Today
Today

Last lecture

• Finite model finding for first-order logic with quantifiers, relations, and transitive closure
Today

Last lecture

- Finite model finding for first-order logic with quantifiers, relations, and transitive closure

Today

- Reasoning about (partial) correctness of programs
  - Hoare Logic
  - Verification Condition Generation
Today

Last lecture

• Finite model finding for first-order logic with quantifiers, relations, and transitive closure

Today

• Reasoning about (partial) correctness of programs
  • Hoare Logic
  • Verification Condition Generation

Based on lectures by Isil Dillig, Daniel Jackson, and Viktor Kuncak
Program verification & checking (L10–L15)
Program verification & checking (L10–L15)

Classic verification (L10, L11)

• Checking that all (terminating) executions satisfy an FOL property on all inputs
Program verification & checking (L10–L15)

Classic verification (L10, L11)
- Checking that all (terminating) executions satisfy an FOL property on all inputs

Bounded verification (L12)
- Scope-complete checking of FOL properties
Program verification & checking (L10–L15)

Classic verification (L10, L11)
• Checking that all (terminating) executions satisfy an FOL property on all inputs

Bounded verification (L12)
• Scope-complete checking of FOL properties

Symbolic execution (L13)
• Systematic checking of FOL properties
Program verification & checking (L10–L15)

Classic verification (L10, L11)
• Checking that all (terminating) executions satisfy an FOL property on all inputs

Bounded verification (L12)
• Scope-complete checking of FOL properties

Symbolic execution (L13)
• Systematic checking of FOL properties

Model checking (L14, L15)
• Exhaustive checking of temporal properties of abstracted programs
Program verification & checking (L10–L15)

Classic verification (L10, L11)
  • Checking that all (terminating) executions satisfy an FOL property on all inputs

Bounded verification (L12)
  • Scope-complete checking of FOL properties

Symbolic execution (L13)
  • Systematic checking of FOL properties

Model checking (L14, L15)
  • Exhaustive checking of temporal properties of abstracted programs

Active research topic for 45 years
Program verification & checking (L10–L15)

Classic verification (L10, L11)
- Checking that all (terminating) executions satisfy an FOL property on all inputs

Bounded verification (L12)
- Scope-complete checking of FOL properties

Symbolic execution (L13)
- Systematic checking of FOL properties

Model checking (L14, L15)
- Exhaustive checking of temporal properties of abstracted programs
Program verification & checking (L10–L15)

Classic verification (L10, L11)
  • Checking that all (terminating) executions satisfy an FOL property on all inputs

Bounded verification (L12)
  • Scope-complete checking of FOL properties

Symbolic execution (L13)
  • Systematic checking of FOL properties

Model checking (L14, L15)
  • Exhaustive checking of temporal properties of abstracted programs

Active research topic for 45 years
Classic ideas every computer scientist should know
Understanding the ideas can help you become a better programmer
Classic verification: seminal papers
Classic verification: seminal papers

1967: Assigning Meaning to Programs (Floyd)
Classic verification: seminal papers

1967: Assigning Meaning to Programs (Floyd)

1969: An Axiomatic Basis for Computer Programming (Hoare)
Classic verification: seminal papers

1967: Assigning Meaning to Programs (Floyd)

1969: An Axiomatic Basis for Computer Programming (Hoare)

1975: Guarded Commands, Nondeterminacy and Formal Derivation of Programs (Dijkstra)
Classic verification: seminal papers

1967: Assigning Meaning to Programs (Floyd)
   • 1978 Turing Award

1969: An Axiomatic Basis for Computer Programming (Hoare)
   • 1980 Turing Award

1975: Guarded Commands, Nondeterminacy and Formal Derivation of Programs (Dijkstra)
   • 1972 Turing Award
Specifying correctness in Hoare logic

\{P\} S \{Q\}
Specifying correctness in Hoare logic

Hoare triple

- $S$ is a program statement (or fragment).
- $P$ is an FOL formula called the *precondition*.
- $Q$ is an FOL formula called the *postcondition*.
Specifying correctness in Hoare logic

Hoare triple

- S is a program statement (or fragment).
- P is an FOL formula called the *precondition*.
- Q is an FOL formula called the *postcondition*.

Partial correctness (Hoare triple semantics)

- If S executes from a state satisfying P, and if its execution terminates, then the resulting state satisfies Q.
Specifying correctness in Hoare logic

**Hoare triple**

- \( S \) is a program statement (or fragment).
- \( P \) is an FOL formula called the *precondition*.
- \( Q \) is an FOL formula called the *postcondition*.

**Partial correctness (Hoare triple semantics)**

- If \( S \) executes from a state satisfying \( P \), and if its execution terminates, then the resulting state satisfies \( Q \).

**Total correctness**

- If \( S \) executes from a state satisfying \( P \), then its execution terminates and the resulting state satisfies \( Q \).
Specifying correctness in Hoare logic

Hoare triple
- S is a program statement (or fragment).
- P is an FOL formula called the *precondition*.
- Q is an FOL formula called the *postcondition*.

Partial correctness (Hoare triple semantics)
- If S executes from a state satisfying P, and if its execution terminates, then the resulting state satisfies Q.

Total correctness
- If S executes from a state satisfying P, then its execution terminates and the resulting state satisfies Q.
Specifying correctness in Hoare logic

Hoare triple

- S is a program statement (or fragment).
- P is an FOL formula called the \textit{precondition}.
- Q is an FOL formula called the \textit{postcondition}.

Partial correctness (Hoare triple semantics)

- If S executes from a state satisfying P, and if its execution terminates, then the resulting state satisfies Q.

Total correctness

- If S executes from a state satisfying P, then its execution terminates and the resulting state satisfies Q.
Specifying correctness in Hoare logic

**Hoare triple**

- S is a program statement (or fragment).
- P is an FOL formula called the *precondition*.
- Q is an FOL formula called the *postcondition*.

**Partial correctness (Hoare triple semantics)**

- If S executes from a state satisfying P, and if its execution terminates, then the resulting state satisfies Q.

**Total correctness**

- If S executes from a state satisfying P, then its execution terminates and the resulting state satisfies Q.
Examples of Hoare triples
Examples of Hoare triples

{false} S {Q}
Examples of Hoare triples

\{false\} S \{Q\}

- Valid for all S and Q.
Examples of Hoare triples

\{false\} \text{S} \{Q\}
- Valid for all S and Q.

\{P\} \text{while (true) do skip} \{Q\}
Examples of Hoare triples

\{false\} S \{Q\}
- Valid for all S and Q.

\{P\} while (true) do skip \{Q\}
- Valid for all P and Q.
Examples of Hoare triples

\{false\} S \{Q\}
  • Valid for all S and Q.

\{P\} while (true) do skip \{Q\}
  • Valid for all P and Q.

\{true\} S \{Q\}
Examples of Hoare triples

{false} S {Q}
• Valid for all S and Q.

{P} while (true) do skip {Q}
• Valid for all P and Q.

{true} S {Q}
• If S terminates, then Q must hold.
Examples of Hoare triples

{false} S {Q}
  • Valid for all S and Q.

{P} while (true) do skip {Q}
  • Valid for all P and Q.

{true} S {Q}
  • If S terminates, then Q must hold.

{P} S {true}
Examples of Hoare triples

{false} S {Q}
  • Valid for all S and Q.

{P} while (true) do skip {Q}
  • Valid for all P and Q.

{true} S {Q}
  • If S terminates, then Q must hold.

{P} S {true}
  • Valid for all P and S.
A simple imperative language

- **Expression E**
  - Z | V | E₁ + E₂ | E₁ * E₂

- **Conditional C**
  - true | false | E₁ = E₂ | E₁ ≤ E₂

- **Statement S**
  - skip                  (Skip)
  - V := E                (Assignment)
  - S₁; S₂               (Composition)
  - if C then S₁ else S₂ (If)
  - while C do S         (While)
Proving partial correctness in Hoare logic

A simple imperative language

- **Expression E**
  - Z | V | E₁ + E₂ | E₁ * E₂

- **Conditional C**
  - true | false | E₁ = E₂ | E₁ ≤ E₂

- **Statement S**
  - skip (Skip)
  - V := E (Assignment)
  - S₁; S₂ (Composition)
  - if C then S₁ else S₂ (If)
  - while C do S (While)

One inference rule for every statement in the language:

\[ \vdash \{P\}_1 S_1 \{Q_1\} \ldots \vdash \{P\}_n S_n \{Q_n\} \]
\[ \vdash \{P\} S \{Q\} \]

If the Hoare triples \{P₁\} \ S₁\{Q₁\} \ldots \{Pₙ\}Sₙ\{Qₙ\} are provable, then so is \{P\}S\{Q\}.
Inference rules for Hoare logic

\[ \vdash \{P\} \text{skip} \{P\} \]
Inference rules for Hoare logic

\[\vdash \{P\} \text{skip} \{P\}\]

\[\vdash \{Q[E/x]\} x := E \{Q\}\]
Inference rules for Hoare logic

\[
\begin{align*}
\Gamma \vdash \{P\} \text{skip} \{P\} \\
\Gamma \vdash \{Q[E/x]\} x := E \{Q\} \\
\Gamma \vdash \{P_1\} S \{Q_1\} &\quad P \Rightarrow P_1 \quad Q_1 \Rightarrow Q \\
\Gamma \vdash \{P\} S \{Q\}
\end{align*}
\]
Inference rules for Hoare logic

\[ \frac{\vdash \{P\} \text{skip} \{P\} }{\vdash \{Q[E/x]\} x := E \{Q\} } \]

\[ \frac{\vdash \{P\} S_i \{R\} \quad \vdash \{R\} S_i \{Q\} }{\vdash \{P\} S_i; S_i \{Q\} } \]

\[ \vdash \{P\} S \{Q\} \]

\[ \frac{\vdash \{P\} S \{Q\} \quad P \Rightarrow P_i \quad Q_i \Rightarrow Q }{ \vdash \{P\} S \{Q\} } \]
Inference rules for Hoare logic

\[ \vdash \{P\} \text{skip} \{P\} \]

\[ \vdash \{Q[E/x]\} \ x := E \ \{Q\} \]

\[ \vdash \{P\} \ S_1 \ \{R\} \quad \vdash \{R\} \ S_2 \ \{Q\} \]

\[ \vdash \{P\} \ S_1 ; \ S_2 \ \{Q\} \]

\[ \vdash \{P \land C\} \ S_1 \ \{Q\} \quad \vdash \{P \land \neg C\} \ S_2 \ \{Q\} \]

\[ \vdash \{P\} \ \text{if} \ C \ \text{then} \ S_1 \ \text{else} \ S_2 \ \{Q\} \]

\[ \vdash \{P_1\} \ S \ \{Q_1\} \quad P \Rightarrow P_1 \quad Q_1 \Rightarrow Q \]

\[ \vdash \{P\} \ S \ \{Q\} \]
Inference rules for Hoare logic

\[ \vdash \{P\} \text{skip} \{P\} \]

\[ \vdash \{Q[E/x]\} x := E \{Q\} \]

\[ \vdash \{P_1\} S \{Q_1\} \quad P \Rightarrow P_1 \quad Q_1 \Rightarrow Q \]

\[ \vdash \{P\} S \{Q\} \]

\[ \vdash \{P\} S_1 \{R\} \quad \vdash \{R\} S_2 \{Q\} \]

\[ \vdash \{P\} S_1; S_2 \{Q\} \]

\[ \vdash \{P \land C\} S_1 \{Q\} \quad \vdash \{P \land \neg C\} S_2 \{Q\} \]

\[ \vdash \{P\} \text{if} C \text{ then } S_1 \text{ else } S_2 \{Q\} \]

\[ \vdash \{P \land C\} S \{P\} \]

\[ \vdash \{P\} \text{while} C \text{ do } S \{P \land \neg C\} \]

	extit{loop invariant}
Example: proof outline

\{x \leq n\}
while (x < n) do
  \{x \leq n \land x < n\}
  \{x+1 \leq n\}       \quad \text{\textit{// consequence}}
x := x + 1
  \{x \leq n\}       \quad \text{\textit{// assignment}}
\{x \leq n \land x \geq n\} \quad \text{\textit{// while}}
\{x \geq n\}       \quad \text{\textit{// consequence}}
Example: proof outline with auxiliary variables

\{x = X \land y = Y\}  
\{y = Y \land x = X\}  
t := x  
\{y = Y \land t = X\}  // assignment  
x := y  
\{x = Y \land t = X\}  // assignment  
y := t  
\{x = Y \land y = X\}  // assignment
Soundness and relative completeness
Soundness and relative completeness

Proof rules for Hoare logic are sound

If $\vdash \{P\} S \{Q\}$ then $\models \{P\} S \{Q\}$
Soundness and relative completeness

Proof rules for Hoare logic are sound

If $\vdash \{P\} S \{Q\}$ then $\models \{P\} S \{Q\}$

Proof rules for Hoare logic are relatively complete

If $\models \{P\} S \{Q\}$ then $\vdash \{P\} S \{Q\}$, assuming an oracle for deciding implications
Automating Hoare logic with VC generation

Program annotated with pre/post conditions, loop invariants

Verification Condition Generator (VCG)

verification condition (VC)

SMT solver
Automating Hoare logic with VC generation

Program annotated with pre/post conditions, loop invariants

Verification Condition Generator (VCG)

verification condition (VC)

A formula $\varphi$ generated automatically from the annotated program.
The program satisfies the specification if $\varphi$ is valid.

SMT solver
Automating Hoare logic with VC generation

Program annotated with pre/post conditions, loop invariants

Verification Condition Generator (VCG)

Forwards computation:
• Starting from the precondition, generate formulas to prove the postcondition.
• Based on computing strongest postconditions \( (sp) \).

Backwards computation:
• Starting from the postcondition, generate formulas to prove the precondition.
• Based on computing weakest liberal preconditions \( (wp) \).
VC generation with WP and SP
VC generation with WP and SP

wp(S, Q)

• The weakest predicate that guarantees Q will hold after executing S from a state satisfying that predicate.
VC generation with WP and SP

\(wp(S, Q)\)

- The weakest predicate that guarantees \(Q\) will hold after executing \(S\) from a state satisfying that predicate.

\(sp(S, P)\)

- The strongest predicate that holds after \(S\) is executed from a state satisfying \(P\).
VC generation with WP and SP

wp(S, Q)
• The weakest predicate that guarantees Q will hold after executing S from a state satisfying that predicate.

sp(S, P)
• The strongest predicate that holds after S is executed from a state satisfying P.

{P} S {Q} is valid iff
• P \Rightarrow wp(S, Q)
• sp(S, P) \Rightarrow Q
Computing $wp(S, Q)$
Computing $\text{wp}(S, Q)$

$\text{wp}(S, Q)$:
Computing \( \text{wp}(S, Q) \)

\( \text{wp}(S, Q): \)

- \( \text{wp}(\text{skip}, Q) = Q \)
Computing \( \text{wp}(S, Q) \)

\( \text{wp}(S, Q): \)

\begin{itemize}
  \item \( \text{wp}(\text{skip}, Q) = Q \)
  \item \( \text{wp}(x := E, Q) = Q[E / x] \)
\end{itemize}
Computing $wp(S, Q)$

$wp(S, Q)$:

- $wp(\text{skip}, Q) = Q$
- $wp(x := E, Q) = Q[E / x]$
- $wp(S_1; S_2, Q) = wp(S_1, wp(S_2, Q))$
Computing $wp(S, Q)$

$wp(S, Q)$:

- $wp($skip, $Q) = Q$
- $wp($x := $E, Q) = Q[E / x]$
- $wp(S_1; S_2, Q) = wp(S_1, wp(S_2, Q))$
- $wp($if $C$ then $S_1$ else $S_2$, $Q) = C \rightarrow wp(S_1, Q) \land \neg C \rightarrow wp(S_2, Q)$
Computing \( wp(S, Q) \)

\( wp(S, Q): \)

\begin{itemize}
  \item \( wp(\text{skip}, Q) = Q \)
  \item \( wp(x := E, Q) = Q[E / x] \)
  \item \( wp(S_1; S_2, Q) = wp(S_1, wp(S_2, Q)) \)
  \item \( wp(\text{if } C \text{ then } S_1 \text{ else } S_2, Q) = C \rightarrow wp(S_1, Q) \land \neg C \rightarrow wp(S_2, Q) \)
  \item \( wp(\text{while } C \text{ do } S, Q) = ? \)
\end{itemize}
Computing \( \text{wp}(S, Q) \)

\( \text{wp}(S, Q) : \)

- \( \text{wp}(\text{skip}, Q) = Q \)
- \( \text{wp}(x := E, Q) = Q[E / x] \)
- \( \text{wp}(S_1; S_2, Q) = \text{wp}(S_1, \text{wp}(S_2, Q)) \)
- \( \text{wp}(\text{if } C \text{ then } S_1 \text{ else } S_2, Q) = C \rightarrow \text{wp}(S_1, Q) \land \neg C \rightarrow \text{wp}(S_2, Q) \)
- \( \text{wp}(\text{while } C \text{ do } S, Q) = \times \)

A fixpoint: cannot be expressed as a syntactic construction in terms of the postcondition.
Computing \( wp(S, Q) \)

\( wp(S, Q): \)

- \( wp(\text{skip}, Q) = Q \)
- \( wp(x := E, Q) = Q[E / x] \)
- \( wp(S_1; S_2, Q) = wp(S_1, wp(S_2, Q)) \)
- \( wp(\text{if } C \text{ then } S_1 \text{ else } S_2, Q) = C \rightarrow wp(S_1, Q) \land \neg C \rightarrow wp(S_2, Q) \)
- \( wp(\text{while } C \text{ do } S, Q) = \) ✗

Approximate \( wp(S, Q) \) with \( awp(S, Q) \).
Computing awp(S, Q)

awp(S, Q):

• awp(skip, Q) = Q
• awp(x := E, Q) = Q[E / x]
• awp(S_1; S_2, Q) = awp(S_1, awp(S_2, Q))
• awp(if C then S_1 else S_2, Q) = C → awp(S_1, Q) ∧ ¬C → awp(S_2, Q)
• awp(while C do {I} S, Q) = I
Computing \texttt{awp}(S, Q)

\texttt{awp}(S, Q):

- \texttt{awp}(**skip**, Q) = Q
- \texttt{awp}(x := E, Q) = Q[E / x]
- \texttt{awp}(S_1; S_2, Q) = \texttt{awp}(S_1, \texttt{awp}(S_2, Q))
- \texttt{awp}(\texttt{if} \ C \ \texttt{then} \ S_1 \ \texttt{else} \ S_2, Q) = C \rightarrow \texttt{awp}(S_1, Q) \land \neg C \rightarrow \texttt{awp}(S_2, Q)
- \texttt{awp}(\texttt{while} \ C \ \texttt{do} \ \{I\} \ S, Q) = I

Loop invariant provided by an oracle (e.g., programmer).
Computing \(awp(S, Q)\)

\(awp(S, Q):\)

- \(awp(\text{skip}, Q) = Q\)
- \(awp(x := E, Q) = Q[E/x]\)
- \(awp(S_1; S_2, Q) = awp(S_1, awp(S_2, Q))\)
- \(awp(\text{if } C \text{ then } S_1 \text{ else } S_2, Q) = C \rightarrow awp(S_1, Q) \land \neg C \rightarrow awp(S_2, Q)\)
- \(awp(\text{while } C \text{ do } \{I\} S, Q) = I\)

For each statement \(S\), also define \(VC(S, Q)\) that encodes additional conditions that must be checked.
Computing $\text{VC}(S, Q)$
Computing $\text{VC}(S, Q)$

$\text{VC}(S, Q)$:
Computing $VC(S, Q)$

$VC(S, Q)$:

• $VC(skip, Q) = true$
Computing $\text{VC}(S, Q)$

$\text{VC}(S, Q)$:

- $\text{VC}(\text{skip}, Q) = \text{true}$
- $\text{VC}(x := E, Q) = \text{true}$
Computing $\text{VC}(S, Q)$

$\text{VC}(S, Q)$:

- $\text{VC}(\text{skip}, Q) = \text{true}$
- $\text{VC}(x := E, Q) = \text{true}$
- $\text{VC}(S_1; S_2, Q) = \text{VC}(S_2, Q) \land \text{VC}(S_1, \text{awp}(S_2, Q))$
Computing \( VC(S, Q) \)

\( VC(S, Q) : \)

- \( VC(\text{skip}, Q) = \text{true} \)
- \( VC(x := E, Q) = \text{true} \)
- \( VC(S_1; S_2, Q) = VC(S_2, Q) \land VC(S_1, \text{awp}(S_2, Q)) \)
- \( VC(\text{if } C \text{ then } S_1 \text{ else } S_2, Q) = VC(S_1, Q) \land VC(S_2, Q) \)
Computing $\text{VC}(S, Q)$

$\text{VC}(S, Q)$:

- $\text{VC}(\text{skip}, Q) = \text{true}$
- $\text{VC}(x := E, Q) = \text{true}$
- $\text{VC}(S_1; S_2, Q) = \text{VC}(S_2, Q) \land \text{VC}(S_1, \text{awp}(S_2, Q))$
- $\text{VC}(\text{if } C \text{ then } S_1 \text{ else } S_2, Q) = \text{VC}(S_1, Q) \land \text{VC}(S_2, Q)$
- $\text{VC}(\text{while } C \text{ do } \{I\} S, Q) = (I \land C \Rightarrow \text{awp}(S, I) \land \text{VC}(S, I)) \land (I \land \neg C \Rightarrow Q)$

$I$ is an invariant. $I$ is strong enough.
Verifying a Hoare triple

Theorem: \( \{P\} \ S \ \{Q\} \) is valid if

\[ VC(S, Q) \land P \rightarrow awp(S, Q) \]
Theorem: \( \{P\} S \{Q\} \) is valid if

\[ VC(S, Q) \land P \rightarrow awp(S, Q) \]

The other direction doesn’t hold because loop invariants may not be strong enough or they may be incorrect.

Might get false alarms.
Summary

Today

- Reasoning about partial correctness of programs
  - Hoare Logic
  - VCG, WP, SP

Next lecture

- Guest lecture by Rustan Leino!
- Verification with Dafny, Boogie, and Z3.