CS720

Logical Foundations of Computer Science

Lecture 16: Program verification (part 3)

Tiago Cogumbreiro

`Equiv.v`

Due Thursday October 25, 11:59pm EST

`Imp.v`

Due Friday October 26, 11:59pm EST

`Hoare.v`, `HoareAsLogic.v`, `Hoare2.v`

Due Thursday November 1, 11:59pm EST

Summary

- Axiomatic Hoare Logic
- Program verification using Hoare logic

On the strength of propositions

Recall the rule for consequence

$$\frac{P \twoheadrightarrow P' \qquad \{P'\}\, c\, \{Q'\} \qquad Q' \twoheadrightarrow Q}{\{P\}\, c\, \{Q\}}$$

> We mentioned that we can **strengthen** the pre-condition and **weaken** the post-condition.

1. *Strengthening a pre-condition* ($P \twoheadrightarrow P'$) means having **more assumptions** to reach the same goal.
2. *Weakening a post-condition* ($Q' \twoheadrightarrow Q$) means having **fewer goals** to prove.

Stronger and weaker statements

- When you think of the strength of propositions, think of $\implies$ as $\geq$.
- We say that P is (strictly) **stronger** than Q if $P \implies Q$ (and $\neg(Q \implies P)$)
  That is, (strict-)strength corresponds to (strict-)implication.

> Between $x = 3 \wedge y = 10$ and $x = 3$, which is stronger than the other?

Stronger and weaker statements

- When you think of the strength of propositions, think of $\implies$ as $\geq$.
- We say that P is (strictly) **stronger** than Q if $P \implies Q$ (and $\neg(Q \implies P)$)
  That is, (strict-)strength corresponds to (strict-)implication.

> Between $x = 3 \wedge y = 10$ and $x = 3$, which is stronger than the other?
>
> - $x = 3 \wedge y = 10$ is stronger than $x = 3$, which is weaker

Weakest pre-condition

> E.W. Dijkstra, A Discipline of Programming, Prentice-Hall, 1976.

The *weakest-pre-condition* of a program c and a post-condition $\{Q\}$, is such that we can always prove Q.

```
Definition wp (c:com) (Q:Assertion) : Assertion :=
  fun s => forall s', c / s \\ s' -> Q s'.
```

1. **Theorem** (WP is the pre-condition of any program): $\{\mathtt{wp}(c, Q)\}\ c\ \{Q\}$
2. **Theorem** (WP is the weakest pre-condition): If $\{P\}\ c\ \{Q\}$, then $\{P\} \twoheadrightarrow \{\mathtt{wp}(c, Q)\}$.

Axiomatic Hoare Logic

`HoareAsLogic.v`

Hoare Logic Theory

$$\{P\}\ \texttt{SKIP}\ \{P\}\ \text{(H-skip)} \qquad \{P[x \mapsto a]\}\ x ::= a\ \{P\}\ \text{(H-asgn)}$$

$$\frac{\{P\}\ c_1\ \{Q\} \qquad \{Q\}\ c_2\ \{R\}}{\{P\}\ c_1;;c_2\ \{R\}}\ \text{(H-seq)}$$

$$\frac{P \twoheadrightarrow P' \qquad \{P'\}\ c\ \{Q'\} \qquad Q' \twoheadrightarrow Q}{\{P\}\ c\ \{Q\}}\ \text{(H-cons)}$$

$$\frac{\{P \wedge b\}\ c_1\ \{Q\} \qquad \{P \wedge \neg b\}\ c_2\ \{Q\}}{\{P\}\ \texttt{IFB}\ b\ \texttt{THEN}\ c_1\ \texttt{ELSE}\ c_2\ \texttt{FI}\ \{Q\}}\ \text{(H-if)}$$

$$\frac{\{P \wedge b\}\ c\ \{P\}}{\{P\}\ \texttt{WHILE}\ b\ \texttt{DO}\ c\ \texttt{END}\ \{P \wedge \neg b\}}\ \text{(H-while)}$$

Hoare Logic as an Axiomatic Logic

The theorems in Slide 6 are necessary and sufficient to show that a Hoare's triple holds.
We can use the theorems as axioms (rules) and encode Hoare's Logic **axiomatically**!

- **Necessary** condition (soundness): $\texttt{hoare\_proof}(P, c, Q) \to \{P\}\ c\ \{Q\}$
- **Sufficient** condition (completeness): $\{P\}\ c\ \{Q\} \to \texttt{hoare\_proof}(P, c, Q)$

```
Inductive hoare_proof : Assertion → com → Assertion → Type :=
  | H_Skip : forall P, hoare_proof P (SKIP) P
  | H_Asgn : forall Q V a, hoare_proof (assn_sub V a Q) (V ::= a) Q
  | H_Seq  : forall P c Q d R, hoare_proof P c Q → hoare_proof Q d R → hoare_proof P (c;;d) R
  | H_If : forall P Q b c1 c2,
    hoare_proof (fun st ⇒ P st /\ bassn b st) c1 Q →
    hoare_proof (fun st ⇒ P st /\ ~(bassn b st)) c2 Q →
    hoare_proof P (IFB b THEN c1 ELSE c2 FI) Q
  | H_While : forall P b c,
    hoare_proof (fun st ⇒ P st /\ bassn b st) c P →
    hoare_proof P (WHILE b DO c END) (fun st ⇒ P st /\ ~ (bassn b st))
  | H_Consequence  : forall (P Q P' Q' : Assertion) c,
    hoare_proof P' c Q' → (forall st, P st → P' st) → (forall st, Q' st → Q st) → hoare_proof P c Q.
```

Why an Axiomatic Hoare Logic?

- When defining a logic axiomatically, you get the principles of injectivity (`inversion`) and induction for free.
- For instance, given some evidence, we can reason about how we reached that conclusion (ie, using the rules/constructors).
- In this specific case, it forces us to think more deeply about Hoare's logic (eg, learn about the weakest pre-conditon).

> Your homework is to prove soundness and completeness!
>
> - Soundness (easy): $\texttt{hoare\_proof}(P, c, Q) \to \{P\}\, c\, \{Q\}$
> - Completeness (hard): $\{P\}\, c\, \{Q\} \to \texttt{hoare\_proof}(P, c, Q)$

Exercise

```
Theorem hoare_proof_complete: forall P c Q,
  {{P}} c {{Q}} → hoare_proof P c Q.
Proof.
```

The proof follows by induction on the structure of `c`.

- At each case our goal is to apply the rule that relates to the term. (when `c = SKIP` we apply `H-skip`, when `c = s ::= a` we apply `H-asgn`, and so on).
- When applying a rule and it requires a condition `?P` we don't know how to fill, supply the weakest precondition of the post-condition.

Verifying programs

`Hoare2.v`

Example

What does this algorithm do and how do we specify this algorithm?

```
X ::= X + Y;;
Y ::= X - Y;;
X ::= X - Y
```

Example

Pre and post condition

```
{{ X = m /\ Y = n }}
X ::= X + Y;;
Y ::= X - Y;;
X ::= X - Y
{{ X = n /\ Y = m }}
```

Fully annotated example

Let us learn how to annotate a program

```
  {{ X = m ∧ Y = n }} ->>
  {{ (X + Y) - ((X + Y) - Y) = n ∧ (X + Y) - Y = m }}
    X ::= X + Y;;
  {{ X - (X - Y) = n ∧ X - Y = m }}
    Y ::= X - Y;;
  {{ X - Y = n ∧ Y = m }}
   X ::= X - Y
  {{ X = n ∧ Y = m }}
```

Annotating assignments/sequences

> Start from the post-condition and work backwards. The pre-condition of the program must imply the pre-condition of the first instruction.

```
1. {{ X = m ∧ Y = n }} ->>
2. {{                             }}
     X ::= X + Y;;
3. {{                             }}
     Y ::= X - Y;;
4. {{                            }}
    X ::= X - Y
5. {{ X = n ∧ Y = m }}
```

Annotating assignments/sequences

> In an assignment you have `{{ P [ X |-> A ] }} X ::= a {{ P }}`, so take the post-condition (5) and replace `X` by `a`.

```
1. {{ X = m ∧ Y = n }} ->>
2. {{                             }}
     X ::= X + Y;;
3. {{                             }}
     Y ::= X - Y;;
4. {{                            }}
    X ::= X - Y
5. {{ X = n ∧ Y = m }}
```

Annotating assignments/sequences

> In an assignment you have `{{ P [ X |-> A ] }} X ::= a {{ P }}`, so take the post-condition (5) and replace `X` by `a`.

```
1. {{ X = m ∧ Y = n }} ->>
2. {{                             }}
     X ::= X + Y;;
3. {{                             }}
     Y ::= X - Y;;
4. {{ X - Y = n ∧ Y = m }}
    X ::= X - Y
5. {{ X = n ∧ Y = m }}
```

Annotating assignments/sequences

> In an assignment you have `{{ P [ X |-> A ] }} X ::= a {{ P }}`, so take the post-condition (5) and replace `X` by `a`.

```
1. {{ X = m ∧ Y = n }} ->>
2. {{                             }}
     X ::= X + Y;;
3. {{ X - (X - Y) = n ∧ X - Y = m }}
     Y ::= X - Y;;
4. {{ X - Y = n ∧ Y = m }}
    X ::= X - Y
5. {{ X = n ∧ Y = m }}
```

Annotating assignments/sequences

> In an assignment you have `{{ P [ X |-> A ] }} X ::= a {{ P }}`, so take the post-condition (5) and replace `X` by `a`.

```
1. {{ X = m ∧ Y = n }} ->>
2. {{ (X + Y) - ((X + Y) - Y) = n ∧ (X + Y) - Y = m }}
     X ::= X + Y;;
3. {{ X - (X - Y) = n ∧ X - Y = m }}
     Y ::= X - Y;;
4. {{ X - Y = n ∧ Y = m }}
    X ::= X - Y
5. {{ X = n ∧ Y = m }}
```

> Finally, prove all the consequence-rules, that is show that `(1) ->> (2)`.

Annotating conditionals

Conditionals follow this structure

```
{{ P }}
IFB b THEN
    {{ P ∧ b }}
    c1
    {{ Q }}
ELSE
    {{ P ∧ ¬b }}
    c2
    {{ Q }}
FI
{{ Q }}
```

```
{{True}}
     IFB X ≤ Y THEN
{{              }} ->>
{{              }}
                   Z ::= Y - X
{{              }}
                 ELSE
{{              }} ->>
{{              }}
    Z ::= X - Y
{{              }}
                FI
{{Z + X = Y ∨ Z + Y = X}}
```

Annotating conditionals

Conditionals follow this structure

```
{{ P }}
IFB b THEN
    {{ P ∧ b }}
    c1
    {{ Q }}
ELSE
    {{ P ∧ ¬b }}
    c2
    {{ Q }}
FI
{{ Q }}
```

```
{{True}}
     IFB X ≤ Y THEN
{{                 }} ->>
{{                 }}
                 Z ::= Y - X
{{Z + X = Y ∨ Z + Y = X}}
               ELSE
{{                 }} ->>
{{                 }}
    Z ::= X - Y
{{Z + X = Y ∨ Z + Y = X}}
               FI
 {{Z + X = Y ∨ Z + Y = X}}
```

Annotating conditionals

Conditionals follow this structure

```
{{ P }}
IFB b THEN
    {{ P ∧ b }}
    c1
    {{ Q }}
ELSE
    {{ P ∧ ¬b }}
    c2
    {{ Q }}
FI
{{ Q }}
```

```
{{True}}
     IFB X ≤ Y THEN
{{ True /\ X ≤ Y }} ->>
{{                }}
              Z ::= Y - X
{{Z + X = Y V Z + Y = X}}
            ELSE
{{ True /\ ~ (X ≤ Y) }} ->>
{{                }}
    Z ::= X - Y
{{Z + X = Y V Z + Y = X}}
            FI
 {{Z + X = Y V Z + Y = X}}
```

Annotating conditionals

Conditionals follow this structure

```
{{ P }}
IFB b THEN
    {{ P ∧ b }}
    c1
    {{ Q }}
ELSE
    {{ P ∧ ¬b }}
    c2
    {{ Q }}
FI
{{ Q }}
```

```
{{True}}
     IFB X ≤ Y THEN
{{ True /\ X ≤ Y }} ->>
{{ (Y - X) + X = Y \/ (Y - X) + Y = X }}
                Z ::= Y - X
{{Z + X = Y V Z + Y = X}}
              ELSE
{{ True /\ ~ (X ≤ Y) }} ->>
{{ (X - Y) + X = Y V (X - Y) + Y = X }}
    Z ::= X - Y
{{Z + X = Y V Z + Y = X}}
             FI
 {{Z + X = Y V Z + Y = X}}
```

Annotating loops

```
{{ P }}
WHILE b DO
    {{ P ∧ b }}
    c
    {{ P }}
END
{{ P ∧ ¬b }}
```

```
{{ True }} ->>
{{                }}
    X ::= m;;
{{                }}
    Y ::= 0;;
{{                }}
    WHILE n ≤ X DO
{{                }} ->>
{{                }}
    X ::= X - n;;
{{                }}
    Y ::= Y + 1
{{                }}
    END
{{ n * Y + X = m ∧ X < n }}
```

Annotating loops

```
{{ P }}
WHILE b DO
    {{ P ∧ b }}
    c
    {{ P }}
END
{{ P ∧ ¬b }}
```

```
{{ True }} ->>
{{             }}
    X ::= m;;
{{             }}
    Y ::= 0;;
{{ n * Y + X = m }}
    WHILE n ≤ X DO
{{             }} ->>
{{             }}
    X ::= X - n;;
{{             }}
    Y ::= Y + 1
{{ n * Y + X = m }}
    END
{{ n * Y + X = m ∧ X < n }}
```

Annotating loops

```
{{ P }}
WHILE b DO
    {{ P ∧ b }}
    c
    {{ P }}
END
{{ P ∧ ¬b }}
```

```
{{ True }} ->>
{{             }}
    X ::= m;;
{{             }}
    Y ::= 0;;
{{ n * Y + X = m }}
    WHILE n ≤ X DO
{{ n * Y + X = m /\ n ≤ X }} ->>
{{             }}
    X ::= X - n;;
{{             }}
    Y ::= Y + 1
{{ n * Y + X = m }}
    END
{{ n * Y + X = m ∧ X < n }}
```

Annotating loops

```
{{ P }}
WHILE b DO
    {{ P ∧ b }}
    c
    {{ P }}
END
{{ P ∧ ¬b }}
```

```
{{ True }} ->>
{{             }}
    X ::= m;;
{{             }}
    Y ::= 0;;
{{ n * Y + X = m }}
    WHILE n ≤ X DO
{{ n * Y + X = m /\ n ≤ X }} ->>
{{             }}
    X ::= X - n;;
{{ n * (Y + 1) + X = m }}
    Y ::= Y + 1
{{ n * Y + X = m }}
    END
{{ n * Y + X = m ∧ X < n }}
```

Annotating loops

```
{{ P }}
WHILE b DO
    {{ P ∧ b }}
    c
    {{ P }}
END
{{ P ∧ ¬b }}
```

```
{{ True }} ->>
{{              }}
    X ::= m;;
{{              }}
    Y ::= 0;;
{{ n * Y + X = m }}
    WHILE n ≤ X DO
{{ n * Y + X = m /\ n ≤ X }} ->>
{{ n * (Y + 1) + (X - n) = m }}
    X ::= X - n;;
{{ n * (Y + 1) + X = m }}
    Y ::= Y + 1
{{ n * Y + X = m }}
    END
{{ n * Y + X = m ∧ X < n }}
```

Annotating loops

```
{{ P }}
WHILE b DO
    {{ P ∧ b }}
    c
    {{ P }}
END
{{ P ∧ ¬b }}
```

```
{{ True }} ->>
{{                 }}
    X ::= m;;
{{ n * 0 + X = m }}
    Y ::= 0;;
{{ n * Y + X = m }}
    WHILE n ≤ X DO
{{ n * Y + X = m /\ n ≤ X }} ->>
{{ n * (Y + 1) + (X - n) = m }}
    X ::= X - n;;
{{ n * (Y + 1) + X = m }}
    Y ::= Y + 1
{{ n * Y + X = m }}
    END
{{ n * Y + X = m ∧ X < n }}
```

Annotating loops

```
{{ P }}
WHILE b DO
    {{ P ∧ b }}
    c
    {{ P }}
END
{{ P ∧ ¬b }}
```

```
{{ True }} ->>
{{ n * 0 + m = m }}
    X ::= m;;
{{ n * 0 + X = m }}
    Y ::= 0;;
{{ n * Y + X = m }}
    WHILE n ≤ X DO
{{ n * Y + X = m /\ n ≤ X }} ->>
{{ n * (Y + 1) + (X - n) = m }}
    X ::= X - n;;
{{ n * (Y + 1) + X = m }}
    Y ::= Y + 1
{{ n * Y + X = m }}
    END
{{ n * Y + X = m ∧ X < n }}
```

Loop invariants

Finding the loop invariant P is undecidable!

> It depends on what the body of `c` is and its surrounding conditions:
>
> 1. weak enough to be implied by the loop's precondition
> 2. strong enough to imply the program's postcondition
> 3. preserved by one iteration of the loop

To read, a survey on the subject:

Loop invariants: analysis, classification, and examples. Furia et al. [10.1145/2506375]

Example

- First, fill in the pre-/post-conditions template

```
{{ X = m ∧ Y = n }}

WHILE !(X = 0) DO

    Y ::= Y - 1;;
    X ::= X - 1

END

{{ Y = n - m }}
```

Example with the template

- First, fill in the pre-/post-conditions template

```
{{ X = m ∧ Y = n }} ->>
{{ I }}
WHILE !(X = 0) DO
    {{ I /\ !(X = 0) }}
    Y ::= Y - 1;;
    X ::= X - 1
    {{ I }}
END
{{ I /\ ~ !(X = 0) }} ->>
{{ Y = n - m }}
```

Example with the template

Second, fill in the assignments

```
{{ X = m ∧ Y = n }} ->>
{{ I }}
WHILE !(X = 0) DO
    {{ I /\ !(X = 0) }} ->>
    {{ I [X |-> X-1] [Y |-> Y-1] }}
    Y ::= Y - 1;;
    {{ I [X |-> X-1] }}
    X ::= X - 1
    {{ I }}
END
{{ I /\ ~ !(X = 0) }} ->>
{{ Y = n - m }}
```

Invariant heuristics

Technique 1: Use the weakest invariant, that is let `I` be `True`.

```
{{ X = m ∧ Y = n }} ->>
{{ I }}
WHILE !(X = 0) DO
    {{ I /\ !(X = 0) }} ->>
    {{ I [X |-> X-1] [Y |-> Y-1] }}
    Y ::= Y - 1;;
    {{ I [X |-> X-1] }}
    X ::= X - 1
    {{ I }}
END
{{ I /\ ~ !(X = 0) }} ->>
{{ Y = n - m }}
```

```
{{ X = m ∧ Y = n }} ->>
{{ True }}
WHILE !(X = 0) DO
    {{ True  /\ !(X = 0) }} ->>
    {{ True [X |-> X-1] [Y |-> Y-1] }}
    Y ::= Y - 1;;
    {{ True [X |-> X-1] }}
    X ::= X - 1
    {{ True }}
END
{{ True /\ ~ !(X = 0) }} ->>
{{ Y = n - m }}
```

Invariant heuristics

Technique 1: Use the weakest invariant, that is let `I` be `True`.

```
{{ X = m ∧ Y = n }} ->>
{{ I }}
WHILE !(X = 0) DO
    {{ I /\ !(X = 0) }} ->>
    {{ I [X |-> X-1] [Y |-> Y-1] }}
    Y ::= Y - 1;;
    {{ I [X |-> X-1] }}
    X ::= X - 1
    {{ I }}
END
{{ I /\ ~ !(X = 0) }} ->>
{{ Y = n - m }}
```

```
{{ X = m ∧ Y = n }} ->>
{{ True }}
WHILE !(X = 0) DO
    {{ True  /\ !(X = 0) }} ->>
    {{ True [X |-> X-1] [Y |-> Y-1] }}
    Y ::= Y - 1;;
    {{ True [X |-> X-1] }}
    X ::= X - 1
    {{ True }}
END
{{ True /\ ~ !(X = 0) }} ->>
{{ Y = n - m }}
```

In this example it fails, as `X <> 0 ->> Y = n - m` is unprovable!

Invariant heuristics

Technique 2: Use the loop's post-condition, that is let `I` be `Y = n - m`.

```
{{ X = m ∧ Y = n }} ->>
{{ I }}
WHILE !(X = 0) DO
    {{ I /\ !(X = 0) }} ->>
    {{ I [X |-> X-1] [Y |-> Y-1] }}
    Y ::= Y - 1;;
    {{ I [X |-> X-1] }}
    X ::= X - 1
    {{ I }}
END
{{ I /\ ~ !(X = 0) }} ->>
{{ Y = n - m }}
```

```
{{ X = m ∧ Y = n }} ->>
{{ Y = n - m }}
WHILE !(X = 0) DO
    {{ Y = n - m  /\ !(X = 0) }} ->>
    {{ Y - 1 = n - m }}
    Y ::= Y - 1;;
    {{ Y = n - m [X |-> X-1] }}
    X ::= X - 1
    {{ Y = n - m }}
END
{{ Y = n - m /\ ~ !(X = 0) }} ->>
{{ Y = n - m }}
```

Invariant heuristics

Technique 2: Use the loop's post-condition, that is let `I` be `Y = n - m`.

```
{{ X = m ∧ Y = n }} ->>
{{ I }}
WHILE !(X = 0) DO
    {{ I /\ !(X = 0) }} ->>
    {{ I [X |-> X-1] [Y |-> Y-1] }}
    Y ::= Y - 1;;
    {{ I [X |-> X-1] }}
    X ::= X - 1
    {{ I }}
END
{{ I /\ ~ !(X = 0) }} ->>
{{ Y = n - m }}
```

```
{{ X = m ∧ Y = n }} ->>
{{ Y = n - m }}
WHILE !(X = 0) DO
    {{ Y = n - m  /\ !(X = 0) }} ->>
    {{ Y - 1 = n - m }}
    Y ::= Y - 1;;
    {{ Y = n - m [X |-> X-1] }}
    X ::= X - 1
    {{ Y = n - m }}
END
{{ Y = n - m /\ ~ !(X = 0) }} ->>
{{ Y = n - m }}
```

In this example it fails, `Y` changes during the loop, while `m` and `n` are constant. *Idea: check how the values of `Y` and `X` relate to each other (sample their values by executing the program).*

Invariant heuristics

Technique 3: Sample the variables mentioned in the post-condition and think of what would their value be in the i-th iteration? Let `I` be `Y - X = n - m`.

```
{{ X = m ∧ Y = n }} ->>
{{ I }}
WHILE !(X = 0) DO
    {{ I /\ !(X = 0) }} ->>
    {{ I [X |-> X-1] [Y |-> Y-1] }}
    Y ::= Y - 1;;
    {{ I [X |-> X-1] }}
    X ::= X - 1
    {{ I }}
END
{{ I /\ ~ !(X = 0) }} ->>
{{ Y = n - m }}
```

```
{{ X = m ∧ Y = n }} ->>
{{ Y - X = n - m }}
WHILE !(X = 0) DO
    {{ Y - X = n - m  /\ !(X = 0) }} ->>
    {{ (Y - 1) - (X - 1) = n - m }}
    Y ::= Y - 1;;
    {{ Y - (X - 1) = n - m [X |-> X-1] }}
    X ::= X - 1
    {{ Y - X = n - m }}
END
{{ Y - X = n - m /\ ~ !(X = 0) }} ->>
{{ Y = n - m }}
```

Summary

- Axiomatic Hoare Logic
- Program verification using Hoare logic