Is Peter Correct or Incorrect?

Patrick Cousot
Courant Institute, New York University
Peter’s Incorrectness Logic

• In POPL 2020, Peter O’Hearn introduced the nonconformist idea of an incorrectness logic

  We explore our hypothesis by defining incorrectness logic, a formalism that is similar to Hoare’s logic of program correctness [Hoare 1969], except that it is oriented to proving incorrectness rather than correctness.
Peter’s Incorrectness Logic

• In POPL 2020, Peter O’Hearn introduced the nonconformist idea of an incorrectness logic

  We explore our hypothesis by defining incorrectness logic, a formalism that is similar to Hoare’s logic of program correctness [Hoare 1969], except that it is oriented to proving incorrectness rather than correctness.

• Is it?
Peter’s Incorrectness Logic

• And he moderately enjoyed other approaches to incorrectness
• Such as `necessary preconditions”'

The concept of necessary precondition [Cousot et al. 2013] is related. A necessary precondition for a program is a predicate which, whenever falsified, leads to divergence or an error, but never to successful termination.
Peter’s Incorrectness Logic

• And he moderately enjoyed other approaches to incorrectness
• Such as “necessary preconditions”

The concept of necessary precondition [Cousot et al. 2013] is related. A necessary precondition for a program is a predicate which, whenever falsified, leads to divergence or an error, but never to successful termination.

• That he doesn’t really like

Finally, there are programs for which no non-trivial necessary pre-condition exists (e.g., skip + error()), but where perfectly fine presumptions exist for incorrectness logic.
Peter’s Incorrectness Logic

• And he moderately enjoyed other approaches to incorrectness
• Such as ``necessary preconditions’’

The concept of necessary precondition [Cousot et al. 2013] is related. A necessary precondition for a program is a predicate which, whenever falsified, leads to divergence or an error, but never to successful termination.

• That he doesn’t really like

Finally, there are programs for which no non-trivial necessary pre-condition exists (e.g., skip + error()), but where perfectly fine presumptions exist for incorrectness logic.

• Should he?
Peter’s Incorrectness Logic

In summary, there is a rich variety of problems for both experimental and theoretical work to bring the foundations of reasoning about program incorrectness onto a par with the extensively developed foundations for correctness.
Singularities of Logics
Emptiness versus Universality

- **Emptiness**: some programs satisfy no formula of the logic
  - Ex. 1: a potentially nonterminating satisfies no formula of the Manna-Pnueli total correctness logic
  - Ex. 2: Peter’s example for “necessary preconditions”
- **Universality**: some programs satisfy all formulae of the logic
  - $W = \text{while } (\text{true}) \text{ skip}$ satisfies all Hoare triples $\{P\} W \{Q\}$
  - i.e. false is always false and true is always true
Emptiness versus Universality

- **Emptiness**: some programs satisfy no formula of the logic
  - Ex. 1: a potentially nonterminating satisfies no formula of the Manna-Pnueli total correctness logic
  - Ex. 2: Peter’s example for “necessary preconditions”

- **Universality**: some programs satisfy all formulas of the logic
  - Ex. 1: \( W = \text{while (true) skip} \) satisfies all Hoare triples \( \{P\} W \{Q\} \)
    \[ \text{false is always false and true is always true} \]
Emptiness versus Universality

- **Emptiness**: some programs satisfy no formula of the logic
  - Ex. 1: a potentially nonterminating satisfies no formula of the Manna-Pnueli total correctness logic
  - Ex. 2: Peter’s example for "necessary preconditions"

- **Universality**: some programs satisfy all formulae of the logic
  - Ex. 1: $W = \text{while (true) skip}$ satisfies all Hoare triples $\{P\} W \{Q\}$
  - i.e. in logic false is never satisfied and true is always satisfied
Foundations of Reasoning on Logics
Method to design a program transformational logics

1. Define the natural relational semantics $\llbracket S \rrbracket_\perp$ of the programming language (in structural fixpoint form)

2. Define the theory of the logics as an abstraction $\alpha(\llbracket S \rrbracket_\perp)$ of the collecting semantics $\{\llbracket S \rrbracket_\perp\}$ (strongest (hyper) property)

3. Calculate the theory $\alpha(\llbracket S \rrbracket_\perp)$ in structural fixpoint form by fixpoint abstraction

4. Calculate the proof system by fixpoint induction and Aczel correspondence between fixpoints and deductive systems
Method to design a program transformational logics

1. Define the natural relational semantics $\llbracket S \rrbracket \bot$ of the programming language (in structural fixpoint form)

2. Define the theory of the logics as an abstraction $\alpha(\{\llbracket S \rrbracket \bot\})$ of the collecting semantics $\{\llbracket S \rrbracket \bot\}$ (strongest (hyper) property)

Theory of a logic = the subset of all true formulas
Method to design a program transformational logics

1. Define the natural relational semantics $\sem{S}_\bot$ of the programming language (in structural fixpoint form)

2. Define the theory of the logics as an abstraction $\alpha(\sem{S}_\bot)$ of the collecting semantics $\{\sem{S}_\bot\}$ (strongest (hyper) property)

3. Calculate the theory $\alpha(\sem{S}_\bot)$ in structural fixpoint form by fixpoint abstraction

Theory of a logic = the subset of all true formulas
Method to design a program transformational logics

1. Define the natural relational semantics \( \llbracket S \rrbracket \bot \) of the programming language (in structural fixpoint form)

2. Define the theory of the logics as an abstraction \( \alpha(\{\llbracket S \rrbracket \bot\}) \) of the collecting semantics \( \{\llbracket S \rrbracket \bot\} \) (strongest (hyper) property)

3. Calculate the theory \( \alpha(\{\llbracket S \rrbracket \bot\}) \) in structural fixpoint form by fixpoint abstraction

4. Calculate the proof system by fixpoint induction and Aczel correspondence between fixpoints and deductive systems

Theory of a logic = the subset of all true formulas
The Design of
Hoare Incorrectness Logic (HL)
I) Relational semantics
1. Angelic relational semantics $[[S]]^e$

- **Syntax**:  
  
  $$S \in S ::= x = A \mid \text{skip} \mid S;S \mid \text{if } (B) \ S \ \text{else } S \mid \text{while } (B) \ S$$

- **States**:  
  
  $\Sigma$

- **Angelic relational semantics**:  
  
  $$[[S]]^e \in \wp (\Sigma \times \Sigma)$$
I. Angelic relational semantics $[S]$ (in deductive form)

- Notations using judgements:
  - $\sigma \vdash S \Rightarrow^e \sigma'$ for $(\sigma, \sigma') \in [S]^e$
  - $\sigma \vdash \text{while}(B)\ S \Rightarrow^i \sigma'$ for $\sigma$ leads to $\sigma'$ after 0 or more iterations


1. Angelic relational semantics $\llbracket S \rrbracket$ (in deductive form)

- Notations using judgements:
  - $\sigma \vdash S \Rightarrow^e \sigma'$ for $\langle \sigma, \sigma' \rangle \in \llbracket S \rrbracket^e$
  - $\sigma \vdash \text{while}(B) S \Rightarrow \sigma'$ for $\sigma$ leads to $\sigma'$ after 0 or more iterations

- Semantics of the conditional iteration* $W = \text{while}(B) S$:

(a) $\sigma \vdash W \Rightarrow^i \sigma$

(b) $\frac{B[ B ] \sigma, \; \sigma \vdash S \Rightarrow^e \sigma', \; \sigma' \vdash W \Rightarrow^i \sigma''}{\sigma \vdash W \Rightarrow^i \sigma''}$

(c) $\frac{\sigma \vdash W \Rightarrow^i \sigma', \; B[ \neg B ] \sigma'}{\sigma \vdash W \Rightarrow^e \sigma'}$
I. Angelic relational semantics $\llbracket S \rrbracket$ (in fixpoint form)

- **Semantics of the conditional iteration**  
  \[ W = \text{while}(B) \ S : \]
  \[
  F^e(X) \triangleq \text{id} \cup ([B] ; [S]^e ; X), \quad X \in \varphi(\Sigma \times \Sigma) \\
  \llbracket \text{while } (B) \ S \rrbracket^e \triangleq \text{lfp} \subseteq F^e ; \llbracket \neg B \rrbracket
  \]

- Derived using Aczel correspondence between deductive systems and set-theoretic fixpoints (forthcoming)
II) Abstraction of the semantics to the theory
Exact abstractions
Abstraction

- Hyper properties to properties abstraction:

\[
\langle \varphi(\varphi(\Sigma \times \Sigma)), \subseteq \rangle \xleftrightarrow{\alpha_C} \langle \varphi(\Sigma \times \Sigma), \subseteq \rangle \quad \alpha_C(P) \doteq \bigcup P \quad \gamma_C(S) \doteq \varphi(S)
\]
Abstraction

- Hyper properties to properties abstraction:
  \[
  \langle \varnothing (\varnothing (\Sigma \times \Sigma)), \subseteq \rangle \xleftarrow{\gamma C} \alpha_C \xrightarrow{\alpha_C} \langle \varnothing (\Sigma \times \Sigma), \subseteq \rangle \quad \alpha_C(P) \triangleq \bigcup P \quad \gamma_C(S) \triangleq \varnothing (S)
  \]

- Post-image isomorphism:
  \[
  \langle \varnothing (\Sigma \times \Sigma), \subseteq \rangle \xleftarrow{\text{pre}} \xrightarrow{\text{post}} \langle \varnothing (\Sigma) \rightarrow \varnothing (\Sigma), \subseteq \rangle \quad \text{post}(R) \triangleq \lambda P \cdot \{ \sigma' \mid \exists \sigma \in P \land \langle \sigma, \sigma' \rangle \in R \}
  \]
  \[
  \text{pre}(R) \triangleq \lambda X \cdot \{ \sigma \mid \forall \sigma' \in Q \cdot \langle \sigma, \sigma' \rangle \in R \}
  \]
Abstraction

- **Hyper properties to properties abstraction:**

\[
\langle \varphi(\Sigma \times \Sigma) \rangle, \subseteq \xleftrightarrow{\gamma_C} \alpha_C \to \langle \varphi(\Sigma), \subseteq \rangle \quad \alpha_C(P) \doteq \bigcup P \quad \gamma_C(S) \doteq \varphi(S)
\]

- **Post-image isomorphism:**

\[
\langle \varphi(\Sigma \times \Sigma) \rangle, \subseteq \xleftarrow{\widehat{\text{post}}} \langle \varphi(\Sigma) \to \varphi(\Sigma), \subseteq \rangle \quad \text{post}(R) \doteq \lambda P \cdot \{ \sigma' \mid \exists \sigma \in P \land \langle \sigma, \sigma' \rangle \in R \}
\]

\[
\widehat{\text{pre}}(R) \doteq \lambda X \cdot \{ \sigma \mid \forall \sigma' \in Q \cdot \langle \sigma, \sigma' \rangle \in R \}
\]

- **Graph isomorphism** (a function is isomorphic to its graph, which is a function relation):

\[
\langle \varphi(\Sigma) \to \varphi(\Sigma), = \rangle \xleftarrow{\gamma_G} \langle \varphi_{\text{fun}}(\varphi(\Sigma) \times \varphi(\Sigma)), = \rangle \quad f \in \varphi(\Sigma) \to \varphi(\Sigma)
\]

\[
\alpha_G(f) = \{ \langle P, f(P) \rangle \mid P \in \varphi(\Sigma) \}
\]

\[
\gamma_G(R) \doteq \lambda P \cdot (Q \text{ such that } \langle P, S \rangle \in R)
\]
Abstraction

• Negation abstraction:

\[ X \in \wp(\mathcal{X}), \alpha^{-}(X) \triangleq \neg X \text{ (where } \neg X \triangleq \mathcal{X} \setminus X) \]

\[
\langle \wp(\mathcal{X}), \subseteq \rangle \xleftarrow{\alpha^{-}} \langle \wp(\mathcal{X}), \supseteq \rangle \quad \text{and} \quad \langle \wp(\mathcal{X}), \supseteq \rangle \xleftarrow{\alpha^{-}} \langle \wp(\mathcal{X}), \subseteq \rangle
\]
Consequence approximation
Approximation abstraction

• The component wise approximation:

\[ \langle x, y \rangle \sqsubseteq, \leq \langle x', y' \rangle \triangleq x \sqsubseteq x' \wedge y \leq y' \]
Approximation abstraction

- The component wise approximation:

\[ \langle x, y \rangle \sqsubseteq, \sqsubseteq \langle x', y' \rangle \iff x \sqsubseteq x' \land y \leq y' \]

- Over-approximation:

\[ \text{post}(\sqsubseteq, \sqsupseteq) = \lambda R \cdot \{ \{P, Q\} \mid \exists \{P', Q'\} \in R . P \sqsubseteq P' \land Q' \subseteq Q \} \]
Approximation abstraction

• The component wise approximation:

\[ \langle x, y \rangle \subseteq, \leq \langle x', y' \rangle \equiv x \subseteq x' \land y \leq y' \]

• Over-approximation:

\[ \text{post}(\subseteq, \supseteq) = \lambda R \cdot \{ \langle P, Q \rangle | \exists \langle P', Q' \rangle \in R . P \subseteq P' \land Q' \subseteq Q \} \]

• Under-approximation:

\[ \text{post}(\supseteq, \subseteq) = \lambda R \cdot \{ \langle P, Q \rangle | \exists \langle P', Q' \rangle \in R . P' \subseteq P \land Q \subseteq Q' \} \]
Comparing logics through their theories
Comparing logics through their theories

• Strongest postcondition logic (SL): \[ T(S) \triangleq \alpha_G \circ \text{post} \circ \alpha_C([S]) \]
  \[ = \{ (P, \text{post}[S]P) \mid P \in \wp(\Sigma) \} \]
Comparing logics through their theories

• Strongest postcondition logic (SL): \[ \mathcal{T}(s) \triangleq \alpha_G \circ \text{post} \circ \alpha_C(\{[S]\}) = \{\langle P, \text{post}[S]P \rangle | P \in \wp(\Sigma)\} \]

• Hoare logic (HL): \[ \mathcal{T}_{\text{HL}}(w) \triangleq \text{post}(\sqsupseteq) \circ \mathcal{T}(w) \]
Comparing logics through their theories

- Strongest postcondition logic (SL): \( \mathcal{T}(S) \triangleq \alpha_G \circ \text{post} \circ \alpha_C([\llbracket S \rrbracket]) \)
  \[ = \{ (P, \text{post}[S]P) \mid P \in \wp(\Sigma) \} \]

- Hoare logic (HL):
  \( \mathcal{T}_{\text{HL}}(W) \triangleq \text{post}(\geq \subseteq) \circ \mathcal{T}(W) \)

- Incorrectness logic (IL):
  \( \mathcal{T}_{\text{IL}}(S) \triangleq \text{post}(\subseteq \geq) \circ \mathcal{T}(S) \)
Comparing logics through their theories

- **Strongest postcondition logic (SL):** \( \mathcal{T}(S) \equiv \alpha_G \circ \text{post} \circ \alpha_C(\{S\}) \)
  \[ \equiv \{ \langle P, \text{post}[S]P \rangle \mid P \in \wp(\Sigma) \} \]

- **Hoare logic (HL):** \( \mathcal{T}_{HL}(W) \equiv \text{post}(\sqsupseteq \subseteq) \circ \mathcal{T}(W) \)

- **Incorrectness logic (IL):** \( \mathcal{T}_{IL}(S) \equiv \text{post}(\subseteq \sqsupseteq) \circ \mathcal{T}(S) \)

- **Hoare incorrectness logic (\(\overline{\text{HL}}\)):** \( \mathcal{T}_{HL}(W) = \text{post}(\subseteq, \sqsupseteq) \circ \alpha_{\neg} \circ \mathcal{T}_{HL}(W) \)

\[ \text{post} = \text{post} (\text{while } (B) \text{ S} \bar{\text{,}} \{ \langle P, \text{post}[S]P \rangle \mid P \in \wp(\Sigma) \}) \]
Comparing logics through their theories

Fig. 3. Hierarchical taxonomy of transformational assertional logics

O’Hearn Fest, POPL 2024, London
Fixpoint abstraction
2. Abstraction

• The abstraction of a fixpoint is a fixpoint (POPL 79)

**Theorem II.2.1 (Fixpoint Abstraction).** If \( \langle C, \sqsubseteq \rangle \iff \langle A, \preceq \rangle \) is a Galois connection between complete lattices \( \langle C, \sqsubseteq \rangle \) and \( \langle A, \preceq \rangle \), \( f \in C \xrightarrow{i} C \) and \( \bar{f} \in A \xrightarrow{i} A \) are increasing and commuting, that is, \( \alpha \circ f = \bar{f} \circ \alpha \), then \( \alpha(\text{lfp}^{\sqsubseteq} f) = \text{lfp}^{\preceq} \bar{f} \) (while semi-commutation \( \alpha \circ f \preceq \bar{f} \circ \alpha \) implies \( \alpha(\text{lfp}^{\sqsubseteq} f) \preceq \text{lfp}^{\preceq} \bar{f} \)).
2. Abstraction

• The abstraction of a fixpoint is a fixpoint (POPL 79)

**Theorem II.2.1 (Fixpoint abstraction).** If \( \langle C, \sqsubseteq \rangle \leftrightarrow_i^\alpha \langle A, \preceq \rangle \) is a Galois connection between complete lattices \( \langle C, \sqsubseteq \rangle \) and \( \langle A, \preceq \rangle \), \( f \in C \xrightarrow{i} C \) and \( \bar{f} \in A \xrightarrow{i} A \) are increasing and commuting, that is, \( \alpha \circ f = \bar{f} \circ \alpha \), then \( \alpha(lfp^\sqsubseteq f) = lfp^\preceq \bar{f} \) (while semi-commutation \( \alpha \circ f \preceq \bar{f} \circ \alpha \) implies \( \alpha(lfp^\sqsubseteq f) \preceq lfp^\preceq \bar{f} \)).

• We get a fixpoint definition of the theory of strongest postconditions logic (SL)

• For the iteration \( W = \text{while } (B) S : \)

\[
\mathcal{T}(W) \triangleq \{ \langle P, \text{post}[\neg B](lfp^\sqsubseteq \lambda X \cdot P \cup \text{post}([B]; [S]^e)X) \rangle \mid P \in \wp(\Sigma) \}
\]
1 PROPERTIES OF STRONGER POSTCONDITIONS

Lemma 1.1 (Composition). \( \text{post}(X ; Y) = \text{post}(Y) \cap \text{post}(X) \).

Proof of Lem. 1.1.

\begin{align*}
\text{post}(X ; Y) & = \{ P \mid \exists \sigma \in \mathcal{E} \exists \gamma \in \mathcal{S} \forall Q : \Gamma(X, \{ P \} ; Q, \{ Y \} ; \Gamma(x, \{ \gamma \} \cap \mathcal{E}^{\mathcal{S}})) \} \\
& = \{ P \mid \exists \sigma \in \mathcal{E} \exists \gamma \in \mathcal{S} \forall Q : \Gamma(X, \{ P \} ; Q, \{ Y \} ; \Gamma(x, \{ \gamma \} \cap \mathcal{E}^{\mathcal{S}})) \} \\
& = \{ P \mid \exists \sigma \in \mathcal{E} \exists \gamma \in \mathcal{S} \forall Q : \Gamma(X, \{ P \} ; Q, \{ Y \} ; \Gamma(x, \{ \gamma \} \cap \mathcal{E}^{\mathcal{S}})) \} \\
& = \{ P \mid \exists \sigma \in \mathcal{E} \exists \gamma \in \mathcal{S} \forall Q : \Gamma(X, \{ P \} ; Q, \{ Y \} ; \Gamma(x, \{ \gamma \} \cap \mathcal{E}^{\mathcal{S}})) \} \tag{\text{def. } \text{post}} \\
& = \{ P \mid \exists \sigma \in \mathcal{E} \exists \gamma \in \mathcal{S} \forall Q : \Gamma(X, \{ P \} ; Q, \{ Y \} ; \Gamma(x, \{ \gamma \} \cap \mathcal{E}^{\mathcal{S}})) \} \tag{\text{def. function composition} \gamma} \square
\end{align*}

Lemma 1.2 (Test). \( \text{post}([P \cap \mathcal{B}]_\mathcal{B}) \).

Proof of Lem. 1.2.

\begin{align*}
\text{post}([P \cap \mathcal{B}]_\mathcal{B}) & = \{ \exists \sigma \in \mathcal{E} \exists \gamma \in \mathcal{S} \forall Q : \Gamma(X, \{ P \cap \mathcal{B} \} ; Q, \{ \mathcal{B} \} ; \Gamma(x, \{ \gamma \} \cap \mathcal{E}^{\mathcal{S}})) \} \\
& = \{ \exists \sigma \in \mathcal{E} \exists \gamma \in \mathcal{S} \forall Q : \Gamma(X, \{ P \cap \mathcal{B} \} ; Q, \{ \mathcal{B} \} ; \Gamma(x, \{ \gamma \} \cap \mathcal{E}^{\mathcal{S}})) \} \\
& = \{ \exists \sigma \in \mathcal{E} \exists \gamma \in \mathcal{S} \forall Q : \Gamma(X, \{ P \cap \mathcal{B} \} ; Q, \{ \mathcal{B} \} ; \Gamma(x, \{ \gamma \} \cap \mathcal{E}^{\mathcal{S}})) \} \\
& = \{ \exists \sigma \in \mathcal{E} \exists \gamma \in \mathcal{S} \forall Q : \Gamma(X, \{ P \cap \mathcal{B} \} ; Q, \{ \mathcal{B} \} ; \Gamma(x, \{ \gamma \} \cap \mathcal{E}^{\mathcal{S}})) \} \tag{\text{def. } \text{post}} \\
& = \{ \exists \sigma \in \mathcal{E} \exists \gamma \in \mathcal{S} \forall Q : \Gamma(X, \{ P \cap \mathcal{B} \} ; Q, \{ \mathcal{B} \} ; \Gamma(x, \{ \gamma \} \cap \mathcal{E}^{\mathcal{S}})) \} \tag{\text{def. function composition} \gamma} \square
\end{align*}

Lemma 1.3 (Strongest Postcondition). \( \mathcal{T}(S) = \text{post}[S] = \{ (P, \text{post}[S]_P) \mid P \in S \} \).

Proof of Lem. 1.3.

\begin{align*}
\mathcal{T}(S) & = \{ (P, \text{post}[S]) \mid P \in S \} \\
& = \{ (P, \text{post}[S]) \mid P \in S \} \\
& = \{ (P, \text{post}[S]) \mid P \in S \} \\
& = \{ (P, \text{post}[S]) \mid P \in S \} \\
& = \{ (P, \text{post}[S]) \mid P \in S \} \tag{\text{def. } \text{post}} \\
& = \{ (P, \text{post}[S]) \mid P \in S \} \tag{\text{def. function composition} \gamma} \square
\end{align*}

Lemma 1.4 (Strongest Postcondition over Approximation). \( \mathcal{T}_\mathcal{E}(S) = \{ (P, \text{post}[S]_P) \mid P \in S \} \).

Proof of Lem. 1.4.

\begin{align*}
\mathcal{T}_\mathcal{E}(S) & = \{ (P, \text{post}[S]) \mid P \in S \} \\
& = \{ (P, \text{post}[S]) \mid P \in S \} \\
& = \{ (P, \text{post}[S]) \mid P \in S \} \\
& = \{ (P, \text{post}[S]) \mid P \in S \} \\
& = \{ (P, \text{post}[S]) \mid P \in S \} \tag{\text{def. } \text{post}} \\
& = \{ (P, \text{post}[S]) \mid P \in S \} \tag{\text{def. function composition} \gamma} \square
\end{align*}

Corollary 1.8 (Conditional iteration strongest postcondition graph). \( \mathcal{T}(W) = \{ (P, \text{post}[W]) \mid P \in \mathcal{E}[W] \} \).

Proof of Cor. 1.8.

\begin{align*}
\mathcal{T}(W) & = \{ (P, \text{post}[W]) \mid P \in \mathcal{E}[W] \} \\
& = \{ (P, \text{post}[W]) \mid P \in \mathcal{E}[W] \} \\
& = \{ (P, \text{post}[W]) \mid P \in \mathcal{E}[W] \} \\
& = \{ (P, \text{post}[W]) \mid P \in \mathcal{E}[W] \} \\
& = \{ (P, \text{post}[W]) \mid P \in \mathcal{E}[W] \} \tag{\text{def. } \text{post}} \\
& = \{ (P, \text{post}[W]) \mid P \in \mathcal{E}[W] \} \tag{\text{def. function composition} \gamma} \square
\end{align*}

For simplicity, we assume conditional iteration \((X \rightarrow Y) \cap \mathcal{B}[\mathcal{B}]_\mathcal{B} \) with \((X \rightarrow Y) \cap \mathcal{B}[\mathcal{B}]_\mathcal{B} \) no break.

Lemma 1.5 (Composition). \( \text{post}(P \times \mathcal{B}) = \text{post}(P) \cap \mathcal{B}[\mathcal{B}]_\mathcal{B} \).

Proof of Lem. 1.5.

\begin{align*}
\text{post}(P \times \mathcal{B}) & = \{ P \mid \exists \sigma \in \mathcal{E} \exists \gamma \in \mathcal{S} \forall Q : \Gamma(X, \{ P \} ; Q, \{ \mathcal{B} \} ; \Gamma(x, \{ \gamma \} \cap \mathcal{E}^{\mathcal{S}})) \} \\
& = \{ P \mid \exists \sigma \in \mathcal{E} \exists \gamma \in \mathcal{S} \forall Q : \Gamma(X, \{ P \} ; Q, \{ \mathcal{B} \} ; \Gamma(x, \{ \gamma \} \cap \mathcal{E}^{\mathcal{S}})) \} \tag{\text{def. } \text{post}} \\
& = \{ P \mid \exists \sigma \in \mathcal{E} \exists \gamma \in \mathcal{S} \forall Q : \Gamma(X, \{ P \} ; Q, \{ \mathcal{B} \} ; \Gamma(x, \{ \gamma \} \cap \mathcal{E}^{\mathcal{S}})) \} \tag{\text{def. function composition} \gamma} \square
\end{align*}

Corollary 1.9 (Pointwise commutation). \( \forall X \in \mathcal{E}[\mathcal{E} \rightarrow \mathcal{E}] \rightarrow \mathcal{E}(X) \).

Proof of Cor. 1.9.

\begin{align*}
\forall X \in \mathcal{E}[\mathcal{E} \rightarrow \mathcal{E}] \rightarrow \mathcal{E}(X) & = \{ \exists \sigma \in \mathcal{E} \exists \gamma \in \mathcal{S} \forall Q : \Gamma(X, \{ \gamma \} \cap \mathcal{E}^{\mathcal{S}})) \} \\
& = \{ \exists \sigma \in \mathcal{E} \exists \gamma \in \mathcal{S} \forall Q : \Gamma(X, \{ \gamma \} \cap \mathcal{E}^{\mathcal{S}})) \} \tag{\text{def. } \text{post}} \\
& = \{ \exists \sigma \in \mathcal{E} \exists \gamma \in \mathcal{S} \forall Q : \Gamma(X, \{ \gamma \} \cap \mathcal{E}^{\mathcal{S}})) \} \tag{\text{def. function composition} \gamma} \square
\end{align*}

Theorem 1.7 (Iteration strongest postcondition). \( \text{post}[\mathcal{B}] = \text{post}[-E][\mathcal{B}[\mathcal{B}]_\mathcal{B}] \).

Proof of Th. 1.7.

\begin{align*}
\text{post}[\mathcal{B}] & = \text{post}([\mathcal{B}[\mathcal{B}]_\mathcal{B}]_\mathcal{B}) \tag{\text{def. } \text{post}} \\
& = \text{post}([\mathcal{B}[\mathcal{B}]_\mathcal{B}]_\mathcal{B}) \tag{\text{def. function composition} \gamma} \square
\end{align*}
IV) Design of the proof system
Aczel correspondence
Aczel correspondence between deductive systems and fixpoints

• Rules: $\frac{P}{c}$ (\(\mathcal{U}\) universe, \(P \in \wp_{\text{fin}}(\mathcal{U})\) premiss, \(c \in \mathcal{U}\) conclusion, \(\emptyset\) axiom)
Aczel correspondence between deductive systems and fixpoints

- Rules: \( \frac{P}{c} \) (\( \mathcal{U} \) universe, \( P \in \wp_{\text{fin}}(\mathcal{U}) \) premiss, \( c \in \mathcal{U} \) conclusion, \( \emptyset \) axiom)
- Deductive system: \( R = \{ \frac{P_i}{c_i} \mid i \in \Delta \} \), \( R \in \wp(\wp_{\text{fin}}(\mathcal{U}) \times \mathcal{U}) \)
Aczel correspondence between deductive systems and fixpoints

- Rules: $\frac{P}{c}$ (U universe, $P \in \wp_{\text{fin}}(U)$ premiss, $c \in U$ conclusion, $\emptyset$ axiom)

- Deductive system: $R = \left\{ \frac{P_i}{c_i} \mid i \in \Delta \right\}$, $R \in \wp(\wp_{\text{fin}}(U) \times U)$

- Subset of the universe $U$ defined by $R$:

$$F(R) \triangleq \left\{ c \mid \exists \frac{P}{c} \in R. P \subseteq X \right\}$$

$$= \{ t_n \in U \mid \exists t_1, \ldots, t_{n-1} \in U. \forall k \in [1, n]. \exists \frac{P}{c} \in R. P \subseteq \{ t_1, \ldots, t_{k-1} \} \land t_k = c \}$$

$\downarrow$ proof theoretic

$\triangleq \text{lfp} \subseteq F(R)$

$\triangleq \{ c \mid \exists \frac{P}{c} \in R. P \subseteq X \}$

$\leftarrow$ model theoretic (gfp for coinduction)

$\downarrow$ consequence operator
Aczel correspondence between deductive systems and fixpoints

- **Rules**: \( \frac{P}{c} \) (\( U \) universe, \( P \in \wp_{\text{fin}}(U) \) premiss, \( c \in U \) conclusion, \( \emptyset \) axiom)

- **Deductive system**: \( R = \{ \frac{P_i}{c_i} \mid i \in \Delta \} \), \( R \in \wp (\wp_{\text{fin}}(U) \times U) \)

- **Subset of the universe \( U \)** defined by \( R \):
  \[
  \begin{align*}
  & \{ t_n \in U \mid \exists t_1, \ldots, t_{n-1} \in U . \forall k \in [1, n] . \exists \frac{P}{c} \in R . \, P \subseteq \{ t_1, \ldots, t_{k-1} \} \land t_k = c \} \\
  & = \text{lfp} \subseteq F(R) \\
  & F(R)X \triangleq \{ c \mid \exists \frac{P}{c} \in R . \, P \subseteq X \} \quad \leftarrow \text{model theoretic (gfp for coinduction)}
  \end{align*}
  \]

- **Deductive system defining** \( \text{lfp} \subseteq F : \, R_F \triangleq \{ \frac{P}{c} \mid P \subseteq U \land c \in F(P) \} \)
Why not using Aczel method to get the proof system at this point?

- We get a sound and complete proof system
Why not using Aczel method to get the proof system at this point?

- We get a sound and complete proof system
- **BUT** impractical:
  - you first *prove the strongest postcondition*, and then
  - use the *consequence rule to approximate*!
Fixpoint induction
Theorem H.3 (Non empty intersection with abstraction of least fixpoint). Assume that (1) \( \langle L, \sqsubseteq, \bot, \top, \sqcap, \sqcup \rangle \) is an atomic complete lattice; (2) \( f \in L \to L \) preserves nonempty joins \( \sqcup \); (3) \( \langle L, \sqsubseteq \rangle \xrightarrow{\nu} \langle \bar{L}, \leq, \wedge \rangle \); (4) \( \tilde{Q} \in \bar{L} \setminus \{0\} \) where \( 0 \models \alpha(\bot) \); (5) There exists an inductive invariant \( I \in L \) of \( f \) (i.e. \( f(I) \subseteq I \)); (6) \( \langle W, \leq \rangle \) is a well-founded set and \( \nu \in \text{atoms}(I) \to W \) is a (variant) function; (7) There exists a sequence \( \{a_i \in \text{atoms}(I), i \in [1, \infty]\} \) such that (7.a) \( a_1 \in f(\bot) \), (7.b) \( \forall i \in [1, \infty] \cdot a_{i+1} \in \text{atoms}(f(a_i)) \), (7.c) \( \forall i \in [1, \infty] \cdot (a_i \neq a_{i+1}) \Rightarrow (\nu(a_i) > \nu(a_{i+1})) \), (7.d) \( \forall i \in [1, \infty] \cdot (\nu(a_i) \not\sqsubseteq \nu(a_{i+1}) \Rightarrow \alpha(a_i) \wedge \tilde{Q} \neq 0) \); Then, hypotheses (1) to (7) imply \( \alpha(\text{lfp}^= f) \wedge \tilde{Q} \neq 0 \). Conversely (1) to (4) and \( \text{lfp}^= f \sqcap \nu(\tilde{Q}) \neq \bot \) imply (5) to (7).
Calculational design of the proof system
HL does not need a consequence rule

**Theorem 4.1 (Equivalent definitions of HL theories).**

\[
\begin{align*}
\mathcal{T}_{\text{HL}}(w) & \triangleq \text{post}(\leq, \exists) \circ \alpha^- \circ \mathcal{T}_{\text{HL}}(w) \\
& = \text{post}(\leq, \exists) \circ \alpha^- \circ \mathcal{T}_{\text{HL}}(w) \\
& = \text{post}(\leq, \exists)(\neg\{P, Q\} \mid \text{post}[w]P \subseteq Q) \\
& = \text{post}(\leq, \exists)(\{P, Q\} \mid \neg(\text{post}[w]P \subseteq Q)) \\
& = \text{post}(\leq, \exists)(\{P, Q\} \mid \text{post}[w]P \cap \neg Q \neq \emptyset) \\
& = \{(P', Q') \mid \exists(P, Q) \in \{(P, Q) \mid \text{post}[w]P \cap \neg Q \neq \emptyset\} . (P, Q) \leq, \exists (P', Q')\} \\
& = \{(P', Q') \mid \exists(P, Q) . \text{post}[w]P \cap \neg Q \neq \emptyset \wedge (P, Q) \leq, \exists (P', Q')\} \\
& = \{(P', Q') \mid \exists(P, Q) . \text{post}[w]P \cap \neg Q \neq \emptyset \wedge P \in P' \wedge Q \geq Q'\} \\
& = \{(P', Q') \mid \exists Q . \text{post}[w]P' \cap \neg Q \neq \emptyset \wedge Q \geq Q'\}
\end{align*}
\]

\(\wedge\) if \(P \in P'\) then \(\text{post}[w]P \subseteq \text{post}[w]P'\) by (12) so that \(\text{post}[w]P \cap \neg Q \neq \emptyset\) implies \(\text{post}[w]P' \cap \neg Q \neq \emptyset\); 
\(\vee\) conversely, if \(\exists Q . \text{post}[w]P'\), then \(\exists P . \text{post}[w]P \cap \neg Q \neq \emptyset \wedge P \in P'\) by choosing \(P = P'\). 

\(\neg\) if \(Q \supseteq Q'\) then \(\neg Q' \supseteq \neg Q\) so \(\text{post}[w]P' \cap \neg Q \neq \emptyset\) implies \(\text{post}[w]P' \cap \neg Q' \neq \emptyset\); 
\(\neg\) conversely post[\(w\)]\(P'\) \(\neg Q' \neq \emptyset\) implies \(\exists Q . \text{post}[w]P' \cap \neg Q \neq \emptyset \wedge Q \supseteq Q'\) by choosing \(Q = Q'\). 

\(\neg\) if \(\leq, \exists\) then \(\neg\) implies \(\text{post}[w]P' \cap \neg Q \neq \emptyset\) so that \(\text{post}[w]P' \cap \neg Q' \neq \emptyset\) implies \(\text{post}[w]P' \cap \neg Q' \neq \emptyset\). 

\(\alpha^- \circ \mathcal{T}_{\text{HL}}(w) \quad \text{(def. } \alpha^- \text{ and } \mathcal{T}_{\text{HL}} \text{ for Hoare logic)} \quad \Box\)
Theorem 4.2 (Theory of HL)

\[ T_{\text{HL}}(w) = \{ (P, Q) \mid \exists n \geq 1. \exists \{\sigma_i \in I, i \in [1, n]\} . \sigma_i \in P \land \forall i \in [1, n] . \{B[i], \sigma_i\} \in T_{\text{HL}}(S) \land \sigma_n \notin B[i] \land \sigma_n \notin Q \} \]

Proof of Th. 4.2.

\[ T_{\text{HL}}(w) \]

\[ \{ (P, Q) \mid \text{post}[-\emptyset](\text{hip}_g F_P) \land \neg Q \land \emptyset \} \quad \text{[Lem. 1.3, where } F_P(X) \triangleq P \cup \text{post}(\{B[i], [s]^i\}, X)] \]

\[ \{ (P, Q) \mid \text{hip}_g F_P(\emptyset) \land \neg Q \land \emptyset \} \quad \text{[(39), (49)]} \]

\[ \{ (P, Q) \mid \exists l \in p(S) . \emptyset \land \neg Q \land \emptyset \} \quad \text{[since } \forall \]...\]

\[ = \{(P, Q) \mid \exists n \geq 1. \exists \{\sigma_i \in I, i \in [1, n]\} . \sigma_i \in P \land \forall i \in [1, n] . \{B[i], \sigma_i\} \in T_{\text{HL}}(S) \land \sigma_n \notin B[i] \land \sigma_n \notin Q \} \]

\[ \text{is not used and can always be chosen to be } \Sigma \]

\[ \text{\& } \{ (P, Q) \mid \exists n \geq 1. \exists \{\sigma_i \in I, i \in [1, n]\} . \sigma_i \in P \land \forall i \in [1, n] . \{B[i], \sigma_i\} \in T_{\text{HL}}(S) \land \sigma_n \notin B[i] \land \sigma_n \notin Q \} \]

\[ \text{\& } \{ (P, Q) \mid \exists n \geq 1. \exists \{\sigma_i \in I, i \in [1, n]\} . \sigma_i \in P \land \forall i \in [1, n] . \{B[i], \sigma_i\} \in T_{\text{HL}}(S) \land \sigma_n \notin B[i] \land \sigma_n \notin Q \} \]

\[ \text{\& } \{ (P, Q) \mid \exists n \geq 1. \exists \{\sigma_i \in I, i \in [1, n]\} . \sigma_i \in P \land \forall i \in [1, n] . \{B[i], \sigma_i\} \in T_{\text{HL}}(S) \land \sigma_n \notin B[i] \land \sigma_n \notin Q \} \]

\[ \text{\& } \{ (P, Q) \mid \exists n \geq 1. \exists \{\sigma_i \in I, i \in [1, n]\} . \sigma_i \in P \land \forall i \in [1, n] . \{B[i], \sigma_i\} \in T_{\text{HL}}(S) \land \sigma_n \notin B[i] \land \sigma_n \notin Q \} \]

\[ \text{\& } \{ (P, Q) \mid \exists n \geq 1. \exists \{\sigma_i \in I, i \in [1, n]\} . \sigma_i \in P \land \forall i \in [1, n] . \{B[i], \sigma_i\} \in T_{\text{HL}}(S) \land \sigma_n \notin B[i] \land \sigma_n \notin Q \} \]

\[ \text{\& } \{ (P, Q) \mid \exists n \geq 1. \exists \{\sigma_i \in I, i \in [1, n]\} . \sigma_i \in P \land \forall i \in [1, n] . \{B[i], \sigma_i\} \in T_{\text{HL}}(S) \land \sigma_n \notin B[i] \land \sigma_n \notin Q \} \]

\[ \text{\& } \{ (P, Q) \mid \exists n \geq 1. \exists \{\sigma_i \in I, i \in [1, n]\} . \sigma_i \in P \land \forall i \in [1, n] . \{B[i], \sigma_i\} \in T_{\text{HL}}(S) \land \sigma_n \notin B[i] \land \sigma_n \notin Q \} \]

\[ \text{\& } \{ (P, Q) \mid \exists n \geq 1. \exists \{\sigma_i \in I, i \in [1, n]\} . \sigma_i \in P \land \forall i \in [1, n] . \{B[i], \sigma_i\} \in T_{\text{HL}}(S) \land \sigma_n \notin B[i] \land \sigma_n \notin Q \} \]

\[ \text{\& } \{ (P, Q) \mid \exists n \geq 1. \exists \{\sigma_i \in I, i \in [1, n]\} . \sigma_i \in P \land \forall i \in [1, n] . \{B[i], \sigma_i\} \in T_{\text{HL}}(S) \land \sigma_n \notin B[i] \land \sigma_n \notin Q \} \]

\[ \text{\& } \{ (P, Q) \mid \exists n \geq 1. \exists \{\sigma_i \in I, i \in [1, n]\} . \sigma_i \in P \land \forall i \in [1, n] . \{B[i], \sigma_i\} \in T_{\text{HL}}(S) \land \sigma_n \notin B[i] \land \sigma_n \notin Q \} \]

\[ \text{\& } \{ (P, Q) \mid \exists n \geq 1. \exists \{\sigma_i \in I, i \in [1, n]\} . \sigma_i \in P \land \forall i \in [1, n] . \{B[i], \sigma_i\} \in T_{\text{HL}}(S) \land \sigma_n \notin B[i] \land \sigma_n \notin Q \} \]

\[ \text{\& } \{ (P, Q) \mid \exists n \geq 1. \exists \{\sigma_i \in I, i \in [1, n]\} . \sigma_i \in P \land \forall i \in [1, n] . \{B[i], \sigma_i\} \in T_{\text{HL}}(S) \land \sigma_n \notin B[i] \land \sigma_n \notin Q \} \]
Proof system of \( \overline{\text{HL}} \)

**Theorem 4.3** (\( \overline{\text{HL}} \) rules for conditional iteration).

\[
\exists (\sigma_i \in I, i \in [1,n]) \cdot \sigma_1 \in P \land \forall i \in [1,n] \cdot (B[B] \cap \{\sigma_i\}) S (\neg\{\sigma_{i+1}\}) \land \sigma_n \notin B[B] \land \sigma_n \notin Q \\
(P) \text{while (B) } S (Q)
\]  

(3)

**Proof of (3).** We write \( (P) S (Q) \equiv (P, Q) \in \overline{\text{HL}}(S) \);

By structural induction (\( S \) being a strict component of while (B) S), the rule for \( (P) S (Q) \) have already been defined;

By **Aczel method**, the (constant) fixpoint \( \text{lfp}^e \lambda X \cdot S \) is defined by \( \{\emptyset | c \in S\} \);

So for while (B) S we have an axiom \( (P) \text{while (B) } S (Q) \) with side condition \( \exists (\sigma_i \in I, i \in [1,n]) \cdot \sigma_1 \in P \land \forall i \in [1,n] \cdot (B[B] \cap \{\sigma_i\}) S (\neg\{\sigma_{i+1}\}) \land \sigma_n \notin B[B] \land \sigma_n \notin Q \) where \( (B[B] \cap \{\sigma_i\}) S (\neg\{\sigma_{i+1}\}) \) is well-defined by structural induction;

Traditionally, the side condition is written as a premiss, to get (3).

\( \square \)
About incorrectness

- IL is **not** Hoare incorrectness logic (sufficient, not necessary)

\[
-(\{P\} \rightarrow \{Q\}) \iff [P]s[\neg Q] \\
\iff \exists R \in \wp(\Sigma). [P]s[R] \land R \cap \neg Q \neq \emptyset \\
\iff \exists \sigma \in \Sigma. [P]s[\{\sigma\}] \land \sigma \notin Q
\]
Conclusion

• Was Peter correct or incorrect?

• Of course he was correct

• BUT he took the hardest path

• Hoare incorrectness logic is the easiest and most popular way

• They are called debuggers

• It makes debugging a formal activity relying on a formal logic!
Conclusion

• Was Peter correct or incorrect?
• Of course he was correct
• **BUT** he took the hardest path
Conclusion

• Was Peter correct or incorrect?
• Of course he was correct
• **BUT** he took the hardest path
• **Hoare incorrectness logic** is the easiest and most popular way
Conclusion

• Was Peter correct or incorrect?
• Of course he was correct
• BUT he took the hardest path
• Hoare incorrectness logic is the easiest and most popular way
  • It has proof verifiers and theorem provers
Conclusion

• Was Peter correct or incorrect?
• Of course he was correct
• **BUT** he took the hardest path
• Hoare incorrectness logic is the easiest and most popular way
  • It has **proof verifiers and theorem provers**
  • They are called **debuggers**
Conclusion

• Was Peter correct or incorrect?
  • Of course he was correct
  • BUT he took the hardest path
  • Hoare incorrectness logic is the easiest and most popular way
    • It has proof verifiers and theorem provers
    • They are called debuggers
    • Which are therefore formal tools based on a formal logic!
Conclusion

• Was Peter correct or incorrect?
• Of course he was correct
• **BUT** he took the hardest path
• Hoare incorrectness logic is the easiest and most popular way
  • It has *proof verifiers* and *theorem provers*
  • They are called *debuggers*
  • Which are therefore *formal tools based on a formal logic!* 😅
The End, Thank You
The End, Thank You
Happy Sixties to Peter