf{wp}\ e \left\{ v. \ulcorner \varphi(v) \urcorner \right\}\right)$

Spatial hyp

$\texttt{H}\circ: \mathsf{S}_\circ(\sigma_1)$

Goal

$\mathsf{wp}\ e_1' \left\{ w. \mathsf{wp}\ K[w] \left\{ v. \ulcorner \varphi(v) \urcorner \right\} \right\}$

1. Take a closer look at the non-value case. In Iris proofmode, our goal look like this: Hsafe says the current configuration (e_1, σ_1) is safe, we have the Löb induction hypothesis IH, we own the current state, and we need to prove a wp of e_1
2. **[click]** Because (e_1, σ_1) is safe, **[click]** there must be a prim step from (e_1, σ_1), and the new configuration is also safe **[click]**
3. **[click]** We then unfold the definition of prim step, and **[click]** get a base step under an evaluation context. **[click]**
4. **[click]** we focus on the first base step **[click]** using wp-bind. **[click]**
5. **[click]** Now, we need to do case analysis the first base step.
6. **[click]** This results in four cases: e_1' takes a pure step, e_1' is alloc, e_1' is load, or e_1' store.
7. let's prove these cases one by one

If $\mathsf{safe}_\varphi(e_1, \sigma_1)$ ($e_1 \notin \mathit{Val}$)

Pure hyp

$\texttt{Hsafe1}: \mathsf{safe}_\varphi(K[e_1'], \sigma_1)$ case analysis

$\texttt{Hbase}: (e_1', \sigma_1) \rightarrow_{\mathsf{base}} (e_2', \sigma_2)$

$\texttt{Hsafe2}: \mathsf{safe}_\varphi(K[e_2'], \sigma_2)$

Persistent hyp $\quad \texttt{IH}: \triangleright \left(\forall e, \sigma. \ulcorner \mathsf{safe}_\varphi(e, \sigma) \urcorner \mathrel{-\!\!*} \mathsf{S}_\circ(\sigma) \mathrel{-\!\!*} \mathsf{wp}\ e \left\{ v. \ulcorner \varphi(v) \urcorner \right\}\right)$

Spatial hyp $\quad \texttt{H}\circ: \mathsf{S}_\circ(\sigma_1)$

Goal $\quad \mathsf{wp}\ e_1' \left\{ w. \mathsf{wp}\ K[w] \left\{ v. \ulcorner \varphi(v) \urcorner \right\} \right\}$

1. Take a closer look at the non-value case. In Iris proofmode, our goal look like this: Hsafe says the current configuration (e_1, σ_1) is safe, we have the Löb induction hypothesis IH, we own the current state, and we need to prove a wp of e_1
2. **[click]** Because (e_1, σ_1) is safe, **[click]** there must be a prim step from (e_1, σ_1), and the new configuration is also safe **[click]**
3. **[click]** We then unfold the definition of prim step, and **[click]** get a base step under an evaluation context. **[click]**
4. **[click]** we focus on the first base step **[click]** using wp-bind. **[click]**
5. **[click]** Now, we need to do case analysis the first base step.
6. **[click]** This results in four cases: e_1' takes a pure step, e_1' is alloc, e_1' is load, or e_1' store.
7. let's prove these cases one by one

If $\mathsf{safe}_\varphi(e_1, \sigma_1)$ ($e_1 \notin \mathit{Val}$)

Pure hyp

$\texttt{Hsafe1}: \mathsf{safe}_\varphi(K[e_1'], \sigma_1)$ case analysis

$\texttt{Hbase}: (e_1', \sigma_1) \rightarrow_{\mathsf{base}} (e_2', \sigma_2)$

$\texttt{Hsafe2}: \mathsf{safe}_\varphi(K[e_2'], \sigma_2)$

Persiste

Spatial

- $\square\ e_1' \rightarrow_{\mathsf{pure}} e_2'$
- $\square\ \sigma_1(\ell) = v \land (!\ell, \sigma_1) \rightarrow_{\mathsf{base}} (v, \sigma_1)$
- $\square\ \sigma_1(\ell) = v \land (\ell \leftarrow w, \sigma_1) \rightarrow_{\mathsf{base}} ((), \sigma_1[\ell \leftarrow w])$
- $\square\ \ell_0 \notin \sigma_1 \land (\texttt{ref}\ v, \sigma_1) \rightarrow_{\mathsf{base}} (\ell_0, \sigma_1[\ell_0 \leftarrow v])$

Goal

1. Take a closer look at the non-value case. In Iris proofmode, our goal look like this: Hsafe says the current configuration (e_1, σ_1) is safe, we have the Löb induction hypothesis IH, we own the current state, and we need to prove a wp of e_1
2. **[click]** Because (e_1, σ_1) is safe, **[click]** there must be a prim step from (e_1, σ_1), and the new configuration is also safe **[click]**
3. **[click]** We then unfold the definition of prim step, and **[click]** get a base step under an evaluation context. **[click]**
4. **[click]** we focus on the first base step **[click]** using wp-bind. **[click]**
5. **[click]** Now, we need to do case analysis the first base step.
6. **[click]** This results in four cases: e_1' takes a pure step, e_1' is alloc, e_1' is load, or e_1' store.
7. let's prove these cases one by one

If e_1' takes a pure step

Pure hyp

$\texttt{Hsafe1}: \mathsf{safe}_\varphi(K[e_1'], \sigma_1)$

$\texttt{Hpure}: e_1' \to_{\mathsf{pure}} e_2'$

$\texttt{Hsafe2}: \mathsf{safe}_\varphi(K[e_2'], \sigma_1)$

Persistent hyp

$\texttt{IH}: \triangleright \left(\forall e, \sigma. \ulcorner \mathsf{safe}_\varphi(e, \sigma) \urcorner \mathrel{-\!\!*} \mathsf{S}_\circ(\sigma) \mathrel{-\!\!*} \mathsf{wp}\ e \left\{ v. \ulcorner \varphi(v) \urcorner \right\}\right)$

---□

Spatial hyp

$\texttt{H}\circ: \mathsf{S}_\circ(\sigma_1)$

---∗

Goal

$\mathsf{wp}\ e_1' \left\{ w. \mathsf{wp}\ K[w] \left\{ v. \ulcorner \varphi(v) \urcorner \right\} \right\}$

1. First, if e_1' takes a pure step, then **[click]** we can reduce the goal to **[click]** wp e_2' by wp-pure. **[click]**
2. **[click]** and strip **[click]** the later **[click]**
3. **[click]** we then use the inverse direction of wp-bind to **[click]** restore the goal to a form that matches the conclusion of the induction hypothesis **[click]**
4. **[click]** we can now apply the induction hypothesis, **[click]** which yields two subgoals
5. **[click]** and they can be solved by framing
6. **[click]** this concludes the pure step case

If e_1' takes a pure step

Pure hyp

$\texttt{Hsafe1}: \mathsf{safe}_\varphi(K[e_1'], \sigma_1)$

$\texttt{Hpure}: e_1' \to_{\mathsf{pure}} e_2'$

$\texttt{Hsafe2}: \mathsf{safe}_\varphi(K[e_2'], \sigma_1)$

Persistent hyp

$\texttt{IH}: \triangleright \left(\forall e, \sigma. \ulcorner \mathsf{safe}_\varphi(e, \sigma) \urcorner \mathrel{-\!\!*} \mathsf{S}_\circ(\sigma) \mathrel{-\!\!*} \mathsf{wp}\ e \left\{ v. \ulcorner \varphi(v) \urcorner \right\}\right)$

Spatial hyp

$\texttt{H}\circ: \mathsf{S}_\circ(\sigma_1)$

Goal

$\mathsf{wp}\ e_1' \left\{ w. \mathsf{wp}\ K[w] \left\{ v. \ulcorner \varphi(v) \urcorner \right\} \right\}$

by WpPure

1. First, if e_1' takes a pure step, then **[click]** we can reduce the goal to **[click]** wp e_2' by wp-pure. **[click]**
2. **[click]** and strip **[click]** the later **[click]**
3. **[click]** we then use the inverse direction of wp-bind to **[click]** restore the goal to a form that matches the conclusion of the induction hypothesis **[click]**
4. **[click]** we can now apply the induction hypothesis, **[click]** which yields two subgoals
5. **[click]** and they can be solved by framing
6. **[click]** this concludes the pure step case

If e_1' takes a pure step

Pure hyp

$\texttt{Hsafe1}: \mathsf{safe}_\varphi(K[e_1'], \sigma_1)$

$\texttt{Hpure}: e_1' \to_{\mathsf{pure}} e_2'$

$\texttt{Hsafe2}: \mathsf{safe}_\varphi(K[e_2'], \sigma_1)$

Persistent hyp

$\texttt{IH}: \triangleright \left(\forall e, \sigma. \ulcorner \mathsf{safe}_\varphi(e, \sigma) \urcorner \mathrel{-\!\!*} \mathsf{S}_\circ(\sigma) \mathrel{-\!\!*} \mathsf{wp}\ e \left\{ v. \ulcorner \varphi(v) \urcorner \right\}\right)$

---□

Spatial hyp

$\texttt{H}\circ: \mathsf{S}_\circ(\sigma_1)$

---∗

Goal

$\triangleright\, \mathsf{wp}\ e_2' \left\{ w.\ \mathsf{wp}\ K[w] \left\{ v. \ulcorner \varphi(v) \urcorner \right\} \right\}$

by WpPure

1. First, if e_1' takes a pure step, then **[click]** we can reduce the goal to **[click]** wp e_2' by wp-pure. **[click]**
2. **[click]** and strip **[click]** the later **[click]**
3. **[click]** we then use the inverse direction of wp-bind to **[click]** restore the goal to a form that matches the conclusion of the induction hypothesis **[click]**
4. **[click]** we can now apply the induction hypothesis, **[click]** which yields two subgoals
5. **[click]** and they can be solved by framing
6. **[click]** this concludes the pure step case

If e_1' takes a pure step

Pure hyp

$\texttt{Hsafe1}: \mathsf{safe}_\varphi(K[e_1'], \sigma_1)$

$\texttt{Hpure}: e_1' \to_{\mathsf{pure}} e_2'$

$\texttt{Hsafe2}: \mathsf{safe}_\varphi(K[e_2'], \sigma_1)$

Persistent hyp

$\texttt{IH}: \triangleright \left(\forall e, \sigma. \ulcorner \mathsf{safe}_\varphi(e, \sigma) \urcorner \mathrel{-\!\!*} \mathsf{S}_\circ(\sigma) \mathrel{-\!\!*} \mathsf{wp}\ e \left\{ v. \ulcorner \varphi(v) \urcorner \right\}\right)$

---□

Spatial hyp

$\texttt{H}\circ: \mathsf{S}_\circ(\sigma_1)$

---∗

Goal

$\triangleright\, \mathsf{wp}\ e_2' \left\{ w.\ \mathsf{wp}\ K[w] \left\{ v. \ulcorner \varphi(v) \urcorner \right\} \right\}$

1. First, if e_1' takes a pure step, then **[click]** we can reduce the goal to **[click]** wp e_2' by wp-pure. **[click]**
2. **[click]** and strip **[click]** the later **[click]**
3. **[click]** we then use the inverse direction of wp-bind to **[click]** restore the goal to a form that matches the conclusion of the induction hypothesis **[click]**
4. **[click]** we can now apply the induction hypothesis, **[click]** which yields two subgoals
5. **[click]** and they can be solved by framing
6. **[click]** this concludes the pure step case

If e_1' takes a pure step

Pure hyp

$\texttt{Hsafe1}: \mathsf{safe}_\varphi(K[e_1'], \sigma_1)$

$\texttt{Hpure}: e_1' \to_{\mathsf{pure}} e_2'$

$\texttt{Hsafe2}: \mathsf{safe}_\varphi(K[e_2'], \sigma_1)$

Persistent hyp

$\texttt{IH}: \triangleright \left(\forall e, \sigma. \ulcorner \mathsf{safe}_\varphi(e, \sigma) \urcorner \mathrel{-\!\!*} \mathsf{S}_\circ(\sigma) \mathrel{-\!\!*} \mathsf{wp}\ e \left\{ v. \ulcorner \varphi(v) \urcorner \right\}\right)$

strip later

Spatial hyp

$\texttt{H}\circ: \mathsf{S}_\circ(\sigma_1)$

Goal

$\triangleright\, \mathsf{wp}\ e_2' \left\{ w.\ \mathsf{wp}\ K[w] \left\{ v. \ulcorner \varphi(v) \urcorner \right\} \right\}$

1. First, if e_1' takes a pure step, then **[click]** we can reduce the goal to **[click]** wp e_2' by wp-pure. **[click]**
2. **[click]** and strip **[click]** the later **[click]**
3. **[click]** we then use the inverse direction of wp-bind to **[click]** restore the goal to a form that matches the conclusion of the induction hypothesis **[click]**
4. **[click]** we can now apply the induction hypothesis, **[click]** which yields two subgoals
5. **[click]** and they can be solved by framing
6. **[click]** this concludes the pure step case

If e_1' takes a pure step

Pure hyp

$\texttt{Hsafe1}: \mathsf{safe}_\varphi(K[e_1'], \sigma_1)$

$\texttt{Hpure}: e_1' \to_{\mathsf{pure}} e_2'$

$\texttt{Hsafe2}: \mathsf{safe}_\varphi(K[e_2'], \sigma_1)$

Persistent hyp

$\texttt{IH}: \quad (\forall e, \sigma. \ulcorner \mathsf{safe}_\varphi(e, \sigma) \urcorner \mathrel{-\!\!*} \mathsf{S}_\circ(\sigma) \mathrel{-\!\!*} \mathsf{wp}\ e \{ v. \ulcorner \varphi(v) \urcorner \})$

strip later □

Spatial hyp

$\texttt{H}\circ: \mathsf{S}_\circ(\sigma_1)$

Goal

$\mathsf{wp}\ e_2' \{ w.\ \mathsf{wp}\ K[w] \{ v. \ulcorner \varphi(v) \urcorner \} \}$ ∗

1. First, if e_1' takes a pure step, then **[click]** we can reduce the goal to **[click]** wp e_2' by wp-pure. **[click]**
2. **[click]** and strip **[click]** the later **[click]**
3. **[click]** we then use the inverse direction of wp-bind to **[click]** restore the goal to a form that matches the conclusion of the induction hypothesis **[click]**
4. **[click]** we can now apply the induction hypothesis, **[click]** which yields two subgoals
5. **[click]** and they can be solved by framing
6. **[click]** this concludes the pure step case

If e_1' takes a pure step

Pure hyp

$\texttt{Hsafe1}: \mathsf{safe}_\varphi(K[e_1'], \sigma_1)$

$\texttt{Hpure}: e_1' \to_{\mathsf{pure}} e_2'$

$\texttt{Hsafe2}: \mathsf{safe}_\varphi(K[e_2'], \sigma_1)$

Persistent hyp

$\texttt{IH}: \quad \left(\forall e, \sigma. \ulcorner \mathsf{safe}_\varphi(e, \sigma) \urcorner \mathrel{-\!\!*} \mathsf{S}_\circ(\sigma) \mathrel{-\!\!*} \mathsf{wp}\ e \left\{ v. \ulcorner \varphi(v) \urcorner \right\}\right)$

---□

Spatial hyp

$\texttt{H}\circ: \mathsf{S}_\circ(\sigma_1)$

---∗

Goal

$\mathsf{wp}\ e_2' \left\{ w.\ \mathsf{wp}\ K[w] \left\{ v. \ulcorner \varphi(v) \urcorner \right\} \right\}$

1. First, if e_1' takes a pure step, then **[click]** we can reduce the goal to **[click]** wp e_2' by wp-pure. **[click]**
2. **[click]** and strip **[click]** the later **[click]**
3. **[click]** we then use the inverse direction of wp-bind to **[click]** restore the goal to a form that matches the conclusion of the induction hypothesis **[click]**
4. **[click]** we can now apply the induction hypothesis, **[click]** which yields two subgoals
5. **[click]** and they can be solved by framing
6. **[click]** this concludes the pure step case

If e_1' takes a pure step

Pure hyp

$\texttt{Hsafe1}: \mathsf{safe}_\varphi(K[e_1'], \sigma_1)$

$\texttt{Hpure}: e_1' \to_{\mathsf{pure}} e_2'$

$\texttt{Hsafe2}: \mathsf{safe}_\varphi(K[e_2'], \sigma_1)$

Persistent hyp

$\texttt{IH}: \quad \left(\forall e, \sigma. \ulcorner \mathsf{safe}_\varphi(e, \sigma) \urcorner \mathrel{-\!\!*} \mathsf{S}_\circ(\sigma) \mathrel{-\!\!*} \mathsf{wp}\ e \left\{ v. \ulcorner \varphi(v) \urcorner \right\}\right)$

Spatial hyp

$\texttt{H}\circ: \mathsf{S}_\circ(\sigma_1)$

Goal

$\mathsf{wp}\ e_2' \left\{ w.\ \mathsf{wp}\ K[w] \left\{ v. \ulcorner \varphi(v) \urcorner \right\} \right\}$

by WpBindInv

1. First, if e_1' takes a pure step, then **[click]** we can reduce the goal to **[click]** wp e_2' by wp-pure. **[click]**
2. **[click]** and strip **[click]** the later **[click]**
3. **[click]** we then use the inverse direction of wp-bind to **[click]** restore the goal to a form that matches the conclusion of the induction hypothesis **[click]**
4. **[click]** we can now apply the induction hypothesis, **[click]** which yields two subgoals
5. **[click]** and they can be solved by framing
6. **[click]** this concludes the pure step case

If e_1' takes a pure step

Pure hyp

$\texttt{Hsafe1}: \mathsf{safe}_\varphi(K[e_1'], \sigma_1)$

$\texttt{Hpure}: e_1' \to_{\mathsf{pure}} e_2'$

$\texttt{Hsafe2}: \mathsf{safe}_\varphi(K[e_2'], \sigma_1)$

Persistent hyp

$\texttt{IH}: \quad \left(\forall e, \sigma. \ulcorner \mathsf{safe}_\varphi(e, \sigma) \urcorner \mathrel{-\!\!*} \mathsf{S}_\circ(\sigma) \mathrel{-\!\!*} \mathsf{wp}\ e \left\{ v. \ulcorner \varphi(v) \urcorner \right\}\right)$

---□

Spatial hyp

$\texttt{H}\circ: \mathsf{S}_\circ(\sigma_1)$

---✱

Goal

$\mathsf{wp}\ K[e_2'] \left\{ v. \ulcorner \varphi(v) \urcorner \right\}$

by WpBindInv

1. First, if e_1' takes a pure step, then **[click]** we can reduce the goal to **[click]** wp e_2' by wp-pure. **[click]**
2. **[click]** and strip **[click]** the later **[click]**
3. **[click]** we then use the inverse direction of wp-bind to **[click]** restore the goal to a form that matches the conclusion of the induction hypothesis **[click]**
4. **[click]** we can now apply the induction hypothesis, **[click]** which yields two subgoals
5. **[click]** and they can be solved by framing
6. **[click]** this concludes the pure step case

If e_1' takes a pure step

Pure hyp

$\texttt{Hsafe1}: \mathsf{safe}_\varphi(K[e_1'], \sigma_1)$

$\texttt{Hpure}: e_1' \to_{\mathsf{pure}} e_2'$

$\texttt{Hsafe2}: \mathsf{safe}_\varphi(K[e_2'], \sigma_1)$

Persistent hyp

$\texttt{IH}: \quad \left(\forall e, \sigma. \ulcorner \mathsf{safe}_\varphi(e, \sigma) \urcorner \mathrel{-\!\!*} \mathsf{S}_\circ(\sigma) \mathrel{-\!\!*} \mathsf{wp}\ e \left\{ v. \ulcorner \varphi(v) \urcorner \right\}\right)$

---□

Spatial hyp

$\texttt{H}\circ: \mathsf{S}_\circ(\sigma_1)$

---∗

Goal

$\mathsf{wp}\ K[e_2'] \left\{ v. \ulcorner \varphi(v) \urcorner \right\}$

1. First, if e_1' takes a pure step, then **[click]** we can reduce the goal to **[click]** wp e_2' by wp-pure. **[click]**
2. **[click]** and strip **[click]** the later **[click]**
3. **[click]** we then use the inverse direction of wp-bind to **[click]** restore the goal to a form that matches the conclusion of the induction hypothesis **[click]**
4. **[click]** we can now apply the induction hypothesis, **[click]** which yields two subgoals
5. **[click]** and they can be solved by framing
6. **[click]** this concludes the pure step case

If e_1' takes a pure step

Pure hyp

$\texttt{Hsafe1}: \mathsf{safe}_\varphi(K[e_1'], \sigma_1)$

$\texttt{Hpure}: e_1' \to_{\mathsf{pure}} e_2'$

$\texttt{Hsafe2}: \mathsf{safe}_\varphi(K[e_2'], \sigma_1)$

Persistent hyp

$\texttt{IH}: \quad \left(\forall e, \sigma. \ulcorner \mathsf{safe}_\varphi(e, \sigma) \urcorner \mathrel{-\!\!*} \mathsf{S}_\circ(\sigma) \mathrel{-\!\!*} \mathsf{wp}\ e \left\{ v. \ulcorner \varphi(v) \urcorner \right\}\right)$ apply

---

Spatial hyp

$\texttt{H}\circ: \mathsf{S}_\circ(\sigma_1)$

Goal

$\mathsf{wp}\ K[e_2'] \left\{ v. \ulcorner \varphi(v) \urcorner \right\}$

1. First, if e_1' takes a pure step, then **[click]** we can reduce the goal to **[click]** wp e_2' by wp-pure. **[click]**
2. **[click]** and strip **[click]** the later **[click]**
3. **[click]** we then use the inverse direction of wp-bind to **[click]** restore the goal to a form that matches the conclusion of the induction hypothesis **[click]**
4. **[click]** we can now apply the induction hypothesis, **[click]** which yields two subgoals
5. **[click]** and they can be solved by framing
6. **[click]** this concludes the pure step case

If e_1' takes a pure step

Pure hyp `Hsafe1`: $\mathsf{safe}_\varphi(K[e_1'], \sigma_1)$

`Hpure`: $e_1' \to_{\mathsf{pure}} e_2'$

`Hsafe2`: $\mathsf{safe}_\varphi(K[e_2'], \sigma_1)$

Persistent hyp `IH`: $\left(\forall e, \sigma.\ \ulcorner \mathsf{safe}_\varphi(e, \sigma) \urcorner \mathrel{-\!\!*} \mathsf{S}_\circ(\sigma) \mathrel{-\!\!*} \mathsf{wp}\ e \left\{ v.\ \ulcorner \varphi(v) \urcorner \right\}\right)$ apply

---

Spatial hyp `Ho`: $\mathsf{S}_\circ(\sigma_1)$

Goals $\ulcorner \mathsf{safe}_\varphi(K[e_2'], \sigma_1) \urcorner$ $\quad$ $\mathsf{S}_\circ(\sigma_1)$

1. First, if e_1' takes a pure step, then **[click]** we can reduce the goal to **[click]** wp e_2' by wp-pure. **[click]**
2. **[click]** and strip **[click]** the later **[click]**
3. **[click]** we then use the inverse direction of wp-bind to **[click]** restore the goal to a form that matches the conclusion of the induction hypothesis **[click]**
4. **[click]** we can now apply the induction hypothesis, **[click]** which yields two subgoals
5. **[click]** and they can be solved by framing
6. **[click]** this concludes the pure step case

If e_1' takes a pure step

Pure hyp

$\texttt{Hsafe1}: \mathsf{safe}_\varphi(K[e_1'], \sigma_1)$

$\texttt{Hpure}: e_1' \to_{\mathsf{pure}} e_2'$

$\texttt{Hsafe2}: \mathsf{safe}_\varphi(K[e_2'], \sigma_1)$ frame

---

Persistent hyp

$\texttt{IH}: \quad \left(\forall e, \sigma. \ulcorner \mathsf{safe}_\varphi(e, \sigma) \urcorner \mathrel{-\!\!*} \mathsf{S}_\circ(\sigma) \mathrel{-\!\!*} \mathsf{wp}\ e \left\{ v. \ulcorner \varphi(v) \urcorner \right\}\right)$

Spatial hyp

$\texttt{H}\circ: \mathsf{S}_\circ(\sigma_1)$ frame

Goals

$\ulcorner \mathsf{safe}_\varphi(K[e_2'], \sigma_1) \urcorner$ $\mathsf{S}_\circ(\sigma_1)$

1. First, if e_1' takes a pure step, then **[click]** we can reduce the goal to **[click]** wp e_2' by wp-pure. **[click]**
2. **[click]** and strip **[click]** the later **[click]**
3. **[click]** we then use the inverse direction of wp-bind to **[click]** restore the goal to a form that matches the conclusion of the induction hypothesis **[click]**
4. **[click]** we can now apply the induction hypothesis, **[click]** which yields two subgoals
5. **[click]** and they can be solved by framing
6. **[click]** this concludes the pure step case

If e_1' takes a pure step

Pure hyp `Hsafe1`: $\mathsf{safe}_\varphi(K[e_1'], \sigma_1)$

- ☑ $e_1' \to_{\mathsf{pure}} e_2'$
- ☐ $\sigma_1(\ell) = v \land (!\ell, \sigma_1) \to_{\mathsf{base}} (v, \sigma_1)$
- ☐ $\sigma_1(\ell) = v \land (\ell \leftarrow w, \sigma_1) \to_{\mathsf{base}} ((), \sigma_1[\ell \leftarrow w])$
- ☐ $\ell_0 \notin \sigma_1 \land (\mathbf{ref}\ v, \sigma_1) \to_{\mathsf{base}} (\ell_0, \sigma_1[\ell_0 \leftarrow v])$

Persiste

Spatial

Goal ✓

1. First, if e_1' takes a pure step, then **[click]** we can reduce the goal to **[click]** wp e_2' by wp-pure. **[click]**
2. **[click]** and strip **[click]** the later **[click]**
3. **[click]** we then use the inverse direction of wp-bind to **[click]** restore the goal to a form that matches the conclusion of the induction hypothesis **[click]**
4. **[click]** we can now apply the induction hypothesis, **[click]** which yields two subgoals
5. **[click]** and they can be solved by framing
6. **[click]** this concludes the pure step case

If $e_1' = !\,\ell$

Pure hyp

$\texttt{Hsafe1}: \mathsf{safe}_\varphi(K[!\,\ell], \sigma_1)$ $\qquad$ $\texttt{Hlookup}: \sigma_1(\ell) = v$

$\texttt{Hbase}: (!\,\ell, \sigma_1) \rightarrow_{\mathsf{base}} (v, \sigma_1)$

$\texttt{Hsafe2}: \mathsf{safe}_\varphi(K[v], \sigma_1)$

Persistent hyp

$\texttt{IH}: \triangleright \left(\forall e, \sigma.\, \ulcorner \mathsf{safe}_\varphi(e, \sigma) \urcorner \mathrel{-\!\!*} \mathsf{S}_\circ(\sigma) \mathrel{-\!\!*} \mathsf{wp}\ e \left\{ v.\, \ulcorner \varphi(v) \urcorner \right\}\right)$

Spatial hyp

$\texttt{H}\circ: \mathsf{S}_\circ(\sigma_1)$

Goal

$\mathsf{wp}\ !\,\ell \left\{ w.\, \mathsf{wp}\ K[w] \left\{ v.\, \ulcorner \varphi(v) \urcorner \right\} \right\}$

1. Third, if e_1' is a load
2. **[click]** we can destruct the state ownership to **[click]** extract a points-to assertion. This is possible because we have Hlookup, saying that ℓ is part of σ_1. **[click]**
3. **[click]** with Hl, we can now use wp-load to process the goal **[click]**
4. and **[click]** the remaining steps are similar to the second case
5. and **[click]** this concludes the load case

If $e_1' = !\,\ell$

Pure hyp

$\texttt{Hsafe1}\colon \mathsf{safe}_\varphi(K[!\,\ell], \sigma_1)$ $\qquad$ $\texttt{Hlookup}\colon \sigma_1(\ell) = v$

$\texttt{Hbase}\colon (!\,\ell, \sigma_1) \to_{\mathsf{base}} (v, \sigma_1)$

$\texttt{Hsafe2}\colon \mathsf{safe}_\varphi(K[v], \sigma_1)$

Persistent hyp $\quad \texttt{IH}\colon \triangleright \left(\forall e, \sigma.\, \ulcorner \mathsf{safe}_\varphi(e, \sigma) \urcorner \mathrel{-\!\!*} \mathsf{S}_\circ(\sigma) \mathrel{-\!\!*} \mathsf{wp}\ e \left\{ v.\, \ulcorner \varphi(v) \urcorner \right\}\right)$

Spatial hyp $\quad \texttt{H}\circ\colon \mathsf{S}_\circ(\sigma_1)$ $\qquad$ destruct $\qquad \mathsf{S}_\circ(\sigma_1) \triangleq \ast_{(\ell \leftarrow v) \in \sigma_1} (\ell \mapsto v)$

Goal $\quad \mathsf{wp}\ !\,\ell \left\{ w.\, \mathsf{wp}\ K[w] \left\{ v.\, \ulcorner \varphi(v) \urcorner \right\} \right\}$

1. Third, if e_1' is a load
2. **[click]** we can destruct the state ownership to **[click]** extract a points-to assertion. This is possible because we have Hlookup, saying that ℓ is part of σ_1. **[click]**
3. **[click]** with Hl, we can now use wp-load to process the goal **[click]**
4. and **[click]** the remaining steps are similar to the second case
5. and **[click]** this concludes the load case

If $e_1' = !\,\ell$

Pure hyp

Hsafe1: $\mathsf{safe}_\varphi(K[!\,\ell], \sigma_1)$

Hlookup: $\sigma_1(\ell) = v$

Hbase: $(!\,\ell, \sigma_1) \rightarrow_{\mathsf{base}} (v, \sigma_1)$

Hsafe2: $\mathsf{safe}_\varphi(K[v], \sigma_1)$

Persistent hyp

IH: $\triangleright \left(\forall e, \sigma.\ \ulcorner \mathsf{safe}_\varphi(e, \sigma) \urcorner \mathrel{-\!\!*} \mathsf{S}_\circ(\sigma) \mathrel{-\!\!*} \mathsf{wp}\ e \left\{ v.\ \ulcorner \varphi(v) \urcorner \right\}\right)$

Spatial hyp

Ho: $\mathsf{S}_\circ(\sigma_1 \setminus \{\ell\})$

Hl: $\ell \mapsto v$

destruct $\mathsf{S}_\circ(\sigma_1) \triangleq \ast_{(\ell \leftarrow v) \in \sigma_1} (\ell \mapsto v)$

Goal

$\mathsf{wp}\ !\,\ell \left\{ w.\ \mathsf{wp}\ K[w] \left\{ v.\ \ulcorner \varphi(v) \urcorner \right\} \right\}$

1. Third, if e_1' is a load
2. **[click]** we can destruct the state ownership to **[click]** extract a points-to assertion. This is possible because we have Hlookup, saying that ℓ is part of σ_1. **[click]**
3. **[click]** with Hl, we can now use wp-load to process the goal **[click]**
4. and **[click]** the remaining steps are similar to the second case
5. and **[click]** this concludes the load case

If $e_1' = !\,\ell$

Pure hyp

$\texttt{Hsafe1}\colon \mathsf{safe}_\varphi(K[!\,\ell], \sigma_1)$ $\qquad$ $\texttt{Hlookup}\colon \sigma_1(\ell) = v$

$\texttt{Hbase}\colon (!\,\ell, \sigma_1) \rightarrow_{\mathsf{base}} (v, \sigma_1)$

$\texttt{Hsafe2}\colon \mathsf{safe}_\varphi(K[v], \sigma_1)$

Persistent hyp

$\texttt{IH}\colon \triangleright \left(\forall e, \sigma.\, \ulcorner \mathsf{safe}_\varphi(e, \sigma) \urcorner \mathrel{-\!\!*} \mathsf{S}_\circ(\sigma) \mathrel{-\!\!*} \mathsf{wp}\ e \left\{ v.\, \ulcorner \varphi(v) \urcorner \right\}\right)$

Spatial hyp

$\texttt{H}\circ\colon \mathsf{S}_\circ(\sigma_1 \setminus \{\ell\})$

$\texttt{Hl}\colon \ell \mapsto v$

Goal

$\mathsf{wp}\ !\,\ell \left\{ w.\, \mathsf{wp}\ K[w] \left\{ v.\, \ulcorner \varphi(v) \urcorner \right\} \right\}$

1. Third, if e_1' is a load
2. **[click]** we can destruct the state ownership to **[click]** extract a points-to assertion. This is possible because we have Hlookup, saying that ℓ is part of σ_1. **[click]**
3. **[click]** with Hl, we can now use wp-load to process the goal **[click]**
4. and **[click]** the remaining steps are similar to the second case
5. and **[click]** this concludes the load case

If $e_1' = !\,\ell$

Pure hyp

$\texttt{Hsafe1}: \mathsf{safe}_\varphi(K[!\,\ell], \sigma_1)$ $\qquad$ $\texttt{Hlookup}: \sigma_1(\ell) = v$

$\texttt{Hbase}: (!\,\ell, \sigma_1) \rightarrow_{\mathsf{base}} (v, \sigma_1)$

$\texttt{Hsafe2}: \mathsf{safe}_\varphi(K[v], \sigma_1)$

Persistent hyp $\quad \texttt{IH}: \triangleright \left(\forall e, \sigma. \ulcorner \mathsf{safe}_\varphi(e, \sigma) \urcorner \mathrel{-\!\!*} \mathsf{S}_\circ(\sigma) \mathrel{-\!\!*} \mathsf{wp}\ e \left\{ v. \ulcorner \varphi(v) \urcorner \right\}\right)$

Spatial hyp

$\texttt{H}\circ: \mathsf{S}_\circ(\sigma_1 \setminus \{\ell\})$

$\texttt{Hl}: \ell \mapsto v$

Goal $\quad \mathsf{wp}\ !\,\ell \left\{ w. \mathsf{wp}\ K[w] \left\{ v. \ulcorner \varphi(v) \urcorner \right\} \right\}$

by WpLoad

1. Third, if e_1' is a load
2. **[click]** we can destruct the state ownership to **[click]** extract a points-to assertion. This is possible because we have Hlookup, saying that ℓ is part of σ_1. **[click]**
3. **[click]** with Hl, we can now use wp-load to process the goal **[click]**
4. and **[click]** the remaining steps are similar to the second case
5. and **[click]** this concludes the load case

If $e_1' = !\,\ell$

Pure hyp

`Hsafe1`: $\mathsf{safe}_\varphi(K[!\,\ell], \sigma_1)$ `Hlookup`: $\sigma_1(\ell) = v$

`Hbase`: $(!\,\ell, \sigma_1) \rightarrow_{\mathsf{base}} (v, \sigma_1)$

`Hsafe2`: $\mathsf{safe}_\varphi(K[v], \sigma_1)$

Persistent hyp `IH`: $\triangleright\, \left(\forall e, \sigma.\, \ulcorner \mathsf{safe}_\varphi(e, \sigma) \urcorner \mathrel{-\!\!*} \mathsf{S}_\circ(\sigma) \mathrel{-\!\!*} \mathsf{wp}\ e\, \left\{ v.\, \ulcorner \varphi(v) \urcorner \right\}\right)$

Spatial hyp `H`$\circ$: $\mathsf{S}_\circ(\sigma_1 \setminus \{\ell\})$

Goal $\triangleright\, (\ell \mapsto v \mathrel{-\!\!*} \mathsf{wp}\ K[v]\, \left\{ v.\, \ulcorner \varphi(v) \urcorner \right\})$

by WpLoad

1. Third, if e_1' is a load
2. **[click]** we can destruct the state ownership to **[click]** extract a points-to assertion. This is possible because we have Hlookup, saying that ℓ is part of σ_1. **[click]**
3. **[click]** with Hl, we can now use wp-load to process the goal **[click]**
4. and **[click]** the remaining steps are similar to the second case
5. and **[click]** this concludes the load case

If $e_1' = \,!\,\ell$

Pure hyp

$\texttt{Hsafe1}\colon \mathsf{safe}_\varphi(K[!\,\ell], \sigma_1)$ $\qquad$ $\texttt{Hlookup}\colon \sigma_1(\ell) = v$

$\texttt{Hbase}\colon (!\,\ell, \sigma_1) \rightarrow_{\mathsf{base}} (v, \sigma_1)$

$\texttt{Hsafe2}\colon \mathsf{safe}_\varphi(K[v], \sigma_1)$

Persistent hyp $\quad \texttt{IH}\colon \triangleright \left(\forall e, \sigma.\, \ulcorner \mathsf{safe}_\varphi(e, \sigma) \urcorner \mathrel{-\!\!*} \mathsf{S}_\circ(\sigma) \mathrel{-\!\!*} \mathsf{wp}\ e \left\{ v.\, \ulcorner \varphi(v) \urcorner \right\}\right)$

Spatial hyp $\quad \texttt{H}\circ\colon \mathsf{S}_\circ(\sigma_1 \setminus \{\ell\})$

Goal $\quad \triangleright (\ell \mapsto v \mathrel{-\!\!*} \mathsf{wp}\ K[v] \left\{ v.\, \ulcorner \varphi(v) \urcorner \right\})$

1. Third, if e_1' is a load
2. **[click]** we can destruct the state ownership to **[click]** extract a points-to assertion. This is possible because we have Hlookup, saying that ℓ is part of σ_1. **[click]**
3. **[click]** with Hl, we can now use wp-load to process the goal **[click]**
4. and **[click]** the remaining steps are similar to the second case
5. and **[click]** this concludes the load case

If $e_1' = !\,\ell$

Pure hyp

$\texttt{Hsafe1}: \mathsf{safe}_\varphi(K[!\,\ell], \sigma_1)$ $\quad$ $\texttt{Hlookup}: \sigma_1(\ell) = v$

$\texttt{Hbase}: (!\,\ell, \sigma_1) \rightarrow_{\mathsf{base}} (v, \sigma_1)$

$\texttt{Hsafe2}: \mathsf{safe}_\varphi(K[v], \sigma_1)$

Persistent hyp $\quad \texttt{IH}: \triangleright \left(\forall e, \sigma. \ulcorner \mathsf{safe}_\varphi(e, \sigma) \urcorner \mathrel{-\!\!*} \mathsf{S}_\circ(\sigma) \mathrel{-\!\!*} \mathsf{wp}\ e \left\{ v. \ulcorner \varphi(v) \urcorner \right\}\right)$

Spatial hyp $\quad \texttt{H}\circ: \mathsf{S}_\circ(\sigma_1 \setminus \{\ell\})$

Goal $\quad \triangleright (\ell \mapsto v \mathrel{-\!\!*} \mathsf{wp}\ K[v] \left\{ v. \ulcorner \varphi(v) \urcorner \right\})$

strip later and intro

1. Third, if e_1' is a load
2. **[click]** we can destruct the state ownership to **[click]** extract a points-to assertion. This is possible because we have Hlookup, saying that ℓ is part of σ_1. **[click]**
3. **[click]** with Hl, we can now use wp-load to process the goal **[click]**
4. and **[click]** the remaining steps are similar to the second case
5. and **[click]** this concludes the load case

If $e_1' = !\,\ell$

Pure hyp

$\texttt{Hsafe1}: \mathsf{safe}_\varphi(K[!\,\ell], \sigma_1)$ $\qquad$ $\texttt{Hlookup}: \sigma_1(\ell) = v$

$\texttt{Hbase}: (!\,\ell, \sigma_1) \rightarrow_{\mathsf{base}} (v, \sigma_1)$

$\texttt{Hsafe2}: \mathsf{safe}_\varphi(K[v], \sigma_1)$

Persistent hyp

$\texttt{IH}: \quad \left(\forall e, \sigma.\, \ulcorner \mathsf{safe}_\varphi(e, \sigma) \urcorner \mathrel{-\!\!*} \mathsf{S}_\circ(\sigma) \mathrel{-\!\!*} \mathsf{wp}\ e \left\{ v.\, \ulcorner \varphi(v) \urcorner \right\}\right)$

Spatial hyp

$\texttt{H}\circ: \mathsf{S}_\circ(\sigma_1 \setminus \{\ell\})$

$\texttt{Hl}: \ell \mapsto v$

Goal

$\mathsf{wp}\ K[v] \left\{ v.\, \ulcorner \varphi(v) \urcorner \right\}$

strip later and intro

1. Third, if e_1' is a load
2. **[click]** we can destruct the state ownership to **[click]** extract a points-to assertion. This is possible because we have Hlookup, saying that ℓ is part of σ_1. **[click]**
3. **[click]** with Hl, we can now use wp-load to process the goal **[click]**
4. and **[click]** the remaining steps are similar to the second case
5. and **[click]** this concludes the load case

If $e_1' = !\,\ell$

Pure hyp

$\texttt{Hsafe1}: \mathsf{safe}_\varphi(K[!\,\ell], \sigma_1)$ $\qquad$ $\texttt{Hlookup}: \sigma_1(\ell) = v$

$\texttt{Hbase}: (!\,\ell, \sigma_1) \rightarrow_{\mathsf{base}} (v, \sigma_1)$

$\texttt{Hsafe2}: \mathsf{safe}_\varphi(K[v], \sigma_1)$

Persistent hyp

$\texttt{IH}: \quad \left(\forall e, \sigma.\, \ulcorner \mathsf{safe}_\varphi(e, \sigma) \urcorner \mathrel{-\!\!*} \mathsf{S}_\circ(\sigma) \mathrel{-\!\!*} \mathsf{wp}\ e \left\{ v.\, \ulcorner \varphi(v) \urcorner \right\}\right)$ □

Spatial hyp

$\texttt{H}\circ: \mathsf{S}_\circ(\sigma_1 \setminus \{\ell\})$

$\texttt{Hl}: \ell \mapsto v$

Goal

$\mathsf{wp}\ K[v] \left\{ v.\, \ulcorner \varphi(v) \urcorner \right\}$ ∗

1. Third, if e_1' is a load
2. **[click]** we can destruct the state ownership to **[click]** extract a points-to assertion. This is possible because we have Hlookup, saying that ℓ is part of σ_1. **[click]**
3. **[click]** with Hl, we can now use wp-load to process the goal **[click]**
4. and **[click]** the remaining steps are similar to the second case
5. and **[click]** this concludes the load case

If $e_1' = !\,\ell$

Pure hyp

Hsafe1: $\mathsf{safe}_\varphi(K[!\,\ell], \sigma_1)$ Hlookup: $\sigma_1(\ell) = v$

Hbase: $(!\,\ell, \sigma_1) \to_{\mathsf{base}} (v, \sigma_1)$

Hsafe2: $\mathsf{safe}_\varphi(K[v], \sigma_1)$

Persistent hyp

IH: $\left(\forall e, \sigma.\ \ulcorner \mathsf{safe}_\varphi(e, \sigma) \urcorner \mathrel{-\!\!*} \mathsf{S}_\circ(\sigma) \mathrel{-\!\!*} \mathsf{wp}\ e \left\{ v.\ \ulcorner \varphi(v) \urcorner \right\}\right)$

Spatial hyp

Ho: $\mathsf{S}_\circ(\sigma_1 \setminus \{\ell\})$

Hl: $\ell \mapsto v$

combine $\mathsf{S}_\circ(\sigma_1) \triangleq \ast_{(\ell \leftarrow v) \in \sigma_1} (\ell \mapsto v)$

Goal

$\mathsf{wp}\ K[v] \left\{ v.\ \ulcorner \varphi(v) \urcorner \right\}$

1. Third, if e_1' is a load
2. **[click]** we can destruct the state ownership to **[click]** extract a points-to assertion. This is possible because we have Hlookup, saying that ℓ is part of σ_1. **[click]**
3. **[click]** with Hl, we can now use wp-load to process the goal **[click]**
4. and **[click]** the remaining steps are similar to the second case
5. and **[click]** this concludes the load case

If $e_1' = !\,\ell$

Pure hyp

$\texttt{Hsafe1}: \mathsf{safe}_\varphi(K[!\,\ell], \sigma_1)$ $\qquad$ $\texttt{Hlookup}: \sigma_1(\ell) = v$

$\texttt{Hbase}: (!\,\ell, \sigma_1) \to_{\mathsf{base}} (v, \sigma_1)$

$\texttt{Hsafe2}: \mathsf{safe}_\varphi(K[v], \sigma_1)$

Persistent hyp $\quad \texttt{IH}: \quad \left(\forall e, \sigma.\, \ulcorner \mathsf{safe}_\varphi(e, \sigma) \urcorner \mathrel{-\!\!*} \mathsf{S}_\circ(\sigma) \mathrel{-\!\!*} \mathsf{wp}\ e \left\{ v.\, \ulcorner \varphi(v) \urcorner \right\}\right)$

Spatial hyp $\quad \texttt{H}\circ: \mathsf{S}_\circ(\sigma_1)$

combine $\quad \mathsf{S}_\circ(\sigma_1) \triangleq \mathop{\Large *}_{(\ell \leftarrow v) \in \sigma_1} (\ell \mapsto v)$

Goal $\quad \mathsf{wp}\ K[v] \left\{ v.\, \ulcorner \varphi(v) \urcorner \right\}$

1. Third, if e_1' is a load
2. **[click]** we can destruct the state ownership to **[click]** extract a points-to assertion. This is possible because we have Hlookup, saying that ℓ is part of σ_1. **[click]**
3. **[click]** with Hl, we can now use wp-load to process the goal **[click]**
4. and **[click]** the remaining steps are similar to the second case
5. and **[click]** this concludes the load case

If $e_1' = !\,\ell$

Pure hyp

$\texttt{Hsafe1}: \mathsf{safe}_\varphi(K[!\,\ell], \sigma_1)$ $\qquad$ $\texttt{Hlookup}: \sigma_1(\ell) = v$

$\texttt{Hbase}: (!\,\ell, \sigma_1) \rightarrow_{\mathsf{base}} (v, \sigma_1)$

$\texttt{Hsafe2}: \mathsf{safe}_\varphi(K[v], \sigma_1)$

Persistent hyp $\quad \texttt{IH}: \quad \left(\forall e, \sigma.\, \ulcorner \mathsf{safe}_\varphi(e, \sigma) \urcorner \mathrel{-\!\!*} \mathsf{S}_\circ(\sigma) \mathrel{-\!\!*} \mathsf{wp}\ e \left\{ v.\, \ulcorner \varphi(v) \urcorner \right\}\right)$

Spatial hyp $\quad \texttt{H}\circ: \mathsf{S}_\circ(\sigma_1)$

Goal $\quad \mathsf{wp}\ K[v] \left\{ v.\, \ulcorner \varphi(v) \urcorner \right\}$

1. Third, if e_1' is a load
2. **[click]** we can destruct the state ownership to **[click]** extract a points-to assertion. This is possible because we have Hlookup, saying that ℓ is part of σ_1. **[click]**
3. **[click]** with Hl, we can now use wp-load to process the goal **[click]**
4. and **[click]** the remaining steps are similar to the second case
5. and **[click]** this concludes the load case

If $e_1' = !\,\ell$

Pure hyp

`Hsafe1`: $\mathsf{safe}_\varphi(K[!\,\ell], \sigma_1)$ `Hlookup`: $\sigma_1(\ell) = v$

`Hbase`: $(!\,\ell, \sigma_1) \rightarrow_{\mathsf{base}} (v, \sigma_1)$

`Hsafe2`: $\mathsf{safe}_\varphi(K[v], \sigma_1)$ frame

Persistent hyp `IH`: $\left(\forall e, \sigma.\ \ulcorner \mathsf{safe}_\varphi(e, \sigma) \urcorner \mathrel{-\!\!*} \mathsf{S}_\circ(\sigma) \mathrel{-\!\!*} \mathsf{wp}\ e\ \left\{ v.\ \ulcorner \varphi(v) \urcorner \right\}\right)$ apply

Spatial hyp `H`$\circ$: $\mathsf{S}_\circ(\sigma_1)$ frame

Goal $\mathsf{wp}\ K[v]\ \left\{ v.\ \ulcorner \varphi(v) \urcorner \right\}$

1. Third, if e_1' is a load
2. **[click]** we can destruct the state ownership to **[click]** extract a points-to assertion. This is possible because we have Hlookup, saying that ℓ is part of σ_1. **[click]**
3. **[click]** with Hl, we can now use wp-load to process the goal **[click]**
4. and **[click]** the remaining steps are similar to the second case
5. and **[click]** this concludes the load case

If $e_1' = !\,\ell$

Pure hyp

`Hsafe1`: $\mathsf{safe}_\varphi(K[!\,\ell], \sigma_1)$ `Hlookup`: $\sigma_1(\ell) = v$

`Hbase`: $(!\,\ell, \sigma_1) \rightarrow_{\mathsf{base}} (v, \sigma_1)$

`Hsafe2`: $\mathsf{safe}_\varphi(K[v], \sigma_1)$

Persistent hyp `IH`: $\left(\forall e, \sigma.\, \ulcorner \mathsf{safe}_\varphi(e, \sigma) \urcorner \mathrel{-\!\!*} \mathsf{S}_\circ(\sigma) \mathrel{-\!\!*} \mathsf{wp}\ e \left\{ v.\, \ulcorner \varphi(v) \urcorner \right\}\right)$ apply

---

Spatial hyp `H`$\circ$: $\mathsf{S}_\circ(\sigma_1)$ framed

Goal ✓

1. Third, if e_1' is a load
2. **[click]** we can destruct the state ownership to **[click]** extract a points-to assertion. This is possible because we have Hlookup, saying that ℓ is part of σ_1. **[click]**
3. **[click]** with Hl, we can now use wp-load to process the goal **[click]**
4. and **[click]** the remaining steps are similar to the second case
5. and **[click]** this concludes the load case

If $e_1' = !\,\ell$

Pure hyp $\quad$ `Hsafe1`: $\mathsf{safe}_\varphi(K[!\,\ell], \sigma_1)$ $\quad$ `Hlookup`: $\sigma_1(\ell) = v$

Persiste

Spatial

- ☑ $e_1' \to_{\mathsf{pure}} e_2'$
- ☑ $\sigma_1(\ell) = v \land (!\ell, \sigma_1) \to_{\mathsf{base}} (v, \sigma_1)$
- ☐ $\sigma_1(\ell) = v \land (\ell \leftarrow w, \sigma_1) \to_{\mathsf{base}} ((), \sigma_1[\ell \leftarrow w])$
- ☐ $\ell_0 \notin \sigma_1 \land (\mathbf{ref}\ v, \sigma_1) \to_{\mathsf{base}} (\ell_0, \sigma_1[\ell_0 \leftarrow v])$

Goal $\quad$ ✓

1. Third, if e_1' is a load
2. **[click]** we can destruct the state ownership to **[click]** extract a points-to assertion. This is possible because we have Hlookup, saying that ℓ is part of σ_1. **[click]**
3. **[click]** with Hl, we can now use wp-load to process the goal **[click]**
4. and **[click]** the remaining steps are similar to the second case
5. and **[click]** this concludes the load case

If $e_1' = \ell \leftarrow w$

Pure hyp

$\texttt{Hsafe1}: \mathsf{safe}_\varphi(K[\ell \leftarrow w], \sigma_1)$ $\quad$ $\texttt{Hlookup}: \sigma_1(\ell) = v$

$\texttt{Hbase}: (\ell \leftarrow w, \sigma_1) \rightarrow_{\mathsf{base}} ((), \sigma_1[\ell \leftarrow w])$

$\texttt{Hsafe2}: \mathsf{safe}_\varphi(K[()], \sigma_1[\ell \leftarrow w])$

Persistent hyp

$\texttt{IH}: \triangleright \left(\forall e, \sigma. \ulcorner \mathsf{safe}_\varphi(e, \sigma) \urcorner \mathrel{-\!\!*} \mathsf{S}_\circ(\sigma) \mathrel{-\!\!*} \mathsf{wp}\ e \left\{ v. \ulcorner \varphi(v) \urcorner \right\}\right)$

---□

Spatial hyp

$\texttt{H}\circ: \mathsf{S}_\circ(\sigma_1)$

---∗

Goal

$\mathsf{wp}\ \ell \leftarrow w \left\{ w. \mathsf{wp}\ K[w] \left\{ v. \ulcorner \varphi(v) \urcorner \right\} \right\}$

1. finally, for store, everything is similar except that **[click]** we use wp-store **[click]**
2. now we have finished the whole proof of the sequential completeness

If $e_1' = \ell \leftarrow w$

Pure hyp

$\texttt{Hsafe1}: \mathsf{safe}_\varphi(K[\ell \leftarrow w], \sigma_1)$ $\quad$ $\texttt{Hlookup}: \sigma_1(\ell) = v$

$\texttt{Hbase}: (\ell \leftarrow w, \sigma_1) \rightarrow_{\mathsf{base}} ((), \sigma_1[\ell \leftarrow w])$

$\texttt{Hsafe2}: \mathsf{safe}_\varphi(K[()], \sigma_1[\ell \leftarrow w])$

Persistent hyp $\quad \texttt{IH}: \triangleright \left(\forall e, \sigma. \ulcorner \mathsf{safe}_\varphi(e, \sigma) \urcorner \mathrel{-\!\!*} \mathsf{S}_\circ(\sigma) \mathrel{-\!\!*} \mathsf{wp}\ e \left\{ v. \ulcorner \varphi(v) \urcorner \right\}\right)$

Spatial hyp $\quad \texttt{H}\circ: \mathsf{S}_\circ(\sigma_1)$

Goal $\quad \mathsf{wp}\ \ell \leftarrow w \left\{ w. \mathsf{wp}\ K[w] \left\{ v. \ulcorner \varphi(v) \urcorner \right\} \right\}$

by WpStore

1. finally, for store, everything is similar except that **[click]** we use wp-store **[click]**
2. now we have finished the whole proof of the sequential completeness

If $e_1' = \ell \leftarrow w$

Pure hyp $\texttt{Hsafe1}: \mathsf{safe}_\varphi(K[\ell \leftarrow w], \sigma_1)$ $\texttt{Hlookup}: \sigma_1(\ell) = v$

☑ $e_1' \to_{\mathsf{pure}} e_2'$

☑ $\sigma_1(\ell) = v \land (!\ell, \sigma_1) \to_{\mathsf{base}} (v, \sigma_1)$

Persiste ☑ $\sigma_1(\ell) = v \land (\ell \leftarrow w, \sigma_1) \to_{\mathsf{base}} ((), \sigma_1[\ell \leftarrow w])$

Spatial ☐ $\ell_0 \notin \sigma_1 \land (\texttt{ref}\ v, \sigma_1) \to_{\mathsf{base}} (\ell_0, \sigma_1[\ell_0 \leftarrow v])$

Goal $\mathsf{wp}\ \ell \leftarrow w \left\{ w.\ \mathsf{wp}\ K[w] \left\{ v.\ \ulcorner \varphi(v) \urcorner \right\} \right\}$ $*$

by WpStore

1. finally, for store, everything is similar except that **[click]** we use wp-store **[click]**
2. now we have finished the whole proof of the sequential completeness

If $e_1' = \texttt{ref}\ v$

Pure hyp

$\texttt{Hsafe1}: \mathsf{safe}_\varphi(K[\texttt{ref}\ v], \sigma_1)$

$\texttt{Hbase}: (\texttt{ref}\ v, \sigma_1) \rightarrow_{\mathsf{base}} (\ell_0, \sigma_1[\ell_0 \leftarrow v])$

$\texttt{Hsafe2}: \mathsf{safe}_\varphi(K[\ell_0], \sigma_1[\ell_0 \leftarrow v])$

Persistent hyp

$\texttt{IH}: \triangleright \left(\forall e, \sigma.\ \ulcorner \mathsf{safe}_\varphi(e, \sigma) \urcorner \mathrel{-\!\!*} \mathsf{S}_\circ(\sigma) \mathrel{-\!\!*} \mathsf{wp}\ e \left\{ v.\ \ulcorner \varphi(v) \urcorner \right\}\right)$

Spatial hyp

$\texttt{H}\circ: \mathsf{S}_\circ(\sigma_1)$

Goal

$\mathsf{wp}\ \texttt{ref}\ v \left\{ w.\ \mathsf{wp}\ K[w] \left\{ v.\ \ulcorner \varphi(v) \urcorner \right\} \right\}$

1. Second, if e_1' is alloc
2. **[click]** we use wp-alloc to **[click]** prove the first step **[click]**
3. **[click]** strip the later and intro **[click] [click]**
4. **[click]** note that this newly allocated location ℓ is unrelated to ℓ_0 resulting from destructing the base step
5. these two hypotheses are actually useless now, and we can discard them **[click] [click]**
6. **[click]** now we have the ownership of σ_1 and the newly allocated location
7. we can combine them **[click]** and meanwhile learn that ℓ is not in σ_1. **[click]**
8. **[click]** now we apply the induction hypothesis, and solve the state ownership part by framing **[click] [click]**
9. **[click]** we are left with a pure goal **[click]** we can do pure intro to turn the goal into a plain Rocq goal **[click]**
10. **[click]** this requires us to prove **`ref`** v can reduce to ℓ in one base step
11. **[click]** which is true because ℓ is fresh
12. **[click]** this concludes the alloc case

If $e_1' = \texttt{ref}\ v$

Pure hyp

$\texttt{Hsafe1}: \mathsf{safe}_\varphi(K[\texttt{ref}\ v], \sigma_1)$

$\texttt{Hbase}: (\texttt{ref}\ v, \sigma_1) \rightarrow_{\mathsf{base}} (\ell_0, \sigma_1[\ell_0 \leftarrow v])$

$\texttt{Hsafe2}: \mathsf{safe}_\varphi(K[\ell_0], \sigma_1[\ell_0 \leftarrow v])$

Persistent hyp

$\texttt{IH}: \triangleright \left(\forall e, \sigma.\ \ulcorner \mathsf{safe}_\varphi(e, \sigma) \urcorner \mathrel{-\!\!*} \mathsf{S}_\circ(\sigma) \mathrel{-\!\!*} \mathsf{wp}\ e \left\{ v.\ \ulcorner \varphi(v) \urcorner \right\}\right)$

Spatial hyp

$\texttt{H}\circ: \mathsf{S}_\circ(\sigma_1)$

Goal

$\mathsf{wp}\ \texttt{ref}\ v \left\{ w.\ \mathsf{wp}\ K[w] \left\{ v.\ \ulcorner \varphi(v) \urcorner \right\} \right\}$

by WpAlloc

1. Second, if e_1' is alloc
2. **[click]** we use wp-alloc to **[click]** prove the first step **[click]**
3. **[click]** strip the later and intro **[click] [click]**
4. **[click]** note that this newly allocated location ℓ is unrelated to ℓ_0 resulting from destructing the base step
5. these two hypotheses are actually useless now, and we can discard them **[click] [click]**
6. **[click]** now we have the ownership of σ_1 and the newly allocated location
7. we can combine them **[click]** and meanwhile learn that ℓ is not in σ_1. **[click]**
8. **[click]** now we apply the induction hypothesis, and solve the state ownership part by framing **[click] [click]**
9. **[click]** we are left with a pure goal **[click]** we can do pure intro to turn the goal into a plain Rocq goal **[click]**
10. **[click]** this requires us to prove **`ref`** v can reduce to ℓ in one base step
11. **[click]** which is true because ℓ is fresh
12. **[click]** this concludes the alloc case

If $e_1' = \texttt{ref}\ v$

Pure hyp

$\texttt{Hsafe1}: \mathsf{safe}_\varphi(K[\texttt{ref}\ v], \sigma_1)$ $\qquad$ $\texttt{Hfresh}: \ell \notin \sigma_1$

$\texttt{Hbase}: (\texttt{ref}\ v, \sigma_1) \to_{\mathsf{base}} (\ell_0, \sigma_1[\ell_0 \leftarrow v])$

$\texttt{Hsafe2}: \mathsf{safe}_\varphi(K[\ell_0], \sigma_1[\ell_0 \leftarrow v])$

Persistent hyp $\quad \texttt{IH}: \quad \left(\forall e, \sigma.\ \ulcorner \mathsf{safe}_\varphi(e, \sigma) \urcorner \mathrel{-\!\!*} \mathsf{S}_\circ(\sigma) \mathrel{-\!\!*} \mathsf{wp}\ e \left\{ v.\ \ulcorner \varphi(v) \urcorner \right\}\right)$

Spatial hyp $\quad \texttt{H}\circ: \mathsf{S}_\circ(\sigma_1[\ell \leftarrow v])$

Goal $\quad \mathsf{wp}\ K[\ell] \left\{ v.\ \ulcorner \varphi(v) \urcorner \right\}$

by WpAlloc

1. Second, if e_1' is alloc
2. **[click]** we use wp-alloc to **[click]** prove the first step **[click]**
3. **[click]** strip the later and intro **[click] [click]**
4. **[click]** note that this newly allocated location ℓ is unrelated to ℓ_0 resulting from destructing the base step
5. these two hypotheses are actually useless now, and we can discard them **[click] [click]**
6. **[click]** now we have the ownership of σ_1 and the newly allocated location
7. we can combine them **[click]** and meanwhile learn that ℓ is not in σ_1. **[click]**
8. **[click]** now we apply the induction hypothesis, and solve the state ownership part by framing **[click] [click]**
9. **[click]** we are left with a pure goal **[click]** we can do pure intro to turn the goal into a plain Rocq goal **[click]**
10. **[click]** this requires us to prove **`ref`** v can reduce to ℓ in one base step
11. **[click]** which is true because ℓ is fresh
12. **[click]** this concludes the alloc case

If $e_1' = \texttt{ref}\ v$

Pure hyp

$\texttt{Hsafe1}: \mathsf{safe}_\varphi(K[\texttt{ref}\ v], \sigma_1)$ $\qquad$ $\texttt{Hfresh}: \ell \notin \sigma_1$

$\texttt{Hbase}: (\texttt{ref}\ v, \sigma_1) \to_{\mathsf{base}} (\ell_0, \sigma_1[\ell_0 \leftarrow v])$

$\texttt{Hsafe2}: \mathsf{safe}_\varphi(K[\ell_0], \sigma_1[\ell_0 \leftarrow v])$

Persistent hyp $\texttt{IH}: \quad \left(\forall e, \sigma.\ \ulcorner \mathsf{safe}_\varphi(e, \sigma) \urcorner \mathrel{-\!\!*} \mathsf{S}_\circ(\sigma) \mathrel{-\!\!*} \mathsf{wp}\ e \left\{ v.\ \ulcorner \varphi(v) \urcorner \right\}\right)$

Spatial hyp $\texttt{H}\circ: \mathsf{S}_\circ(\sigma_1[\ell \leftarrow v])$

Goal $\mathsf{wp}\ K[\ell] \left\{ v.\ \ulcorner \varphi(v) \urcorner \right\}$

1. Second, if e_1' is alloc
2. **[click]** we use wp-alloc to **[click]** prove the first step **[click]**
3. **[click]** strip the later and intro **[click] [click]**
4. **[click]** note that this newly allocated location ℓ is unrelated to ℓ_0 resulting from destructing the base step
5. these two hypotheses are actually useless now, and we can discard them **[click] [click]**
6. **[click]** now we have the ownership of σ_1 and the newly allocated location
7. we can combine them **[click]** and meanwhile learn that ℓ is not in σ_1. **[click]**
8. **[click]** now we apply the induction hypothesis, and solve the state ownership part by framing **[click] [click]**
9. **[click]** we are left with a pure goal **[click]** we can do pure intro to turn the goal into a plain Rocq goal **[click]**
10. **[click]** this requires us to prove **`ref`** v can reduce to ℓ in one base step
11. **[click]** which is true because ℓ is fresh
12. **[click]** this concludes the alloc case

If $e_1' = \texttt{ref}\ v$

Pure hyp

$\texttt{Hsafe1}: \mathsf{safe}_\varphi(K[\texttt{ref}\ v], \sigma_1)$ $\quad$ $\texttt{Hfresh}: \ell \notin \sigma_1$

$\texttt{Hbase}: (\texttt{ref}\ v, \sigma_1) \to_{\mathsf{base}} (\ell_0, \sigma_1[\ell_0 \leftarrow v])$

$\texttt{Hsafe2}: \mathsf{safe}_\varphi(K[\ell_0], \sigma_1[\ell_0 \leftarrow v])$

ℓ_0 and ℓ are unrelated

Persistent hyp $\quad \texttt{IH}: \quad \left(\forall e, \sigma.\ \ulcorner \mathsf{safe}_\varphi(e, \sigma) \urcorner \mathrel{-\!\!*} \mathsf{S}_\circ(\sigma) \mathrel{-\!\!*} \mathsf{wp}\ e \left\{ v.\ \ulcorner \varphi(v) \urcorner \right\}\right)$

Spatial hyp $\quad \texttt{H}\circ: \mathsf{S}_\circ(\sigma_1[\ell \leftarrow v])$

Goal $\quad \mathsf{wp}\ K[\ell] \left\{ v.\ \ulcorner \varphi(v) \urcorner \right\}$

1. Second, if e_1' is alloc
2. **[click]** we use wp-alloc to **[click]** prove the first step **[click]**
3. **[click]** strip the later and intro **[click] [click]**
4. **[click]** note that this newly allocated location ℓ is unrelated to ℓ_0 resulting from destructing the base step
5. these two hypotheses are actually useless now, and we can discard them **[click] [click]**
6. **[click]** now we have the ownership of σ_1 and the newly allocated location
7. we can combine them **[click]** and meanwhile learn that ℓ is not in σ_1. **[click]**
8. **[click]** now we apply the induction hypothesis, and solve the state ownership part by framing **[click] [click]**
9. **[click]** we are left with a pure goal **[click]** we can do pure intro to turn the goal into a plain Rocq goal **[click]**
10. **[click]** this requires us to prove **`ref`** v can reduce to ℓ in one base step
11. **[click]** which is true because ℓ is fresh
12. **[click]** this concludes the alloc case

If $e_1' = \texttt{ref}\ v$

Pure hyp

$\texttt{Hsafe1}: \mathsf{safe}_\varphi(K[\texttt{ref}\ v], \sigma_1)$ $\quad$ $\texttt{Hfresh}: \ell \notin \sigma_1$

$\texttt{Hbase}: (\texttt{ref}\ v, \sigma_1) \to_{\mathsf{base}} (\ell_0, \sigma_1[\ell_0 \leftarrow v])$

$\texttt{Hsafe2}: \mathsf{safe}_\varphi(K[\ell_0], \sigma_1[\ell_0 \leftarrow v])$

ℓ_0 and ℓ are unrelated

Persistent hyp $\quad \texttt{IH}: \quad \left(\forall e, \sigma.\ \ulcorner \mathsf{safe}_\varphi(e, \sigma) \urcorner \mathrel{-\!\!*} \mathsf{S}_\circ(\sigma) \mathrel{-\!\!*} \mathsf{wp}\ e \left\{ v.\ \ulcorner \varphi(v) \urcorner \right\}\right)$

---□

Spatial hyp $\quad \texttt{H}\circ: \mathsf{S}_\circ(\sigma_1[\ell \leftarrow v])$

Goal $\quad \mathsf{wp}\ K[\ell] \left\{ v.\ \ulcorner \varphi(v) \urcorner \right\}$

1. Second, if e_1' is alloc
2. **[click]** we use wp-alloc to **[click]** prove the first step **[click]**
3. **[click]** strip the later and intro **[click] [click]**
4. **[click]** note that this newly allocated location ℓ is unrelated to ℓ_0 resulting from destructing the base step
5. these two hypotheses are actually useless now, and we can discard them **[click] [click]**
6. **[click]** now we have the ownership of σ_1 and the newly allocated location
7. we can combine them **[click]** and meanwhile learn that ℓ is not in σ_1. **[click]**
8. **[click]** now we apply the induction hypothesis, and solve the state ownership part by framing **[click] [click]**
9. **[click]** we are left with a pure goal **[click]** we can do pure intro to turn the goal into a plain Rocq goal **[click]**
10. **[click]** this requires us to prove **`ref`** v can reduce to ℓ in one base step
11. **[click]** which is true because ℓ is fresh
12. **[click]** this concludes the alloc case

If $e_1' = \texttt{ref}\ v$

Pure hyp

$\texttt{Hsafe1}: \mathsf{safe}_\varphi(K[\texttt{ref}\ v], \sigma_1)$ $\quad$ $\texttt{Hfresh}: \ell \notin \sigma_1$

$\texttt{Hbase}: (\texttt{ref}\ v, \sigma_1) \to_{\mathsf{base}} (\ell_0, \sigma_1[\ell_0 \leftarrow v])$

$\texttt{Hsafe2}: \mathsf{safe}_\varphi(K[\ell_0], \sigma_1[\ell_0 \leftarrow v])$

Persistent hyp $\quad \texttt{IH}: \quad \left(\forall e, \sigma.\ \ulcorner \mathsf{safe}_\varphi(e, \sigma) \urcorner \mathrel{-\!\!*} \mathsf{S}_\circ(\sigma) \mathrel{-\!\!*} \mathsf{wp}\ e \left\{ v.\ \ulcorner \varphi(v) \urcorner \right\}\right)$

Spatial hyp

Goal $\quad \ulcorner \mathsf{safe}_\varphi(K[\ell], \sigma_1[\ell \leftarrow v]) \urcorner$

1. Second, if e_1' is alloc
2. **[click]** we use wp-alloc to **[click]** prove the first step **[click]**
3. **[click]** strip the later and intro **[click] [click]**
4. **[click]** note that this newly allocated location ℓ is unrelated to ℓ_0 resulting from destructing the base step
5. these two hypotheses are actually useless now, and we can discard them **[click] [click]**
6. **[click]** now we have the ownership of σ_1 and the newly allocated location
7. we can combine them **[click]** and meanwhile learn that ℓ is not in σ_1. **[click]**
8. **[click]** now we apply the induction hypothesis, and solve the state ownership part by framing **[click] [click]**
9. **[click]** we are left with a pure goal **[click]** we can do pure intro to turn the goal into a plain Rocq goal **[click]**
10. **[click]** this requires us to prove **`ref`** v can reduce to ℓ in one base step
11. **[click]** which is true because ℓ is fresh
12. **[click]** this concludes the alloc case

If $e_1' = \texttt{ref}\ v$

Pure hyp

$\texttt{Hsafe1}: \mathsf{safe}_\varphi(K[\texttt{ref}\ v], \sigma_1)$ $\qquad$ $\texttt{Hfresh}: \ell \notin \sigma_1$

pure intro

$\texttt{Hbase}: (\texttt{ref}\ v, \sigma_1) \to_{\mathsf{base}} (\ell_0, \sigma_1[\ell_0 \leftarrow v])$

$\texttt{Hsafe2}: \mathsf{safe}_\varphi(K[\ell_0], \sigma_1[\ell_0 \leftarrow v])$

Persistent hyp $\quad \texttt{IH}: \quad \left(\forall e, \sigma.\ \ulcorner \mathsf{safe}_\varphi(e, \sigma) \urcorner \mathrel{-\!\!*} \mathsf{S}_\circ(\sigma) \mathrel{-\!\!*} \mathsf{wp}\ e \left\{ v.\ \ulcorner \varphi(v) \urcorner \right\}\right)$

Spatial hyp

Goal $\quad \ulcorner \mathsf{safe}_\varphi(K[\ell], \sigma_1[\ell \leftarrow v]) \urcorner$

1. Second, if e_1' is alloc
2. **[click]** we use wp-alloc to **[click]** prove the first step **[click]**
3. **[click]** strip the later and intro **[click] [click]**
4. **[click]** note that this newly allocated location ℓ is unrelated to ℓ_0 resulting from destructing the base step
5. these two hypotheses are actually useless now, and we can discard them **[click] [click]**
6. **[click]** now we have the ownership of σ_1 and the newly allocated location
7. we can combine them **[click]** and meanwhile learn that ℓ is not in σ_1. **[click]**
8. **[click]** now we apply the induction hypothesis, and solve the state ownership part by framing **[click] [click]**
9. **[click]** we are left with a pure goal **[click]** we can do pure intro to turn the goal into a plain Rocq goal **[click]**
10. **[click]** this requires us to prove **`ref`** v can reduce to ℓ in one base step
11. **[click]** which is true because ℓ is fresh
12. **[click]** this concludes the alloc case

If $e_1' = \texttt{ref}\ v$

Pure hyp

$\texttt{Hsafe1}: \mathsf{safe}_\varphi(K[\texttt{ref}\ v], \sigma_1)$ $\qquad$ $\texttt{Hfresh}: \ell \notin \sigma_1$

pure intro

$\texttt{Hbase}: (\texttt{ref}\ v, \sigma_1) \to_{\mathsf{base}} (\ell_0, \sigma_1[\ell_0 \leftarrow v])$

$\texttt{Hsafe2}: \mathsf{safe}_\varphi(K[\ell_0], \sigma_1[\ell_0 \leftarrow v])$

$\mathsf{safe}_\varphi(K[\ell], \sigma_1[\ell \leftarrow v])$

1. Second, if e_1' is alloc
2. **[click]** we use wp-alloc to **[click]** prove the first step **[click]**
3. **[click]** strip the later and intro **[click] [click]**
4. **[click]** note that this newly allocated location ℓ is unrelated to ℓ_0 resulting from destructing the base step
5. these two hypotheses are actually useless now, and we can discard them **[click] [click]**
6. **[click]** now we have the ownership of σ_1 and the newly allocated location
7. we can combine them **[click]** and meanwhile learn that ℓ is not in σ_1. **[click]**
8. **[click]** now we apply the induction hypothesis, and solve the state ownership part by framing **[click] [click]**
9. **[click]** we are left with a pure goal **[click]** we can do pure intro to turn the goal into a plain Rocq goal **[click]**
10. **[click]** this requires us to prove **`ref`** v can reduce to ℓ in one base step
11. **[click]** which is true because ℓ is fresh
12. **[click]** this concludes the alloc case

If $e_1' = \texttt{ref}\ v$

Pure hyp

$\texttt{Hsafe1}: \mathsf{safe}_\varphi(K[\texttt{ref}\ v], \sigma_1)$ $\quad$ $\texttt{Hfresh}: \ell \notin \sigma_1$

$\texttt{Hbase}: (\texttt{ref}\ v, \sigma_1) \to_{\mathsf{base}} (\ell_0, \sigma_1[\ell_0 \leftarrow v])$

$\texttt{Hsafe2}: \mathsf{safe}_\varphi(K[\ell_0], \sigma_1[\ell_0 \leftarrow v])$

$\mathsf{safe}_\varphi(K[\ell], \sigma_1[\ell \leftarrow v])$

1. Second, if e_1' is alloc
2. **[click]** we use wp-alloc to **[click]** prove the first step **[click]**
3. **[click]** strip the later and intro **[click] [click]**
4. **[click]** note that this newly allocated location ℓ is unrelated to ℓ_0 resulting from destructing the base step
5. these two hypotheses are actually useless now, and we can discard them **[click] [click]**
6. **[click]** now we have the ownership of σ_1 and the newly allocated location
7. we can combine them **[click]** and meanwhile learn that ℓ is not in σ_1. **[click]**
8. **[click]** now we apply the induction hypothesis, and solve the state ownership part by framing **[click] [click]**
9. **[click]** we are left with a pure goal **[click]** we can do pure intro to turn the goal into a plain Rocq goal **[click]**
10. **[click]** this requires us to prove **`ref`** v can reduce to ℓ in one base step
11. **[click]** which is true because ℓ is fresh
12. **[click]** this concludes the alloc case

If $e_1' = \texttt{ref}\ v$

Pure hyp

$\texttt{Hsafe1}: \mathsf{safe}_\varphi(K[\texttt{ref}\ v], \sigma_1)$ $\qquad$ $\texttt{Hfresh}: \ell \notin \sigma_1$

$\texttt{Hbase}: (\texttt{ref}\ v, \sigma_1) \to_{\mathsf{base}} (\ell_0, \sigma_1[\ell_0 \leftarrow v])$

$\texttt{Hsafe2}: \mathsf{safe}_\varphi(K[\ell_0], \sigma_1[\ell_0 \leftarrow v])$

$\mathsf{safe}_\varphi(K[\ell], \sigma_1[\ell \leftarrow v])$

$\Uparrow$

$(\texttt{ref}\ v, \sigma_1) \to_{\mathsf{base}} (\ell, \sigma_1[\ell \leftarrow v])$

1. Second, if e_1' is alloc
2. **[click]** we use wp-alloc to **[click]** prove the first step **[click]**
3. **[click]** strip the later and intro **[click] [click]**
4. **[click]** note that this newly allocated location ℓ is unrelated to ℓ_0 resulting from destructing the base step
5. these two hypotheses are actually useless now, and we can discard them **[click] [click]**
6. **[click]** now we have the ownership of σ_1 and the newly allocated location
7. we can combine them **[click]** and meanwhile learn that ℓ is not in σ_1. **[click]**
8. **[click]** now we apply the induction hypothesis, and solve the state ownership part by framing **[click] [click]**
9. **[click]** we are left with a pure goal **[click]** we can do pure intro to turn the goal into a plain Rocq goal **[click]**
10. **[click]** this requires us to prove **`ref`** v can reduce to ℓ in one base step
11. **[click]** which is true because ℓ is fresh
12. **[click]** this concludes the alloc case

If $e_1' = \texttt{ref}\ v$

Pure hyp

$\texttt{Hsafe1}: \mathsf{safe}_\varphi(K[\texttt{ref}\ v], \sigma_1)$ $\texttt{Hfresh}: \ell \notin \sigma_1$

$\texttt{Hbase}: (\texttt{ref}\ v, \sigma_1) \to_{\mathsf{base}} (\ell_0, \sigma_1[\ell_0 \leftarrow v])$

$\texttt{Hsafe2}: \mathsf{safe}_\varphi(K[\ell_0], \sigma_1[\ell_0 \leftarrow v])$

$\mathsf{safe}_\varphi(K[\ell], \sigma_1[\ell \leftarrow v])$

$\Uparrow$

$(\texttt{ref}\ v, \sigma_1) \to_{\mathsf{base}} (\ell, \sigma_1[\ell \leftarrow v])$ ✓

(because **ref** chooses ℓ non-deterministically)

1. Second, if e_1' is alloc
2. **[click]** we use wp-alloc to **[click]** prove the first step **[click]**
3. **[click]** strip the later and intro **[click] [click]**
4. **[click]** note that this newly allocated location ℓ is unrelated to ℓ_0 resulting from destructing the base step
5. these two hypotheses are actually useless now, and we can discard them **[click] [click]**
6. **[click]** now we have the ownership of σ_1 and the newly allocated location
7. we can combine them **[click]** and meanwhile learn that ℓ is not in σ_1. **[click]**
8. **[click]** now we apply the induction hypothesis, and solve the state ownership part by framing **[click] [click]**
9. **[click]** we are left with a pure goal **[click]** we can do pure intro to turn the goal into a plain Rocq goal **[click]**
10. **[click]** this requires us to prove **`ref`** v can reduce to ℓ in one base step
11. **[click]** which is true because ℓ is fresh
12. **[click]** this concludes the alloc case

If $e_1' = \texttt{ref}\ v$

Pure hyp

$\texttt{Hsafe1}: \mathsf{safe}_\varphi(K[\texttt{ref}\ v], \sigma_1)$ $\quad$ $\texttt{Hfresh}: \ell \notin \sigma_1$

$\texttt{Hbase}: (\texttt{ref}\ v, \sigma_1) \to_{\mathsf{base}} (\ell_0, \sigma_1[\ell_0 \leftarrow v])$

$\texttt{Hsafe2}: \mathsf{safe}_\varphi(K[\ell_0], \sigma_1[\ell_0 \leftarrow v])$

$\mathsf{safe}_\varphi(K[\ell], \sigma_1[\ell \leftarrow v])$

$\Uparrow$

- ☑ $e_1' \to_{\mathsf{pure}} e_2'$
- ☑ $\sigma_1(\ell) = v \land (!\ell, \sigma_1) \to_{\mathsf{base}} (v, \sigma_1)$
- ☑ $\sigma_1(\ell) = v \land (\ell \leftarrow w, \sigma_1) \to_{\mathsf{base}} ((), \sigma_1[\ell \leftarrow w])$
- ☑ $\ell_0 \notin \sigma_1 \land (\texttt{ref}\ v, \sigma_1) \to_{\mathsf{base}} (\ell_0, \sigma_1[\ell_0 \leftarrow v])$

1. Second, if e_1' is alloc
2. **[click]** we use wp-alloc to **[click]** prove the first step **[click]**
3. **[click]** strip the later and intro **[click] [click]**
4. **[click]** note that this newly allocated location ℓ is unrelated to ℓ_0 resulting from destructing the base step
5. these two hypotheses are actually useless now, and we can discard them **[click] [click]**
6. **[click]** now we have the ownership of σ_1 and the newly allocated location
7. we can combine them **[click]** and meanwhile learn that ℓ is not in σ_1. **[click]**
8. **[click]** now we apply the induction hypothesis, and solve the state ownership part by framing **[click] [click]**
9. **[click]** we are left with a pure goal **[click]** we can do pure intro to turn the goal into a plain Rocq goal **[click]**
10. **[click]** this requires us to prove **`ref`** v can reduce to ℓ in one base step
11. **[click]** which is true because ℓ is fresh
12. **[click]** this concludes the alloc case

Recap

- Löb induction $\quad \triangleright \left(\forall e, \sigma.\, \ulcorner \mathsf{safe}_\varphi(e, \sigma) \urcorner \mathrel{-\!\!*} \mathsf{S}_\diamond(\sigma) \mathrel{-\!\!*} \mathsf{wp}\ e \left\{ v.\, \ulcorner \varphi(v) \urcorner \right\}\right)$
- If $\mathsf{safe}_\varphi(v, \sigma)$ ($v \in \mathit{Val}$) $\implies \ulcorner \varphi(v) \urcorner \vdash \mathsf{wp}\ v \left\{ v.\, \ulcorner \varphi(v) \urcorner \right\}$ (by $\textsc{WpValue}$)
- If $\mathsf{safe}_\varphi(e_1, \sigma_1)$ ($e_1 \notin \mathit{Val}$), then e_1 is reducible

  case analysis on the first step

1. to recap, to prove the completeness of a sequential program logic, we do Löb induction, prove the value case by wp-value, and prove the non-value case by case analysis on the first step
2. **[click]** note that except for the case analysis, everything we just saw is language-independent
3. **[click]** and for the language-dependent part, we update the state ownership using primitive reasoning rules in a way that is compatible with the operational semantics.
4. In fact, we can generalize this reasoning pattern to any sequential languages

Recap

- Löb induction $\triangleright \left(\forall e, \sigma. \ulcorner \mathsf{safe}_\varphi(e, \sigma) \urcorner \mathrel{-\!\!*} \mathsf{S}_\diamond(\sigma) \mathrel{-\!\!*} \mathsf{wp}\ e \left\{ v. \ulcorner \varphi(v) \urcorner \right\}\right)$
- If $\mathsf{safe}_\varphi(v, \sigma)$ ($v \in \mathit{Val}$) $\Longrightarrow$ $\ulcorner \varphi(v) \urcorner \vdash \mathsf{wp}\ v \left\{ v. \ulcorner \varphi(v) \urcorner \right\}$ (by $\textsc{WpValue}$)
- If $\mathsf{safe}_\varphi(e_1, \sigma_1)$ ($e_1 \notin \mathit{Val}$), then e_1 is reducible

case analysis on the first step

Language-independent, expect for the blue part

1. to recap, to prove the completeness of a sequential program logic, we do Löb induction, prove the value case by wp-value, and prove the non-value case by case analysis on the first step
2. **[click]** note that except for the case analysis, everything we just saw is language-independent
3. **[click]** and for the language-dependent part, we update the state ownership using primitive reasoning rules in a way that is compatible with the operational semantics.
4. In fact, we can generalize this reasoning pattern to any sequential languages

Recap

- Löb induction $\triangleright \left(\forall e, \sigma.\, \ulcorner \mathsf{safe}_\varphi(e, \sigma) \urcorner \mathrel{-\!\!*} \mathsf{S}_\diamond(\sigma) \mathrel{-\!\!*} \mathsf{wp}\ e \left\{ v.\, \ulcorner \varphi(v) \urcorner \right\}\right)$
- If $\mathsf{safe}_\varphi(v, \sigma)$ ($v \in \mathit{Val}$) $\Longrightarrow$ $\ulcorner \varphi(v) \urcorner \vdash \mathsf{wp}\ v \left\{ v.\, \ulcorner \varphi(v) \urcorner \right\}$ (by WpValue)
- If $\mathsf{safe}_\varphi(e_1, \sigma_1)$ ($e_1 \notin \mathit{Val}$), then e_1 is reducible

case analysis on the first step

Language-independent, expect for the blue part

For language-dependent part: update $\mathsf{S}_\diamond(\sigma)$ using primitive laws.

1. to recap, to prove the completeness of a sequential program logic, we do Löb induction, prove the value case by wp-value, and prove the non-value case by case analysis on the first step
2. **[click]** note that except for the case analysis, everything we just saw is language-independent
3. **[click]** and for the language-dependent part, we update the state ownership using primitive reasoning rules in a way that is compatible with the operational semantics.
4. In fact, we can generalize this reasoning pattern to any sequential languages

Concurrent Completeness

Sequential:

- Proof environment: $\mathsf{safe}_{\varphi}(e, \sigma)$ and $\mathsf{S}_{\circ}(\sigma)$
- One wp $e \left\{ v. \ulcorner \varphi(v) \urcorner \right\}$

1. let's now consider a concurrent language
2. for the sequential language, we keep safe and the state ownership in the proof environment, and we only need to prove one wp
3. **[click]** but in concurrent setting, we have fork
4. and to prove fork, we need to **[click]** prove two wp's
5. which means we need to share the ownership of the state among the two threads
6. **[click]** the quintessential Iris' solution is to put the state ownership and safety into an invariant, and prove a wp under the invariant
7. **[click]** we also need some ghost resources to ensure the expression we are currently processing is an expression in the invariant

Concurrent Completeness

Sequential:

- Proof environment: $\mathsf{safe}_\varphi(e, \sigma)$ and $\mathsf{S}_\circ(\sigma)$
- One wp $e \left\{ v. \ulcorner \varphi(v) \urcorner \right\}$

But in concurrent setting, we need to prove...

$$\frac{\text{Spatial hyp} \qquad \ulcorner \mathsf{safe\text{-}tp}_\varphi(\vec{e}, \sigma) \urcorner \qquad \mathsf{S}_\circ(\sigma)}{\text{Goal} \qquad \mathsf{wp}\ \texttt{fork}\ \{e_1\}\,;\, e_2 \left\{ v. \ulcorner \varphi(v) \urcorner \right\}} *$$

1. let's now consider a concurrent language
2. for the sequential language, we keep safe and the state ownership in the proof environment, and we only need to prove one wp
3. **[click]** but in concurrent setting, we have fork
4. and to prove fork, we need to **[click]** prove two wp's
5. which means we need to share the ownership of the state among the two threads
6. **[click]** the quintessential Iris' solution is to put the state ownership and safety into an invariant, and prove a wp under the invariant
7. **[click]** we also need some ghost resources to ensure the expression we are currently processing is an expression in the invariant

Concurrent Completeness

Sequential:

- Proof environment: $\mathsf{safe}_\varphi(e, \sigma)$ and $\mathsf{S}_\circ(\sigma)$
- One wp $e \left\{ v. \ulcorner \varphi(v) \urcorner \right\}$

But in concurrent setting, we need to prove...

$$\frac{\text{Spatial hyp} \qquad \ulcorner \mathsf{safe\text{-}tp}_\varphi(\vec{e}, \sigma) \urcorner \qquad \mathsf{S}_\circ(\sigma)}{\text{Goal} \qquad \mathsf{wp}\ e_1 \left\{ v.\ \mathsf{True} \right\} * \mathsf{wp}\ e_2 \left\{ v. \ulcorner \varphi(v) \urcorner \right\}} *$$

1. let's now consider a concurrent language
2. for the sequential language, we keep safe and the state ownership in the proof environment, and we only need to prove one wp
3. **[click]** but in concurrent setting, we have fork
4. and to prove fork, we need to **[click]** prove two wp's
5. which means we need to share the ownership of the state among the two threads
6. **[click]** the quintessential Iris' solution is to put the state ownership and safety into an invariant, and prove a wp under the invariant
7. **[click]** we also need some ghost resources to ensure the expression we are currently processing is an expression in the invariant

Concurrent Completeness

Sequential:

- Proof environment: $\mathsf{safe}_\varphi(e, \sigma)$ and $\mathsf{S}_\circ(\sigma)$
- One wp $e\left\{v.\ulcorner\varphi(v)\urcorner\right\}$

But in concurrent setting, we need to prove...

$$\frac{\text{Spatial hyp} \qquad \ulcorner\mathsf{safe\text{-}tp}_\varphi(\vec{e}, \sigma)\urcorner \qquad \mathsf{S}_\circ(\sigma)}{\text{Goal} \qquad \mathsf{wp}\ e_1\left\{v.\ \mathsf{True}\right\} * \mathsf{wp}\ e_2\left\{v.\ulcorner\varphi(v)\urcorner\right\}}\ *$$

$$I_{compl} \triangleq \boxed{\exists \vec{e}, \sigma.\ \ulcorner\mathsf{safe\text{-}tp}_\varphi(\vec{e}, \sigma)\urcorner * \mathsf{S}_\circ(\sigma)}^{\mathcal{N}}$$

$$\forall i, e.\ I_{compl} \qquad \vdash \mathsf{wp}\ e\left\{v.\ulcorner i = 0 \Rightarrow \varphi(v)\urcorner\right\}$$

1. let's now consider a concurrent language
2. for the sequential language, we keep safe and the state ownership in the proof environment, and we only need to prove one wp
3. **[click]** but in concurrent setting, we have fork
4. and to prove fork, we need to **[click]** prove two wp's
5. which means we need to share the ownership of the state among the two threads
6. **[click]** the quintessential Iris' solution is to put the state ownership and safety into an invariant, and prove a wp under the invariant
7. **[click]** we also need some ghost resources to ensure the expression we are currently processing is an expression in the invariant

Concurrent Completeness

Sequential:

- Proof environment: $\mathsf{safe}_\varphi(e, \sigma)$ and $\mathsf{S}_\circ(\sigma)$
- One wp $e\left\{v.\ulcorner\varphi(v)\urcorner\right\}$

But in concurrent setting, we need to prove...

$$\frac{\text{Spatial hyp} \qquad \ulcorner\mathsf{safe\text{-}tp}_\varphi(\vec{e}, \sigma)\urcorner \qquad \mathsf{S}_\circ(\sigma)}{\text{Goal} \qquad \mathsf{wp}\ e_1\left\{v.\ \mathsf{True}\right\} * \mathsf{wp}\ e_2\left\{v.\ulcorner\varphi(v)\urcorner\right\}}\ *$$

$$I_{compl} \triangleq \boxed{\exists \vec{e}, \sigma.\ \ulcorner\mathsf{safe\text{-}tp}_\varphi(\vec{e}, \sigma)\urcorner * \mathsf{S}_\circ(\sigma) * \bullet \vec{e}^{\,\gamma}}^{\mathcal{N}}$$

$$\forall i, e.\ I_{compl} * i \hookrightarrow^\gamma e \vdash \mathsf{wp}\ e\left\{v.\ulcorner i = 0 \Rightarrow \varphi(v)\urcorner\right\}$$

1. let's now consider a concurrent language
2. for the sequential language, we keep safe and the state ownership in the proof environment, and we only need to prove one wp
3. **[click]** but in concurrent setting, we have fork
4. and to prove fork, we need to **[click]** prove two wp's
5. which means we need to share the ownership of the state among the two threads
6. **[click]** the quintessential Iris' solution is to put the state ownership and safety into an invariant, and prove a wp under the invariant
7. **[click]** we also need some ghost resources to ensure the expression we are currently processing is an expression in the invariant

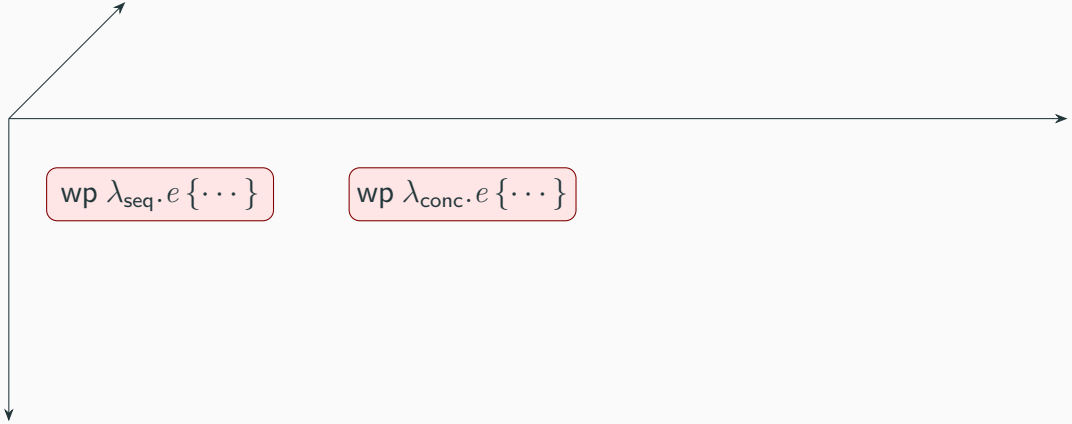
Generalization

$\mathsf{wp}\ \lambda_{\mathsf{seq}}.e\,\{\cdots\}$

$\mathsf{wp}\ \lambda_{\mathsf{conc}}.e\,\{\cdots\}$

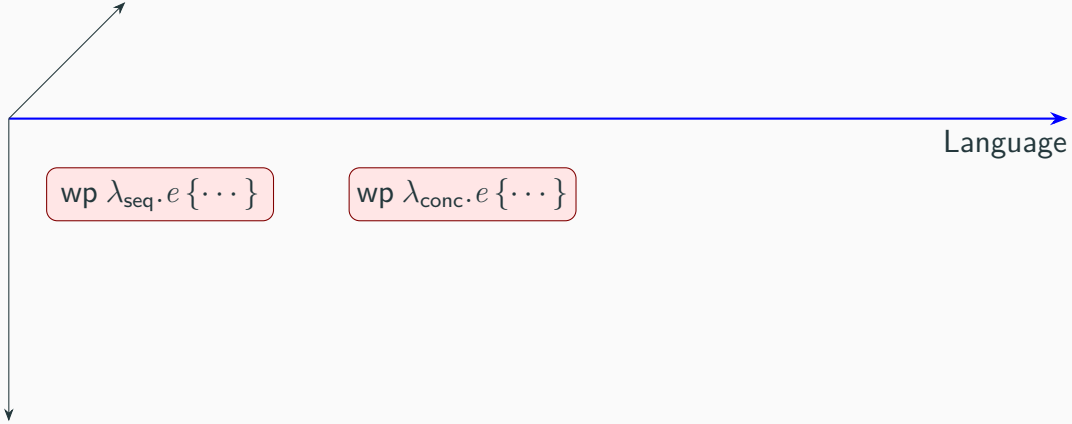
1. Now we have proved completeness for two forms of wps: wp for a sequential language, and wp for a concurrent language
2. But what if we want a more general result?
3. **[click]** I will generalize our results in three dimensions!
4. **[click]** First, and perhaps the most obvious, since we already considered the completeness two program logics and many parts of their proofs are language-independent. So why not prove a language-independent completeness theorem?
5. **[click]** We indeed generalized our completeness result to any language
6. **[click]** under some conditions of course
7. **[click]** Second, we generalize the strength of safety
8. Previously, we only considered partial correctness
9. **[click]** and in fact, if we substitute total wp to partial wp, and use well-founded induction instead of Löb induction, we can also prove a language-independent completeness theorem for total correctness
10. **[click]** Finally, all of these results are about behavior of one program, but what if we want to know the relative behaviors of two programs? This requires a relational logic, which can be built on top of the unary logic
11. **[click]** and we also proved a completeness result for a relational logic, and the completeness result is built on top of the unary completeness theorem

Generalization



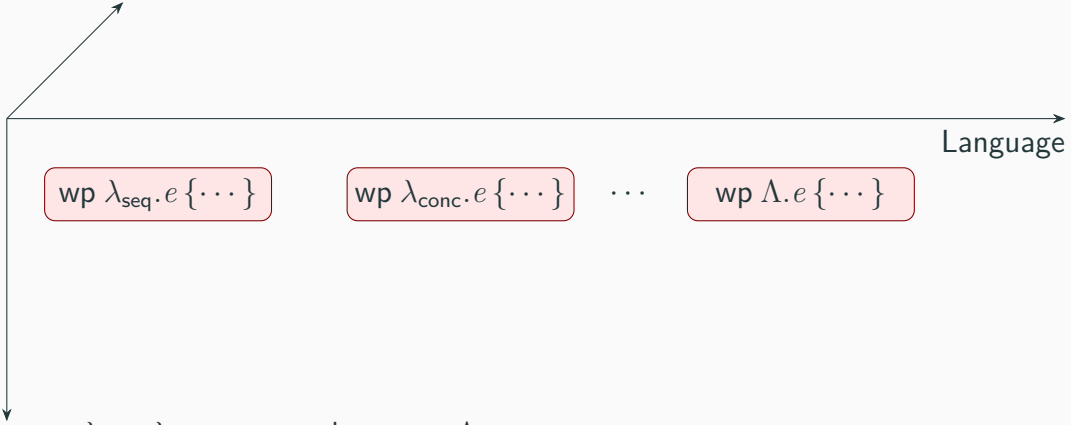
1. Now we have proved completeness for two forms of wps: wp for a sequential language, and wp for a concurrent language
2. But what if we want a more general result?
3. **[click]** I will generalize our results in three dimensions!
4. **[click]** First, and perhaps the most obvious, since we already considered the completeness two program logics and many parts of their proofs are language-independent. So why not prove a language-independent completeness theorem?
5. **[click]** We indeed generalized our completeness result to any language
6. **[click]** under some conditions of course
7. **[click]** Second, we generalize the strength of safety
8. Previously, we only considered partial correctness
9. **[click]** and in fact, if we substitute total wp to partial wp, and use well-founded induction instead of Löb induction, we can also prove a language-independent completeness theorem for total correctness
10. **[click]** Finally, all of these results are about behavior of one program, but what if we want to know the relative behaviors of two programs? This requires a relational logic, which can be built on top of the unary logic
11. **[click]** and we also proved a completeness result for a relational logic, and the completeness result is built on top of the unary completeness theorem

Generalization



1. Now we have proved completeness for two forms of wps: wp for a sequential language, and wp for a concurrent language
2. But what if we want a more general result?
3. **[click]** I will generalize our results in three dimensions!
4. **[click]** First, and perhaps the most obvious, since we already considered the completeness two program logics and many parts of their proofs are language-independent. So why not prove a language-independent completeness theorem?
5. **[click]** We indeed generalized our completeness result to any language
6. **[click]** under some conditions of course
7. **[click]** Second, we generalize the strength of safety
8. Previously, we only considered partial correctness
9. **[click]** and in fact, if we substitute total wp to partial wp, and use well-founded induction instead of Löb induction, we can also prove a language-independent completeness theorem for total correctness
10. **[click]** Finally, all of these results are about behavior of one program, but what if we want to know the relative behaviors of two programs? This requires a relational logic, which can be built on top of the unary logic
11. **[click]** and we also proved a completeness result for a relational logic, and the completeness result is built on top of the unary completeness theorem

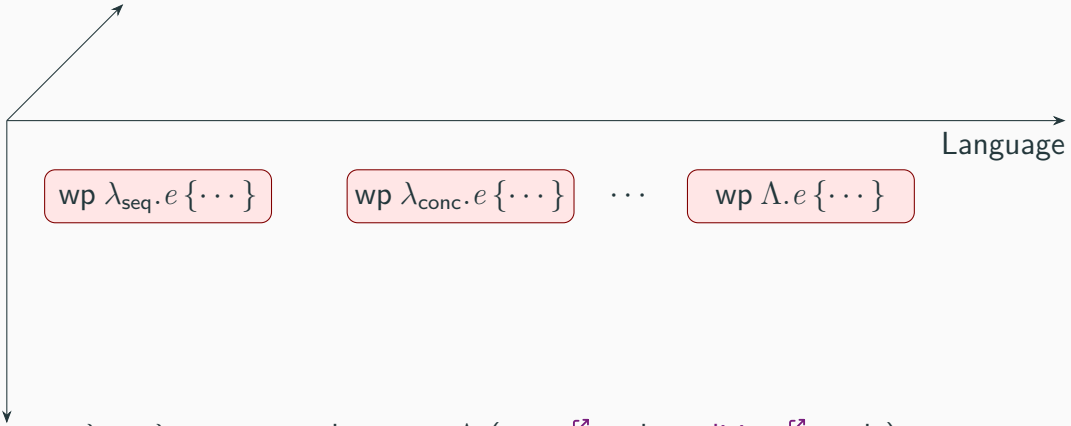
Generalization



- λ_{seq}, $\lambda_{\mathsf{con}} \longrightarrow$ any language Λ

1. Now we have proved completeness for two forms of wps: wp for a sequential language, and wp for a concurrent language
2. But what if we want a more general result?
3. **[click]** I will generalize our results in three dimensions!
4. **[click]** First, and perhaps the most obvious, since we already considered the completeness two program logics and many parts of their proofs are language-independent. So why not prove a language-independent completeness theorem?
5. **[click]** We indeed generalized our completeness result to any language
6. **[click]** under some conditions of course
7. **[click]** Second, we generalize the strength of safety
8. Previously, we only considered partial correctness
9. **[click]** and in fact, if we substitute total wp to partial wp, and use well-founded induction instead of Löb induction, we can also prove a language-independent completeness theorem for total correctness
10. **[click]** Finally, all of these results are about behavior of one program, but what if we want to know the relative behaviors of two programs? This requires a relational logic, which can be built on top of the unary logic
11. **[click]** and we also proved a completeness result for a relational logic, and the completeness result is built on top of the unary completeness theorem

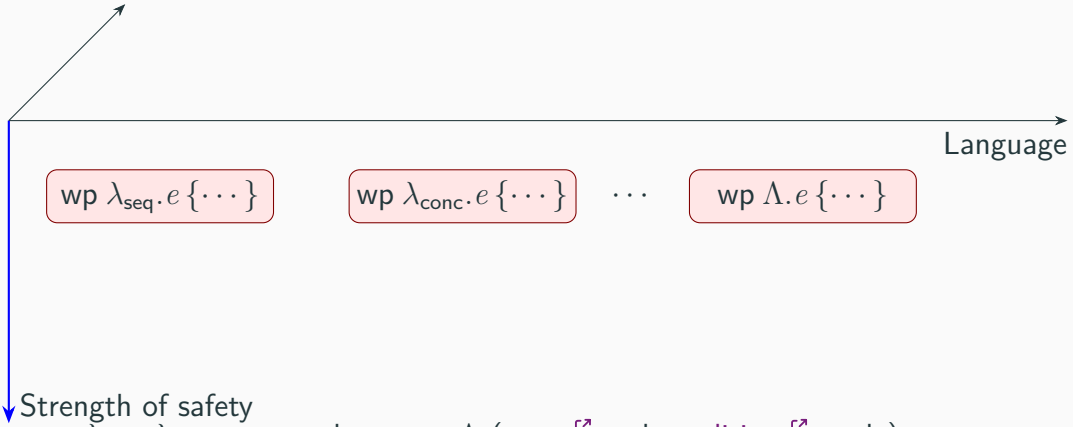
Generalization



- λ_{seq}, $\lambda_{\mathsf{con}} \longrightarrow$ any language Λ (terms and conditions apply)

1. Now we have proved completeness for two forms of wps: wp for a sequential language, and wp for a concurrent language
2. But what if we want a more general result?
3. **[click]** I will generalize our results in three dimensions!
4. **[click]** First, and perhaps the most obvious, since we already considered the completeness two program logics and many parts of their proofs are language-independent. So why not prove a language-independent completeness theorem?
5. **[click]** We indeed generalized our completeness result to any language
6. **[click]** under some conditions of course
7. **[click]** Second, we generalize the strength of safety
8. Previously, we only considered partial correctness
9. **[click]** and in fact, if we substitute total wp to partial wp, and use well-founded induction instead of Löb induction, we can also prove a language-independent completeness theorem for total correctness
10. **[click]** Finally, all of these results are about behavior of one program, but what if we want to know the relative behaviors of two programs? This requires a relational logic, which can be built on top of the unary logic
11. **[click]** and we also proved a completeness result for a relational logic, and the completeness result is built on top of the unary completeness theorem

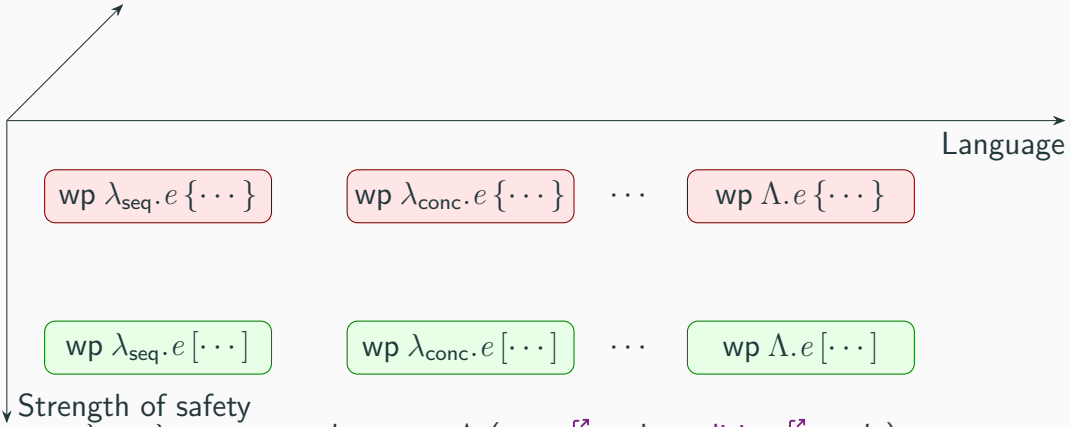
Generalization



- λ_{seq}, $\lambda_{\mathsf{con}} \longrightarrow$ any language Λ (terms and conditions apply)

1. Now we have proved completeness for two forms of wps: wp for a sequential language, and wp for a concurrent language
2. But what if we want a more general result?
3. **[click]** I will generalize our results in three dimensions!
4. **[click]** First, and perhaps the most obvious, since we already considered the completeness two program logics and many parts of their proofs are language-independent. So why not prove a language-independent completeness theorem?
5. **[click]** We indeed generalized our completeness result to any language
6. **[click]** under some conditions of course
7. **[click]** Second, we generalize the strength of safety
8. Previously, we only considered partial correctness
9. **[click]** and in fact, if we substitute total wp to partial wp, and use well-founded induction instead of Löb induction, we can also prove a language-independent completeness theorem for total correctness
10. **[click]** Finally, all of these results are about behavior of one program, but what if we want to know the relative behaviors of two programs? This requires a relational logic, which can be built on top of the unary logic
11. **[click]** and we also proved a completeness result for a relational logic, and the completeness result is built on top of the unary completeness theorem

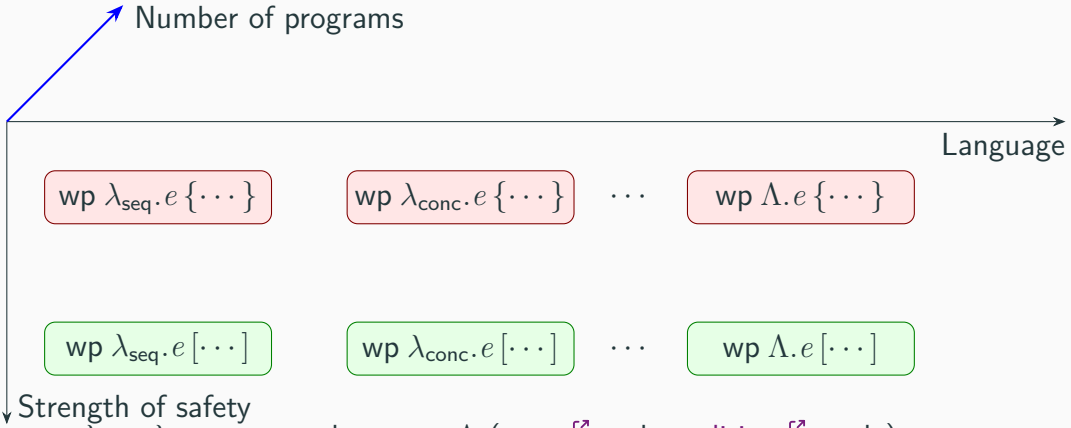
Generalization



- λ_{seq}, $\lambda_{\text{con}} \longrightarrow$ any language Λ (terms and conditions apply)
- wp $e\,\{\cdots\} \longrightarrow$ wp $e\,[\cdots]$ (Löb induction $\longrightarrow$ well-founded induction)

1. Now we have proved completeness for two forms of wps: wp for a sequential language, and wp for a concurrent language
2. But what if we want a more general result?
3. **[click]** I will generalize our results in three dimensions!
4. **[click]** First, and perhaps the most obvious, since we already considered the completeness two program logics and many parts of their proofs are language-independent. So why not prove a language-independent completeness theorem?
5. **[click]** We indeed generalized our completeness result to any language
6. **[click]** under some conditions of course
7. **[click]** Second, we generalize the strength of safety
8. Previously, we only considered partial correctness
9. **[click]** and in fact, if we substitute total wp to partial wp, and use well-founded induction instead of Löb induction, we can also prove a language-independent completeness theorem for total correctness
10. **[click]** Finally, all of these results are about behavior of one program, but what if we want to know the relative behaviors of two programs? This requires a relational logic, which can be built on top of the unary logic
11. **[click]** and we also proved a completeness result for a relational logic, and the completeness result is built on top of the unary completeness theorem

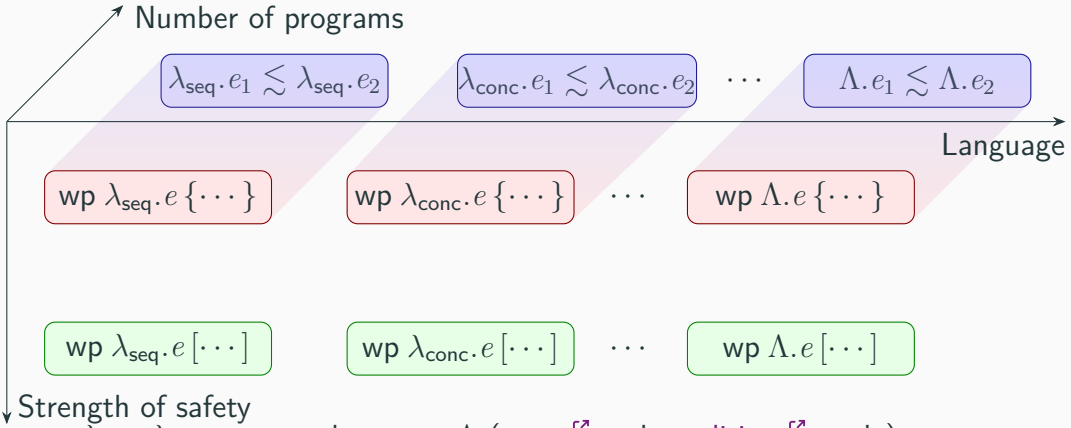
Generalization



- λ_{seq}, $\lambda_{\mathsf{con}} \longrightarrow$ any language Λ (terms and conditions apply)
- wp $e\,\{\cdots\} \longrightarrow$ wp $e\,[\cdots]$ (Löb induction $\longrightarrow$ well-founded induction)

1. Now we have proved completeness for two forms of wps: wp for a sequential language, and wp for a concurrent language
2. But what if we want a more general result?
3. **[click]** I will generalize our results in three dimensions!
4. **[click]** First, and perhaps the most obvious, since we already considered the completeness two program logics and many parts of their proofs are language-independent. So why not prove a language-independent completeness theorem?
5. **[click]** We indeed generalized our completeness result to any language
6. **[click]** under some conditions of course
7. **[click]** Second, we generalize the strength of safety
8. Previously, we only considered partial correctness
9. **[click]** and in fact, if we substitute total wp to partial wp, and use well-founded induction instead of Löb induction, we can also prove a language-independent completeness theorem for total correctness
10. **[click]** Finally, all of these results are about behavior of one program, but what if we want to know the relative behaviors of two programs? This requires a relational logic, which can be built on top of the unary logic
11. **[click]** and we also proved a completeness result for a relational logic, and the completeness result is built on top of the unary completeness theorem

Generalization



- λ_{seq}, $\lambda_{\text{con}} \longrightarrow$ any language Λ (terms and conditions apply)
- wp $e\,\{\cdots\} \longrightarrow$ wp $e\,[\cdots]$ (Löb induction $\longrightarrow$ well-founded induction)
- Unary $\longrightarrow$ binary (built on top of the unary completeness)

1. Now we have proved completeness for two forms of wps: wp for a sequential language, and wp for a concurrent language
2. But what if we want a more general result?
3. **[click]** I will generalize our results in three dimensions!
4. **[click]** First, and perhaps the most obvious, since we already considered the completeness two program logics and many parts of their proofs are language-independent. So why not prove a language-independent completeness theorem?
5. **[click]** We indeed generalized our completeness result to any language
6. **[click]** under some conditions of course
7. **[click]** Second, we generalize the strength of safety
8. Previously, we only considered partial correctness
9. **[click]** and in fact, if we substitute total wp to partial wp, and use well-founded induction instead of Löb induction, we can also prove a language-independent completeness theorem for total correctness
10. **[click]** Finally, all of these results are about behavior of one program, but what if we want to know the relative behaviors of two programs? This requires a relational logic, which can be built on top of the unary logic
11. **[click]** and we also proved a completeness result for a relational logic, and the completeness result is built on top of the unary completeness theorem

Condition 32 (Completeness).

$$\ulcorner \text{base-red}(e,\sigma)\urcorner * \mathrm{S}_\flat(\sigma) * \bullet^\gamma \vec{e} * \ulcorner \mathit{safe\text{-}tp}_\varphi(\vec{e},\sigma)\urcorner * n \hookrightarrow^\gamma K[e] \vdash \Rrightarrow_{\mathcal{E}}$$
$$\left(\ulcorner \text{Atomic } e\urcorner * \forall \Phi.\ \triangleright\ \mathit{WithStatePre}(\mathcal{E},e,n,K,\sigma,\vec{e},\Phi) \twoheadrightarrow \mathrm{wp}_{\mathcal{E}}\ e\ \{\Phi\}\right) \vee$$
$$\left(\mathrm{S}_\flat(\sigma) * \bullet^\gamma \vec{e} * \forall \Phi.\ \triangleright\ \mathit{WithoutStatePre}(\mathcal{E},e,K,\Phi) \twoheadrightarrow \mathrm{wp}_\top\ e\ \{\Phi\}\right)$$

where

$$\mathit{WithStatePre}(\mathcal{E},e,n,K,\sigma,\vec{e},\Phi) \triangleq \forall v',\sigma',\vec{e}_f.\ \ulcorner(e,\sigma) \to (v',\sigma',\vec{e}_f)\urcorner * \mathrm{S}_\flat(\sigma') * \bullet^\gamma \vec{e} * n \hookrightarrow^\gamma K[e] \twoheadrightarrow \Rrightarrow_{\mathcal{E}}$$
$$\Phi(v') * \mathop{\ast}_{e_f \in \vec{e}_f} \mathrm{wp}_\top\ e_f\ \{v.\ \mathrm{True}\}$$

$$\mathit{WithoutStatePre}(\mathcal{E},e,K,\Phi) \triangleq \forall e',\vec{e},\vec{e}_f.\ \mathit{Reduces}(K,e',\vec{e}) \twoheadrightarrow \Rrightarrow_\top$$
$$\mathrm{wp}_\top\ e'\ \{\Phi\} * \mathop{\ast}_{e_f \in \vec{e}_f} \mathrm{wp}_\top\ e_f\ \{v.\ \mathrm{True}\}$$

$$\mathit{Reduces}(e,K,e',\vec{e}) \triangleq \forall \sigma.\ \mathrm{S}_\flat(\sigma) * \bullet^\gamma \vec{e} * \ulcorner \mathit{safe\text{-}tp}_\varphi(\vec{e},\sigma)\urcorner \twoheadrightarrow \Rrightarrow_{\mathcal{E}}$$
$$\exists \sigma'.\ \ulcorner(e,\sigma) \to^+ (e',\sigma',\vec{e}_f)\urcorner * \mathrm{S}_\flat(\sigma') * \bullet^\gamma \vec{e} * n \hookrightarrow^\gamma K[e]$$

Completeness. If Condition 32 holds for a language Λ, then

$$\mathsf{safe}_\varphi(e,\varnothing) \implies \vdash \mathsf{wp}\ e\left\{v.\ \ulcorner\varphi(v)\urcorner\right\}$$

Strength of safety

- λ_{seq}, $\lambda_{\mathsf{con}} \longrightarrow$ any language Λ (terms and conditions apply)
- $\mathsf{wp}\ e\{\cdots\} \longrightarrow \mathsf{wp}\ e[\cdots]$ (Löb induction $\longrightarrow$ well-founded induction)
- Unary $\longrightarrow$ binary (built on top of the unary completeness)

1. **[click]** For those who really want to see the terms, here is the actual condition and the theorem statement.
2. **[click]** Let's ignore the condition, and focus on the theorem statement: it says if a program is safe under an empty state, then we can prove a wp of it.

Generalization

Completeness. If Condition 32 holds for a language Λ, then

$$\mathsf{safe}_\varphi(e, \varnothing) \implies \vdash \mathsf{wp}\ e\left\{v.\ulcorner\varphi(v)\urcorner\right\}$$

Strength of safety

- λ_{seq}, λ_{con} $\longrightarrow$ any language Λ (terms and conditions apply)
- wp $e\{\cdots\}$ $\longrightarrow$ wp $e[\cdots]$ (Löb induction $\longrightarrow$ well-founded induction)
- Unary $\longrightarrow$ binary (built on top of the unary completeness)

1. **[click]** For those who really want to see the terms, here is the actual condition and the theorem statement.
2. **[click]** Let's ignore the condition, and focus on the theorem statement: it says if a program is safe under an empty state, then we can prove a wp of it.

Case Studies

Completeness. If Condition 32 holds for a language Λ, then

$$\mathsf{safe}_{\varphi}(e, \varnothing) \implies \vdash \mathsf{wp}\ e \left\{ v. \ulcorner \varphi(v) \urcorner \right\}$$

Case studies: HeapLang, λ_{Rust}, time credits, and Eris (error credits)

1. Using generic completeness theorems, we do several case studies and proved the completeness of HeapLang, lambdaRust, time credits, and Eris.
2. **[click]** For lambdaRust, since its semantics non-atomic variants for load and store, and detects data races on non-atomic memory access, we need to cleverly defined S circle in a way that eliminates data race on non-atomic memory accesses.
3. **[click]** For Eris, since it has a non-standard safety definition, we cannot directly use the language-independent theorem itself. Nevertheless, we are still able to prove its completeness following the same pattern. Namely, do case analysis on whether the current expression is a value, and for non-value case, do further case analysis on the first reduction and use a primitive law to process each case.

Case Studies

Completeness. If Condition 32 holds for a language Λ, then

$$\mathsf{safe}_{\varphi}(e, \varnothing) \implies \vdash \mathsf{wp}\ e \left\{ v. \ulcorner \varphi(v) \urcorner \right\}$$

Case studies: HeapLang, λ_{Rust}, time credits, and Eris (error credits)

- λ_{Rust}: handle non-atomic memory access

1. Using generic completeness theorems, we do several case studies and proved the completeness of HeapLang, lambdaRust, time credits, and Eris.
2. **[click]** For lambdaRust, since its semantics non-atomic variants for load and store, and detects data races on non-atomic memory access, we need to cleverly defined S circle in a way that eliminates data race on non-atomic memory accesses.
3. **[click]** For Eris, since it has a non-standard safety definition, we cannot directly use the language-independent theorem itself. Nevertheless, we are still able to prove its completeness following the same pattern. Namely, do case analysis on whether the current expression is a value, and for non-value case, do further case analysis on the first reduction and use a primitive law to process each case.

Case Studies

Completeness. If Condition 32 holds for a language Λ, then

$$\mathsf{safe}_{\varphi}(e, \varnothing) \implies \vdash \mathsf{wp}\ e \left\{ v. \ulcorner \varphi(v) \urcorner \right\}$$

Case studies: HeapLang, λ_{Rust}, time credits, and Eris (error credits)

- λ_{Rust}: handle non-atomic memory access
- Eris: redo the completeness theorem following the same pattern

1. Using generic completeness theorems, we do several case studies and proved the completeness of HeapLang, lambdaRust, time credits, and Eris.
2. **[click]** For lambdaRust, since its semantics non-atomic variants for load and store, and detects data races on non-atomic memory access, we need to cleverly defined S circle in a way that eliminates data race on non-atomic memory accesses.
3. **[click]** For Eris, since it has a non-standard safety definition, we cannot directly use the language-independent theorem itself. Nevertheless, we are still able to prove its completeness following the same pattern. Namely, do case analysis on whether the current expression is a value, and for non-value case, do further case analysis on the first reduction and use a primitive law to process each case.

The missing rules in HeapLang

$$
\begin{aligned}
\mathit{unique} \triangleq\ & \texttt{let}\ \ell_1 = \texttt{ref}()\ \texttt{in} \\
& \texttt{free}\ \ell_1; \\
& \texttt{assert}\ \ell_1 \neq \texttt{ref}()
\end{aligned}
$$

1. I said proved HeapLang is complete, but I lied to you because it was incomplete, with several missing rules.
2. Let's first consider this program called unique. It allocates a locate l1, free it, and then allocate another location and asserts that it is not equal to l1.
3. **[click]** This program is safe because in HeapLang, free never actually removes the location from the heap, so new allocation cannot reuse freed locations.
4. **[click]** To prove its safety, we need a mechanism called meta token, which is a ghost state associated with a location, and will not be reclaimed at free
5. A meta-token is generated for every allocation. While most people just discard it, to prove the safety of unique, we need keep it to show the the uniqueness of locations
6. **[click]** If we have this exclusive law
7. **[click]** Unfortunately, such as a rule was missing.

The missing rules in HeapLang

$$unique \triangleq \texttt{let}\ \ell_1 = \texttt{ref}()\ \texttt{in}\ \texttt{free}\ \ell_1;\ \texttt{assert}\ \ell_1 \neq \texttt{ref}()$$

Safe because locations are never reused!

$$\frac{\ell \notin \mathsf{dom}(\sigma)}{(\texttt{ref}\ v, \sigma) \rightarrow_{\mathsf{base}} (\ell, \sigma[\ell \leftarrow v], \epsilon)}$$

$$\frac{\sigma(\ell) = v}{(\texttt{free}\ \ell, \sigma) \rightarrow_{\mathsf{base}} ((), \sigma[\ell \leftarrow \bot]\,\sigma, \epsilon)}$$

1. I said proved HeapLang is complete, but I lied to you because it was incomplete, with several missing rules.
2. Let's first consider this program called unique. It allocates a locate l1, free it, and then allocate another location and asserts that it is not equal to l1.
3. **[click]** This program is safe because in HeapLang, free never actually removes the location from the heap, so new allocation cannot reuse freed locations.
4. **[click]** To prove its safety, we need a mechanism called meta token, which is a ghost state associated with a location, and will not be reclaimed at free
5. A meta-token is generated for every allocation. While most people just discard it, to prove the safety of unique, we need keep it to show the the uniqueness of locations
6. **[click]** If we have this exclusive law
7. **[click]** Unfortunately, such as a rule was missing.

The missing rules in HeapLang

$$unique \triangleq \texttt{let}\ \ell_1 = \texttt{ref}()\ \texttt{in}\ \texttt{free}\ \ell_1;\ \texttt{assert}\ \ell_1 \neq \texttt{ref}()$$

Safe because locations are never reused!

Prove safety using meta tokens.

$$\frac{\ell \notin \mathsf{dom}(\sigma)}{(\texttt{ref}\ v, \sigma) \to_{\mathsf{base}} (\ell, \sigma[\ell \leftarrow v], \epsilon)}$$

$$\frac{\sigma(\ell) = v}{(\texttt{free}\ \ell, \sigma) \to_{\mathsf{base}} ((), \sigma[\ell \leftarrow \bot]\, \sigma, \epsilon)}$$

$$\{\mathsf{True}\}\ \texttt{ref}\ v\,\{\ell.\ \ell \mapsto v * \mathsf{metaTok}(\ell, \top)\}$$

$$\{\ell \mapsto v\}\ \texttt{free}\ \ell\,\{v.\ \ulcorner v = () \urcorner\}$$

1. I said proved HeapLang is complete, but I lied to you because it was incomplete, with several missing rules.
2. Let's first consider this program called unique. It allocates a locate l1, free it, and then allocate another location and asserts that it is not equal to l1.
3. **[click]** This program is safe because in HeapLang, free never actually removes the location from the heap, so new allocation cannot reuse freed locations.
4. **[click]** To prove its safety, we need a mechanism called meta token, which is a ghost state associated with a location, and will not be reclaimed at free
5. A meta-token is generated for every allocation. While most people just discard it, to prove the safety of unique, we need keep it to show the the uniqueness of locations
6. **[click]** If we have this exclusive law
7. **[click]** Unfortunately, such as a rule was missing.

The missing rules in HeapLang

$$unique \triangleq \texttt{let}\ \ell_1 = \texttt{ref}()\ \texttt{in}\ \texttt{free}\ \ell_1;\ \texttt{assert}\ \ell_1 \neq \texttt{ref}()$$

Safe because locations are never reused!

Prove safety using meta tokens.

$$\frac{\ell \notin \mathsf{dom}(\sigma)}{(\texttt{ref}\ v, \sigma) \to_{\mathsf{base}} (\ell, \sigma[\ell \leftarrow v], \epsilon)}$$

$$\frac{\sigma(\ell) = v}{(\texttt{free}\ \ell, \sigma) \to_{\mathsf{base}} ((), \sigma[\ell \leftarrow \bot]\,\sigma, \epsilon)}$$

$\{\mathsf{True}\}\ \texttt{ref}\ v\,\{\ell.\ \ell \mapsto v * \mathsf{metaTok}(\ell, \top)\}$

$\{\ell \mapsto v\}\ \texttt{free}\ \ell\,\{v.\ \ulcorner v = () \urcorner\}$

$$\frac{\mathsf{metaTok}(\ell_1, \top) * \mathsf{metaTok}(\ell_2, \top)}{\ulcorner \ell_1 \neq \ell_2 \urcorner}$$

1. I said proved HeapLang is complete, but I lied to you because it was incomplete, with several missing rules.
2. Let's first consider this program called unique. It allocates a locate l1, free it, and then allocate another location and asserts that it is not equal to l1.
3. **[click]** This program is safe because in HeapLang, free never actually removes the location from the heap, so new allocation cannot reuse freed locations.
4. **[click]** To prove its safety, we need a mechanism called meta token, which is a ghost state associated with a location, and will not be reclaimed at free
5. A meta-token is generated for every allocation. While most people just discard it, to prove the safety of unique, we need keep it to show the the uniqueness of locations
6. **[click]** If we have this exclusive law
7. **[click]** Unfortunately, such as a rule was missing.

The missing rules in HeapLang

$$unique \triangleq \texttt{let}\ \ell_1 = \texttt{ref}()\ \texttt{in}\ \texttt{free}\ \ell_1;\ \texttt{assert}\ \ell_1 \neq \texttt{ref}()$$

Safe because locations are never reused!

Prove safety using meta tokens.

$$\frac{\ell \notin \mathsf{dom}(\sigma)}{(\texttt{ref}\ v, \sigma) \to_{\mathsf{base}} (\ell, \sigma[\ell \leftarrow v], \epsilon)}$$

$$\frac{\sigma(\ell) = v}{(\texttt{free}\ \ell, \sigma) \to_{\mathsf{base}} ((), \sigma[\ell \leftarrow \bot]\,\sigma, \epsilon)}$$

$$\{\mathsf{True}\}\ \texttt{ref}\ v\,\{\ell.\ \ell \mapsto v * \mathsf{metaTok}(\ell, \top)\}$$

$$\{\ell \mapsto v\}\ \texttt{free}\ \ell\,\{v.\ \ulcorner v = () \urcorner\}$$

$$\frac{\mathsf{metaTok}(\ell_1, \top) * \mathsf{metaTok}(\ell_2, \top)}{\ulcorner \ell_1 \neq \ell_2 \urcorner}$$

(was missing)

1. I said proved HeapLang is complete, but I lied to you because it was incomplete, with several missing rules.
2. Let's first consider this program called unique. It allocates a locate l1, free it, and then allocate another location and asserts that it is not equal to l1.
3. **[click]** This program is safe because in HeapLang, free never actually removes the location from the heap, so new allocation cannot reuse freed locations.
4. **[click]** To prove its safety, we need a mechanism called meta token, which is a ghost state associated with a location, and will not be reclaimed at free
5. A meta-token is generated for every allocation. While most people just discard it, to prove the safety of unique, we need keep it to show the the uniqueness of locations
6. **[click]** If we have this exclusive law
7. **[click]** Unfortunately, such as a rule was missing.

Another Example

$$resolve \triangleq \texttt{resolve}\ (\texttt{LitProph}\ 1)\ \texttt{to}\ ()$$

1. The second example involves prophecy variables, but understanding it does not require any knowledge about prophecy variables. In this example, the program directly resolves a prophecy variable literal to unit, without allocating anything.
2. **[click]** This analogizes to directly storing to an location literal.
3. **[click]** Interestingly, although storing to an unallocated location is unsafe, resolving an unallocated prophecy variable is safe according to HeapLang semantics.
4. In HeapLang, resolve reduces to unit and makes an observation of (p,v), but it never checks the existence of the prophecy variable. This means, this code is always safe according to the operational semantics of HeapLang.
5. **[click]** This actually reveals a more fundamental issue, and fixing this requires changing the operational semantics. The new semantics checks p is allocated before making the observation.
6. **[click]** With these issues fixed, we proved the completeness for HeapLang.

Another Example

$resolve \triangleq \texttt{resolve}\ (\texttt{LitProph}\ 1)\ \texttt{to}\ ()$

Analogy: $(\texttt{LitLoc}\ 1) \leftarrow ()$

1. The second example involves prophecy variables, but understanding it does not require any knowledge about prophecy variables. In this example, the program directly resolves a prophecy variable literal to unit, without allocating anything.
2. **[click]** This analogizes to directly storing to an location literal.
3. **[click]** Interestingly, although storing to an unallocated location is unsafe, resolving an unallocated prophecy variable is safe according to HeapLang semantics.
4. In HeapLang, resolve reduces to unit and makes an observation of (p,v), but it never checks the existence of the prophecy variable. This means, this code is always safe according to the operational semantics of HeapLang.
5. **[click]** This actually reveals a more fundamental issue, and fixing this requires changing the operational semantics. The new semantics checks p is allocated before making the observation.
6. **[click]** With these issues fixed, we proved the completeness for HeapLang.

Another Example

$resolve \triangleq$ **resolve** (LitProph 1) **to** ()

Safe because

resolve does not check existence!

Analogy: (LitLoc 1) $\leftarrow$ ()

$$\frac{}{(\textbf{resolve}\ p\ \textbf{to}\ v, \sigma) \xrightarrow{[(p,((),v))]}_{\mathsf{base}} ((), \sigma, \varepsilon)}$$

1. The second example involves prophecy variables, but understanding it does not require any knowledge about prophecy variables. In this example, the program directly resolves a prophecy variable literal to unit, without allocating anything.
2. **[click]** This analogizes to directly storing to an location literal.
3. **[click]** Interestingly, although storing to an unallocated location is unsafe, resolving an unallocated prophecy variable is safe according to HeapLang semantics.
4. In HeapLang, resolve reduces to unit and makes an observation of (p,v), but it never checks the existence of the prophecy variable. This means, this code is always safe according to the operational semantics of HeapLang.
5. **[click]** This actually reveals a more fundamental issue, and fixing this requires changing the operational semantics. The new semantics checks p is allocated before making the observation.
6. **[click]** With these issues fixed, we proved the completeness for HeapLang.

Another Example

$resolve \triangleq$ **resolve** (LitProph 1) **to** ()

Safe because

resolve does not check existence!

Analogy: (LitLoc 1) $\leftarrow$ ()

$$\frac{p \in \sigma.pid}{(\textbf{resolve}\ p\ \textbf{to}\ v, \sigma) \xrightarrow{[(p,((),v))]}_{\mathsf{base}} ((), \sigma, \varepsilon)}$$

1. The second example involves prophecy variables, but understanding it does not require any knowledge about prophecy variables. In this example, the program directly resolves a prophecy variable literal to unit, without allocating anything.
2. **[click]** This analogizes to directly storing to an location literal.
3. **[click]** Interestingly, although storing to an unallocated location is unsafe, resolving an unallocated prophecy variable is safe according to HeapLang semantics.
4. In HeapLang, resolve reduces to unit and makes an observation of (p,v), but it never checks the existence of the prophecy variable. This means, this code is always safe according to the operational semantics of HeapLang.
5. **[click]** This actually reveals a more fundamental issue, and fixing this requires changing the operational semantics. The new semantics checks p is allocated before making the observation.
6. **[click]** With these issues fixed, we proved the completeness for HeapLang.

Another Example

$resolve \triangleq$ **resolve** (LitProph 1) **to** ()

Safe because

resolve does not check existence!

Completeness.

For a HeapLang expression e, $\mathsf{safe}_{\varphi}(e, \varnothing) \implies \vdash \mathsf{wp}\ e \left\{ v. \ulcorner \varphi(v) \urcorner \right\}$

$$\frac{p \in \sigma.pid}{(\textbf{resolve}\ p\ \textbf{to}\ v, \sigma) \xrightarrow{[(p,((),v))]}_{\mathsf{base}} ((), \sigma, \varepsilon)}$$

1. The second example involves prophecy variables, but understanding it does not require any knowledge about prophecy variables. In this example, the program directly resolves a prophecy variable literal to unit, without allocating anything.
2. **[click]** This analogizes to directly storing to an location literal.
3. **[click]** Interestingly, although storing to an unallocated location is unsafe, resolving an unallocated prophecy variable is safe according to HeapLang semantics.
4. In HeapLang, resolve reduces to unit and makes an observation of (p,v), but it never checks the existence of the prophecy variable. This means, this code is always safe according to the operational semantics of HeapLang.
5. **[click]** This actually reveals a more fundamental issue, and fixing this requires changing the operational semantics. The new semantics checks p is allocated before making the observation.
6. **[click]** With these issues fixed, we proved the completeness for HeapLang.

Insight: What Iris features are needed?

- Löb induction
- First-order ghost state, ghost maps
- Invariants of timeless assertions
- WpValue, WpBind, WpPure, FupdWp, and WpAtomic

1. To summarize the first part, let's discuss some lessons learned from proving completeness.
2. Although Iris provides a rich set of features, to prove completeness of whole program specifications, we only need a very few of them:
3. Namely, Löb induction, first-order ghost state, invariants of timeless assertions, and some basic rules on wp. In fact, for the ghost state, all we need is ghost maps.
4. **[click]** And we also need the reversed directions of wpvalue and wpbind, which are not so commonly used
5. **[click]** As for the conditions on a concrete language, the language semantics must satisfy some form of locality. For example, allocation must non-deterministically pick fresh locations, and resolve must ensure the prophecy variable exists
6. **[click]** Everything else is not used for completeness proofs. Notably, we don't need higher-order ghost state, impredicative invariants, and later credits.

Insight: What Iris features are needed?

- Löb induction
- First-order ghost state, ghost maps
- Invariants of timeless assertions
- WpValue, WpBind, WpPure, FupdWp, and WpAtomic
- WpValueInv and WpBindInv

1. To summarize the first part, let's discuss some lessons learned from proving completeness.
2. Although Iris provides a rich set of features, to prove completeness of whole program specifications, we only need a very few of them:
3. Namely, Löb induction, first-order ghost state, invariants of timeless assertions, and some basic rules on wp. In fact, for the ghost state, all we need is ghost maps.
4. **[click]** And we also need the reversed directions of wpvalue and wpbind, which are not so commonly used
5. **[click]** As for the conditions on a concrete language, the language semantics must satisfy some form of locality. For example, allocation must non-deterministically pick fresh locations, and resolve must ensure the prophecy variable exists
6. **[click]** Everything else is not used for completeness proofs. Notably, we don't need higher-order ghost state, impredicative invariants, and later credits.

Insight: What Iris features are needed?

- Löb induction
- First-order ghost state, ghost maps
- Invariants of timeless assertions
- WpValue, WpBind, WpPure, FupdWp, and WpAtomic
- WpValueInv and WpBindInv
- Locality of the semantics
  - Allocation must non-deterministically pick a location
  - Resolve must ensure the prophecy variable exists

1. To summarize the first part, let's discuss some lessons learned from proving completeness.
2. Although Iris provides a rich set of features, to prove completeness of whole program specifications, we only need a very few of them:
3. Namely, Löb induction, first-order ghost state, invariants of timeless assertions, and some basic rules on wp. In fact, for the ghost state, all we need is ghost maps.
4. **[click]** And we also need the reversed directions of wpvalue and wpbind, which are not so commonly used
5. **[click]** As for the conditions on a concrete language, the language semantics must satisfy some form of locality. For example, allocation must non-deterministically pick fresh locations, and resolve must ensure the prophecy variable exists
6. **[click]** Everything else is not used for completeness proofs. Notably, we don't need higher-order ghost state, impredicative invariants, and later credits.

Insight: What Iris features are needed?

- Löb induction
- First-order ghost state, ghost maps
- Invariants of timeless assertions
- WpValue, WpBind, WpPure, FupdWp, and WpAtomic
- WpValueInv and WpBindInv
- Locality of the semantics
  - Allocation must non-deterministically pick a location
  - Resolve must ensure the prophecy variable exists

We don't need

- Higher-order ghost state
- Impredicative invariants
- Later credits

1. To summarize the first part, let's discuss some lessons learned from proving completeness.
2. Although Iris provides a rich set of features, to prove completeness of whole program specifications, we only need a very few of them:
3. Namely, Löb induction, first-order ghost state, invariants of timeless assertions, and some basic rules on wp. In fact, for the ghost state, all we need is ghost maps.
4. **[click]** And we also need the reversed directions of wpvalue and wpbind, which are not so commonly used
5. **[click]** As for the conditions on a concrete language, the language semantics must satisfy some form of locality. For example, allocation must non-deterministically pick fresh locations, and resolve must ensure the prophecy variable exists
6. **[click]** Everything else is not used for completeness proofs. Notably, we don't need higher-order ghost state, impredicative invariants, and later credits.

Concurrent Library Completeness

Zichen Zhang Simon Gregersen Joseph Tassarotti

Logic & Truth for Concurrent Data Structures

Logic

Truth

1. Let's first go back to this picture
2. **[click]** For concurrent data structures, the truth is linearizability, which is the golden standard for concurrent data structures.
3. **[click]** And the logic counterpart is logically atomic triples
4. **[click]** For the soundness direction, Birkedal et al has proved logical atomicity implies linearizability
5. **[click]** But the completeness direction is again untouched

Logic & Truth for Concurrent Data Structures

Logic

Truth

$\mathsf{lin}_\mu(\mathsf{op})$

Linearizable *w.r.t.* sequential behavior μ

1. Let's first go back to this picture
2. **[click]** For concurrent data structures, the truth is linearizability, which is the golden standard for concurrent data structures.
3. **[click]** And the logic counterpart is logically atomic triples
4. **[click]** For the soundness direction, Birkedal et al has proved logical atomicity implies linearizability
5. **[click]** But the completeness direction is again untouched

Logic & Truth for Concurrent Data Structures

$I_\mu \vdash \langle P_\mu \rangle\ \mathsf{op}(obj, x)\ \langle y.\ Q_\mu(y) \rangle$

Logic

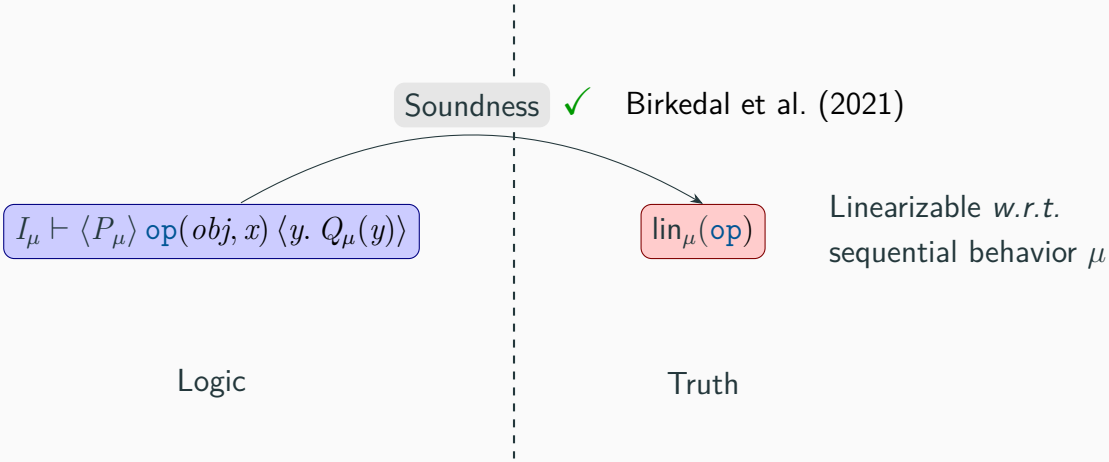
$\mathsf{lin}_\mu(\mathsf{op})$

Truth

Linearizable *w.r.t.* sequential behavior μ

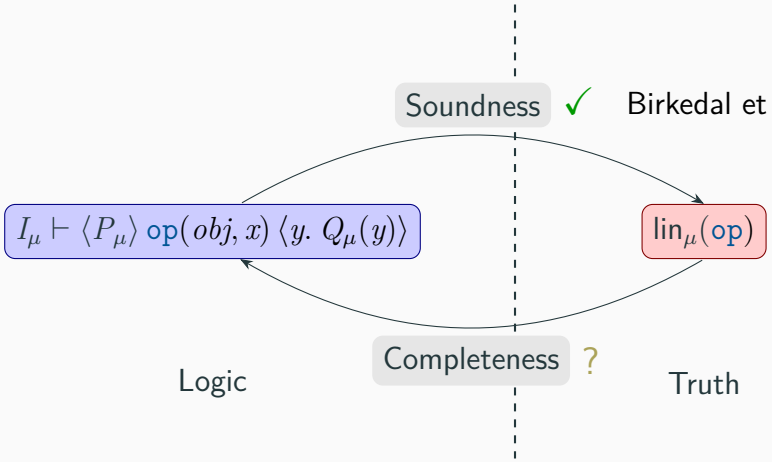
1. Let's first go back to this picture
2. **[click]** For concurrent data structures, the truth is linearizability, which is the golden standard for concurrent data structures.
3. **[click]** And the logic counterpart is logically atomic triples
4. **[click]** For the soundness direction, Birkedal et al has proved logical atomicity implies linearizability
5. **[click]** But the completeness direction is again untouched

Logic & Truth for Concurrent Data Structures



1. Let's first go back to this picture
2. **[click]** For concurrent data structures, the truth is linearizability, which is the golden standard for concurrent data structures.
3. **[click]** And the logic counterpart is logically atomic triples
4. **[click]** For the soundness direction, Birkedal et al has proved logical atomicity implies linearizability
5. **[click]** But the completeness direction is again untouched

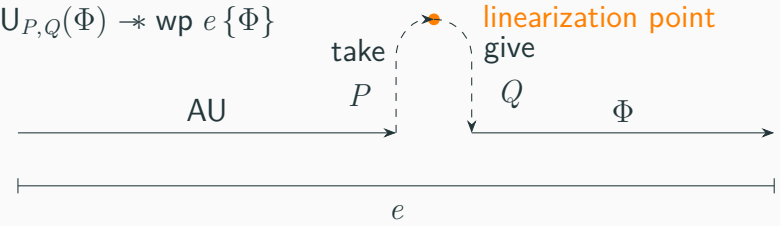
Logic & Truth for Concurrent Data Structures



1. Let's first go back to this picture
2. **[click]** For concurrent data structures, the truth is linearizability, which is the golden standard for concurrent data structures.
3. **[click]** And the logic counterpart is logically atomic triples
4. **[click]** For the soundness direction, Birkedal et al has proved logical atomicity implies linearizability
5. **[click]** But the completeness direction is again untouched

Key Ideas

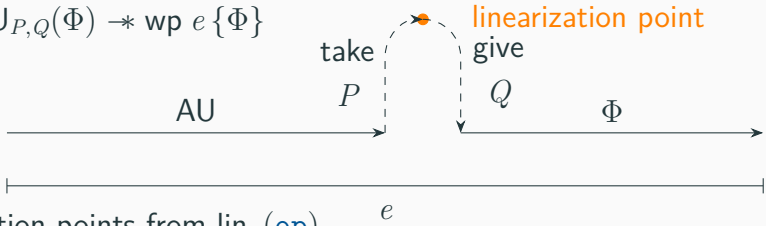
$\langle P \rangle\ e\ \langle Q \rangle \triangleq \forall \Phi.\ \mathsf{AU}_{P,Q}(\Phi) \twoheadrightarrow \mathsf{wp}\ e\ \{\Phi\}$



1. Before starting the proof, let's understand how logically atomic triples are defined.
2. They are defined using another assertion called AU, or atomic update
3. The idea is to prove a logically atomic triple of e, one must find a single physical step during the executing of e to fire the AU.
4. To fire an AU, the prover gets the precondition P from the AU, and after executing one step, it must provide the postcondition Q. In exchange, it gets a receipt Φ that allows it to conclude the proof.
5. **[click]** Go back to the completeness theorem. The ideas is simple: we can extract linearization points from the definition of linearizability and fire AUs at these points
6. But there are two challenges
7. **[click]** First, linearization points can be future dependent, meaning that given a partial trace, there could be multiple linearizations, but only one of them is compatible with the future. But the definition of logically atomic triples requires us to determine linearization points right now, and we cannot wait until the future
8. **[click]** Second, the linearization points extracted from linearizability are just some annotations to a trace, and they are not tied to any physical step. An AU can only be fired at specific physical step.
9. **[click]** The first challenge is solved by prophecy variables, more specially **[click]** typed prophecy variables on undecidable types, which requires the law of excluded middle
10. **[click]** The second challenge is solved by a technique called helping, which involves putting AUs into an invariant. This requires impredicative invariants and later credits.

Key Ideas

$\langle P \rangle\ e\ \langle Q \rangle \triangleq \forall \Phi.\ \mathsf{AU}_{P,Q}(\Phi) \twoheadrightarrow \mathsf{wp}\ e\ \{\Phi\}$

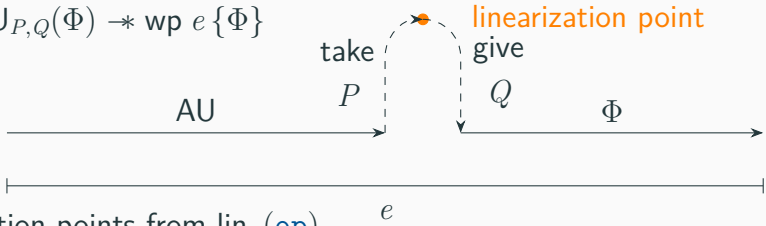


- Extract linearization points from $\mathsf{lin}_{\mu}(\mathtt{op})$
- "Fire" AU at the linearization points

1. Before starting the proof, let's understand how logically atomic triples are defined.
2. They are defined using another assertion called AU, or atomic update
3. The idea is to prove a logically atomic triple of e, one must find a single physical step during the executing of e to fire the AU.
4. To fire an AU, the prover gets the precondition P from the AU, and after executing one step, it must provide the postcondition Q. In exchange, it gets a receipt Φ that allows it to conclude the proof.
5. **[click]** Go back to the completeness theorem. The ideas is simple: we can extract linearization points from the definition of linearizability and fire AUs at these points
6. But there are two challenges
7. **[click]** First, linearization points can be future dependent, meaning that given a partial trace, there could be multiple linearizations, but only one of them is compatible with the future. But the definition of logically atomic triples requires us to determine linearization points right now, and we cannot wait until the future
8. **[click]** Second, the linearization points extracted from linearizability are just some annotations to a trace, and they are not tied to any physical step. An AU can only be fired at specific physical step.
9. **[click]** The first challenge is solved by prophecy variables, more specially **[click]** typed prophecy variables on undecidable types, which requires the law of excluded middle
10. **[click]** The second challenge is solved by a technique called helping, which involves putting AUs into an invariant. This requires impredicative invariants and later credits.

Key Ideas

$\langle P \rangle\ e\ \langle Q \rangle \triangleq \forall \Phi.\ \mathsf{AU}_{P,Q}(\Phi) \twoheadrightarrow \mathsf{wp}\ e\ \{\Phi\}$



- Extract linearization points from $\mathsf{lin}_{\mu}(\mathtt{op})$
- "Fire" AU at the linearization points

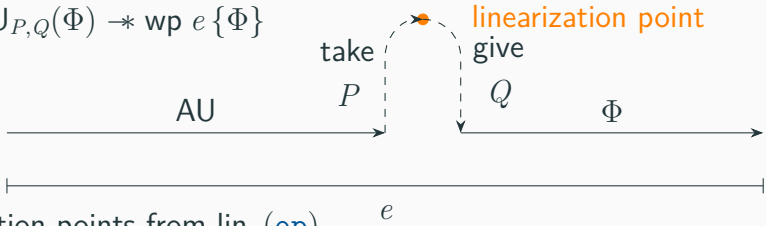
Challenges & Strategies

- Future dependency:

1. Before starting the proof, let's understand how logically atomic triples are defined.
2. They are defined using another assertion called AU, or atomic update
3. The idea is to prove a logically atomic triple of e, one must find a single physical step during the executing of e to fire the AU.
4. To fire an AU, the prover gets the precondition P from the AU, and after executing one step, it must provide the postcondition Q. In exchange, it gets a receipt Φ that allows it to conclude the proof.
5. **[click]** Go back to the completeness theorem. The ideas is simple: we can extract linearization points from the definition of linearizability and fire AUs at these points
6. But there are two challenges
7. **[click]** First, linearization points can be future dependent, meaning that given a partial trace, there could be multiple linearizations, but only one of them is compatible with the future. But the definition of logically atomic triples requires us to determine linearization points right now, and we cannot wait until the future
8. **[click]** Second, the linearization points extracted from linearizability are just some annotations to a trace, and they are not tied to any physical step. An AU can only be fired at specific physical step.
9. **[click]** The first challenge is solved by prophecy variables, more specially **[click]** typed prophecy variables on undecidable types, which requires the law of excluded middle
10. **[click]** The second challenge is solved by a technique called helping, which involves putting AUs into an invariant. This requires impredicative invariants and later credits.

Key Ideas

$\langle P \rangle\ e\ \langle Q \rangle \triangleq \forall \Phi.\ \mathsf{AU}_{P,Q}(\Phi) \twoheadrightarrow \mathsf{wp}\ e\ \{\Phi\}$



- Extract linearization points from $\mathsf{lin}_{\mu}(\mathtt{op})$
- "Fire" AU at the linearization points

Challenges & Strategies

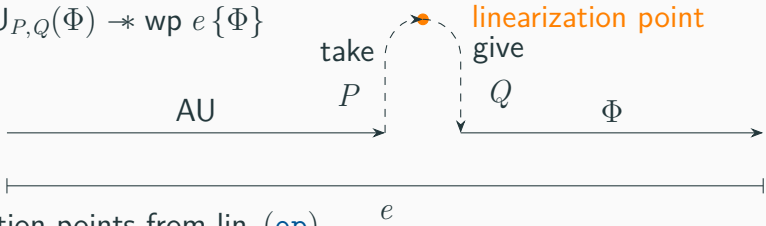
- Future dependency:

- Non structural linearization points:

1. Before starting the proof, let's understand how logically atomic triples are defined.
2. They are defined using another assertion called AU, or atomic update
3. The idea is to prove a logically atomic triple of e, one must find a single physical step during the executing of e to fire the AU.
4. To fire an AU, the prover gets the precondition P from the AU, and after executing one step, it must provide the postcondition Q. In exchange, it gets a receipt Φ that allows it to conclude the proof.
5. **[click]** Go back to the completeness theorem. The ideas is simple: we can extract linearization points from the definition of linearizability and fire AUs at these points
6. But there are two challenges
7. **[click]** First, linearization points can be future dependent, meaning that given a partial trace, there could be multiple linearizations, but only one of them is compatible with the future. But the definition of logically atomic triples requires us to determine linearization points right now, and we cannot wait until the future
8. **[click]** Second, the linearization points extracted from linearizability are just some annotations to a trace, and they are not tied to any physical step. An AU can only be fired at specific physical step.
9. **[click]** The first challenge is solved by prophecy variables, more specially **[click]** typed prophecy variables on undecidable types, which requires the law of excluded middle
10. **[click]** The second challenge is solved by a technique called helping, which involves putting AUs into an invariant. This requires impredicative invariants and later credits.

Key Ideas

$\langle P \rangle\ e\ \langle Q \rangle \triangleq \forall \Phi.\ \mathsf{AU}_{P,Q}(\Phi) \twoheadrightarrow \mathsf{wp}\ e\ \{\Phi\}$



- Extract linearization points from $\mathsf{lin}_{\mu}(\mathtt{op})$
- "Fire" AU at the linearization points

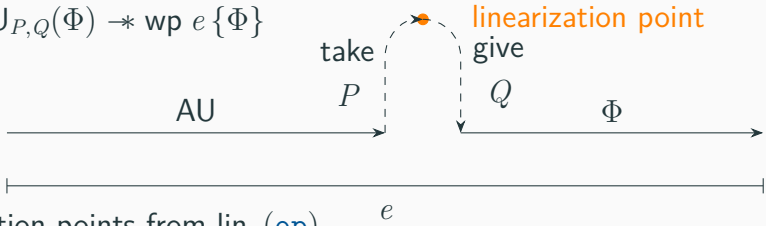
Challenges & Strategies

- Future dependency: prophecy variables
- Non structural linearization points:

1. Before starting the proof, let's understand how logically atomic triples are defined.
2. They are defined using another assertion called AU, or atomic update
3. The idea is to prove a logically atomic triple of e, one must find a single physical step during the executing of e to fire the AU.
4. To fire an AU, the prover gets the precondition P from the AU, and after executing one step, it must provide the postcondition Q. In exchange, it gets a receipt Φ that allows it to conclude the proof.
5. **[click]** Go back to the completeness theorem. The ideas is simple: we can extract linearization points from the definition of linearizability and fire AUs at these points
6. But there are two challenges
7. **[click]** First, linearization points can be future dependent, meaning that given a partial trace, there could be multiple linearizations, but only one of them is compatible with the future. But the definition of logically atomic triples requires us to determine linearization points right now, and we cannot wait until the future
8. **[click]** Second, the linearization points extracted from linearizability are just some annotations to a trace, and they are not tied to any physical step. An AU can only be fired at specific physical step.
9. **[click]** The first challenge is solved by prophecy variables, more specially **[click]** typed prophecy variables on undecidable types, which requires the law of excluded middle
10. **[click]** The second challenge is solved by a technique called helping, which involves putting AUs into an invariant. This requires impredicative invariants and later credits.

Key Ideas

$\langle P \rangle\ e\ \langle Q \rangle \triangleq \forall \Phi.\ \mathsf{AU}_{P,Q}(\Phi) \twoheadrightarrow \mathsf{wp}\ e\ \{\Phi\}$



- Extract linearization points from $\mathsf{lin}_{\mu}(\mathtt{op})$
- "Fire" AU at the linearization points

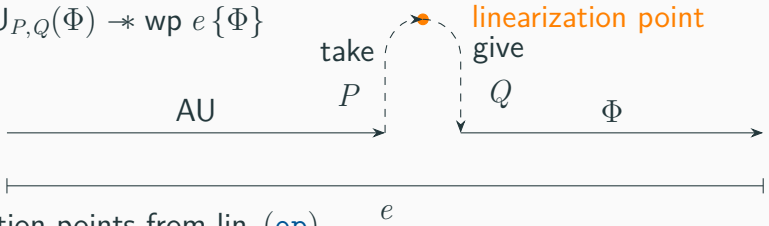
Challenges & Strategies

- Future dependency: typed prophecy variables on undecidable types
  needs excluded middle law
- Non structural linearization points:

1. Before starting the proof, let's understand how logically atomic triples are defined.
2. They are defined using another assertion called AU, or atomic update
3. The idea is to prove a logically atomic triple of e, one must find a single physical step during the executing of e to fire the AU.
4. To fire an AU, the prover gets the precondition P from the AU, and after executing one step, it must provide the postcondition Q. In exchange, it gets a receipt Φ that allows it to conclude the proof.
5. **[click]** Go back to the completeness theorem. The ideas is simple: we can extract linearization points from the definition of linearizability and fire AUs at these points
6. But there are two challenges
7. **[click]** First, linearization points can be future dependent, meaning that given a partial trace, there could be multiple linearizations, but only one of them is compatible with the future. But the definition of logically atomic triples requires us to determine linearization points right now, and we cannot wait until the future
8. **[click]** Second, the linearization points extracted from linearizability are just some annotations to a trace, and they are not tied to any physical step. An AU can only be fired at specific physical step.
9. **[click]** The first challenge is solved by prophecy variables, more specially **[click]** typed prophecy variables on undecidable types, which requires the law of excluded middle
10. **[click]** The second challenge is solved by a technique called helping, which involves putting AUs into an invariant. This requires impredicative invariants and later credits.

Key Ideas

$\langle P \rangle\ e\ \langle Q \rangle \triangleq \forall \Phi.\ \mathsf{AU}_{P,Q}(\Phi) \twoheadrightarrow \mathsf{wp}\ e\ \{\Phi\}$



- Extract linearization points from $\mathsf{lin}_{\mu}(\mathtt{op})$
- "Fire" AU at the linearization points

Challenges & Strategies

- Future dependency: typed prophecy variables on undecidable types
  needs excluded middle law
- Non structural linearization points: helping
  needs impredicative invariants and later credits

1. Before starting the proof, let's understand how logically atomic triples are defined.
2. They are defined using another assertion called AU, or atomic update
3. The idea is to prove a logically atomic triple of e, one must find a single physical step during the executing of e to fire the AU.
4. To fire an AU, the prover gets the precondition P from the AU, and after executing one step, it must provide the postcondition Q. In exchange, it gets a receipt Φ that allows it to conclude the proof.
5. **[click]** Go back to the completeness theorem. The ideas is simple: we can extract linearization points from the definition of linearizability and fire AUs at these points
6. But there are two challenges
7. **[click]** First, linearization points can be future dependent, meaning that given a partial trace, there could be multiple linearizations, but only one of them is compatible with the future. But the definition of logically atomic triples requires us to determine linearization points right now, and we cannot wait until the future
8. **[click]** Second, the linearization points extracted from linearizability are just some annotations to a trace, and they are not tied to any physical step. An AU can only be fired at specific physical step.
9. **[click]** The first challenge is solved by prophecy variables, more specially **[click]** typed prophecy variables on undecidable types, which requires the law of excluded middle
10. **[click]** The second challenge is solved by a technique called helping, which involves putting AUs into an invariant. This requires impredicative invariants and later credits.

Key Ideas

$\langle P \rangle\ e\ \langle Q \rangle \triangleq \forall \Phi.\ \mathsf{AU}_{P,Q}(\Phi) \twoheadrightarrow \mathsf{wp}\ e\ \{\Phi\}$

Completeness.
For a library op, $\mathsf{lin}_\mu(\mathtt{op}) \implies \forall obj, x.\ I_\mu \vdash \langle P_\mu \rangle\ \mathtt{op}(obj, x)\ \langle y.\ Q_\mu(y) \rangle$

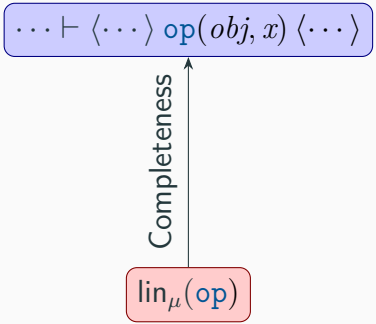
Challenges & Strategies

- Future dependency: typed prophecy variables on undecidable types
  needs excluded middle law
- Non structural linearization points: helping
  needs impredicative invariants and later credits

1. **[click]** With every challenge solved, we proved the completeness of logically atomic triples

Why completeness?

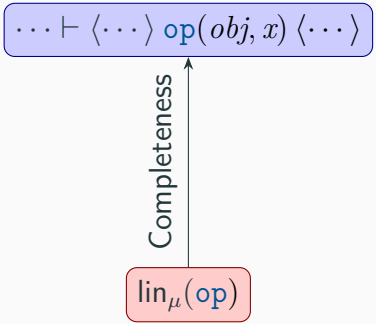
- Theoretically interesting



1. Let's step back and ask why we want completeness
2. first, it is of course theoretically interesting
3. but the completeness for a library specification has a very practically usage
4. **[click]** it allows us to embed the results shown by external techniques and obtain Iris specifications
5. for concurrent data structures specifically there has been decades of research on linearizability proofs
6. **[click]** to name a few of them, they include aspect-oriented proofs, forward simulation with commitment points, and meta-configuration tracking.
7. and they all imply linearizability
8. now with completeness, we can turn linearizability into a logical atomic triple and use it to verify large programs
9. **[click]** as an end-to-end case study, we proved the linearizability of herlihy and wing queue using all three techniques, and obtained the same logically atomic triple in three different ways.

Why completeness?

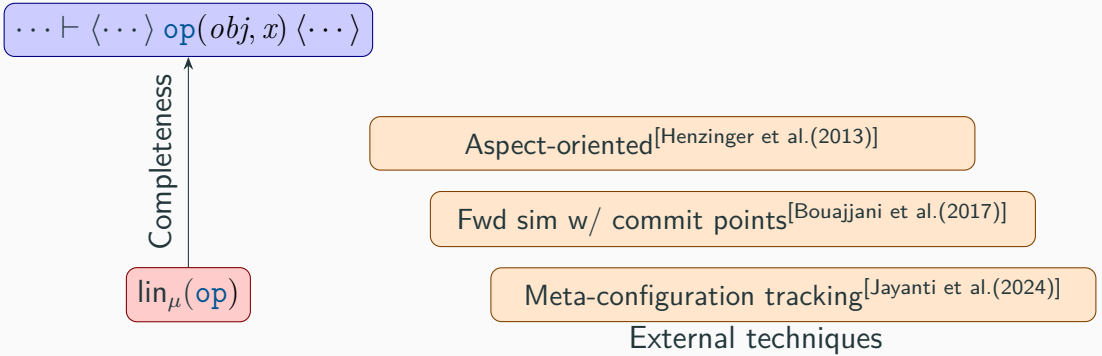
- Theoretically interesting
- Embedding external techniques



1. Let's step back and ask why we want completeness
2. first, it is of course theoretically interesting
3. but the completeness for a library specification has a very practically usage
4. **[click]** it allows us to embed the results shown by external techniques and obtain Iris specifications
5. for concurrent data structures specifically there has been decades of research on linearizability proofs
6. **[click]** to name a few of them, they include aspect-oriented proofs, forward simulation with commitment points, and meta-configuration tracking.
7. and they all imply linearizability
8. now with completeness, we can turn linearizability into a logical atomic triple and use it to verify large programs
9. **[click]** as an end-to-end case study, we proved the linearizability of herlihy and wing queue using all three techniques, and obtained the same logically atomic triple in three different ways.

Why completeness?

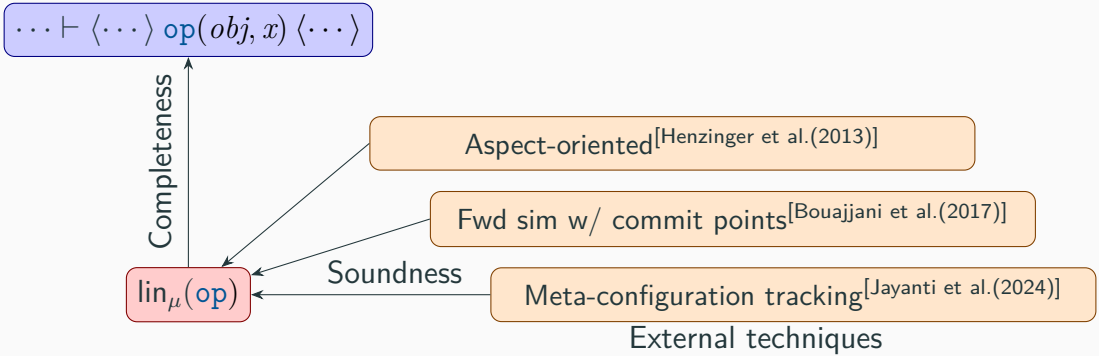
- Theoretically interesting
- Embedding external techniques



1. Let's step back and ask why we want completeness
2. first, it is of course theoretically interesting
3. but the completeness for a library specification has a very practically usage
4. **[click]** it allows us to embed the results shown by external techniques and obtain Iris specifications
5. for concurrent data structures specifically there has been decades of research on linearizability proofs
6. **[click]** to name a few of them, they include aspect-oriented proofs, forward simulation with commitment points, and meta-configuration tracking.
7. and they all imply linearizability
8. now with completeness, we can turn linearizability into a logical atomic triple and use it to verify large programs
9. **[click]** as an end-to-end case study, we proved the linearizability of herlihy and wing queue using all three techniques, and obtained the same logically atomic triple in three different ways.

Why completeness?

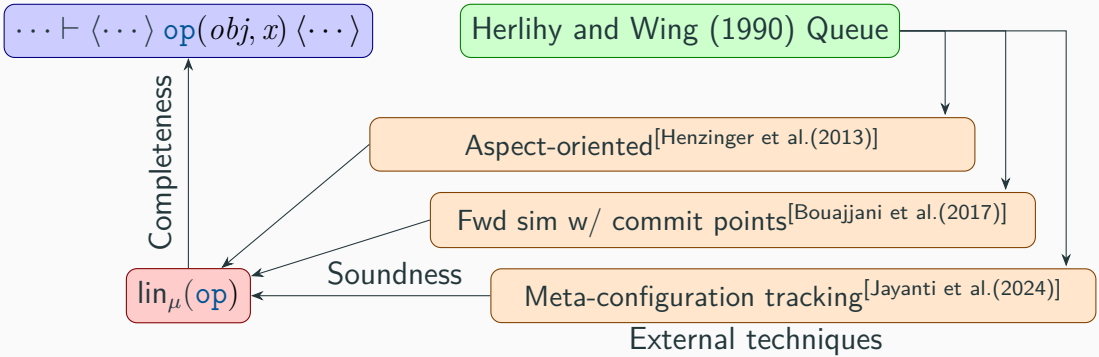
- Theoretically interesting
- Embedding external techniques



1. Let's step back and ask why we want completeness
2. first, it is of course theoretically interesting
3. but the completeness for a library specification has a very practically usage
4. **[click]** it allows us to embed the results shown by external techniques and obtain Iris specifications
5. for concurrent data structures specifically there has been decades of research on linearizability proofs
6. **[click]** to name a few of them, they include aspect-oriented proofs, forward simulation with commitment points, and meta-configuration tracking.
7. and they all imply linearizability
8. now with completeness, we can turn linearizability into a logical atomic triple and use it to verify large programs
9. **[click]** as an end-to-end case study, we proved the linearizability of herlihy and wing queue using all three techniques, and obtained the same logically atomic triple in three different ways.

Why completeness?

- Theoretically interesting
- Embedding external techniques



1. Let's step back and ask why we want completeness
2. first, it is of course theoretically interesting
3. but the completeness for a library specification has a very practically usage
4. **[click]** it allows us to embed the results shown by external techniques and obtain Iris specifications
5. for concurrent data structures specifically there has been decades of research on linearizability proofs
6. **[click]** to name a few of them, they include aspect-oriented proofs, forward simulation with commitment points, and meta-configuration tracking.
7. and they all imply linearizability
8. now with completeness, we can turn linearizability into a logical atomic triple and use it to verify large programs
9. **[click]** as an end-to-end case study, we proved the linearizability of herlihy and wing queue using all three techniques, and obtained the same logically atomic triple in three different ways.

Conclusion

Part 1.

- Language-independent whole program completeness theorems
- Case studies: HeapLang, λ_{Rust}, time credits, and Eris

1. To conclude, we developed language-independent completeness theorems for many forms of Iris-based program logics
2. Using these theorems, we proved the completeness of HeapLang, lambdaRust, time credits, and Eris
3. **[click]** In addition, we proved the completeness of logically atomic triples, and ported three external linearizability proof techniques.
4. **[click]** The first part of this talk will appear at this year's ICFP. The preprint and Rocq development are available, please try it out. And consider joining our discussion session if want to learn more.
5. This is the end of the talk, and thank everyone for attention!

Conclusion

Part 1.

- Language-independent whole program completeness theorems
- Case studies: HeapLang, λ_{Rust}, time credits, and Eris

Part 2.

- Completeness of logically atomic triples
- Ported external techniques: aspect-oriented proofs, forward simulation with commit points, and meta-configuration tracking

1. To conclude, we developed language-independent completeness theorems for many forms of Iris-based program logics
2. Using these theorems, we proved the completeness of HeapLang, lambdaRust, time credits, and Eris
3. **[click]** In addition, we proved the completeness of logically atomic triples, and ported three external linearizability proof techniques.
4. **[click]** The first part of this talk will appear at this year's ICFP. The preprint and Rocq development are available, please try it out. And consider joining our discussion session if want to learn more.
5. This is the end of the talk, and thank everyone for attention!

Conclusion

Part 1. (To appear at ICFP 2026)

- Language-independent whole program completeness theorems
- Case studies: HeapLang, λ_{Rust}, time credits, and Eris

Please try it out!

(and/or join our discussion session)

`https://plf.inf.ethz.ch/research/icfp26-completeness.html`

Part 2.

- Completeness of logically atomic triples
- Ported external techniques: aspect-oriented proofs, forward simulation with commit points, and meta-configuration tracking

1. To conclude, we developed language-independent completeness theorems for many forms of Iris-based program logics
2. Using these theorems, we proved the completeness of HeapLang, lambdaRust, time credits, and Eris
3. **[click]** In addition, we proved the completeness of logically atomic triples, and ported three external linearizability proof techniques.
4. **[click]** The first part of this talk will appear at this year's ICFP. The preprint and Rocq development are available, please try it out. And consider joining our discussion session if want to learn more.
5. This is the end of the talk, and thank everyone for attention!

References i

Lars Birkedal, Thomas Dinsdale-Young, Armaël Guéneau, Guilhem Jaber, Kasper Svendsen, and Nikos Tzevelekos. 2021.
Theorems for free from separation logic specifications.
Proc. ACM Program. Lang. 5, ICFP, Article 81 (Aug. 2021), 29 pages.
doi:10.1145/3473586

Ahmed Bouajjani, Michael Emmi, Constantin Enea, and Suha Orhun Mutluergil. 2017.
Proving Linearizability Using Forward Simulations. In *Computer Aided Verification*, Rupak Majumdar and Viktor Kunčak (Eds.). Springer International Publishing, Cham, 542–563.

References ii


Stephen A. Cook. 1978.
Soundness and Completeness of an Axiom System for Program Verification.
SIAM J. Comput. 7, 1 (1978), 70–90.
doi:10.1137/0207005


Thomas A. Henzinger, Ali Sezgin, and Viktor Vafeiadis. 2013.
Aspect-Oriented Linearizability Proofs. In *CONCUR 2013 – Concurrency Theory*, Pedro R. D'Argenio and Hernán Melgratti (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 242–256.

References iii

- Maurice P. Herlihy and Jeannette M. Wing. 1990.
  Linearizability: a correctness condition for concurrent objects.
  ACM Trans. Program. Lang. Syst. 12, 3 (July 1990), 463–492.
  doi:10.1145/78969.78972
- Prasad Jayanti, Siddhartha Jayanti, Ugur Yavuz, and Lizzie Hernandez. 2024.
  A Universal, Sound, and Complete Forward Reasoning Technique for Machine-Verified Proofs of Linearizability.
  Proc. ACM Program. Lang. 8, POPL, Article 82 (jan 2024), 29 pages.
  doi:10.1145/3632924

References iv


Cliff B. Jones. 1983.
Tentative Steps Toward a Development Method for Interfering Programs.
ACM Trans. Program. Lang. Syst. 5, 4 (1983), 596–619.
doi:10.1145/69575.69577


Susan S. Owicki. 1975.
Axiomatic Proof Techniques for Parallel Programs.
Technical Report. USA.

References v


Colin Stirling. 1988.
A Generalization of Owicki-Gries's Hoare Logic for a Concurrent while Language.
Theor. Comput. Sci. 58 (1988), 347–359.
doi:10.1016/0304-3975(88)90033-3