Programming Paradigms

**Functional.** (Haskell, SML, OCaml, . . .)
- main paradigm: *functions* that *don’t* rely on state
- main ingredient: *recursion*

**Imperative.** (C, Java, Algol, (Visual) Basic, . . .)
- main paradigm: *operations* that *do* manipulate state.
- main ingredient: *loops*
Example: From Recursion to Loops

In Haskell.

```haskell
fact_tr :: Int -> Int -> Int
fact_tr 0 acc = acc
fact_tr n acc = fact_tr (n-1) (acc * n)
fact n = fact_tr n 1
```

In Java.

```java
public static int fact (int n) {
    int acc = 1;
    while (n > 0) { acc = acc * n; n = n-1; }
    return acc;
}
```

Main Difference.

- programs are not simple equations any more
- need to keep track of *changing* values of variables
Verification for Imperative Languages

Main Ingredients.
- properties of program states
- commands that modify state.

Description of both.
- properties of states: formulae, e.g. $x < 0 \land y > x$
- commands are taken from the programming language
- formal rules that tell us how to manipulate both.

Hoare Logic ties in program states and formulae:

\[ \{P\} \text{ program } \{Q\} \]

“Running program in a state that satisfies $P$ gives a state that satisfies $Q$”
C. A. R. (Tony) Hoare

The inventor of this week’s logic is also famous for inventing the **Quicksort** algorithm in 1960 - when he was just **26**! A quote:

Computer programming is an **exact science** in that all the properties of a program and all the consequences of executing it in any given environment can, in principle, be found out from the text of the program itself by means of purely **deductive reasoning**.
Logic = Syntax + Semantics + Calculus

**Example.** Propositional Logic

- syntax: atomic propositions $p$, $q$, $r$, ... + $\land$, $\lor$, $\rightarrow$ and $\neg$
- semantics: truth tables
- calculus: natural deduction.

**Hoare Logic.**

- syntax: triples $\{P\}$ program $\{Q\}$
- semantics: $P$ in pre-state implies $Q$ in post-state
- calculus: Hoare Logic

**Q.** What are pre/post conditions *precisely*? what are the programs? What about termination?
Q. In a Hoare triple \{P\} program \{S\}, what are the programs?

A. We roll our own: a very simple Java-like language

**Assignment** – \texttt{x := e}

where \texttt{x} is a variable, and \texttt{e} is an expression built from variables and arithmetic that returns a number, e.g. \(2 + 3\), \(x \times y + 1\...\)

**Sequencing** – \texttt{S\_1; S\_2}

**Conditional** – \texttt{if \ b \ then \ S\_1 \ else \ S\_2}

where \texttt{b} is an expression built from variables, arithmetic and logic that returns a \texttt{boolean} (true or false), e.g. \(y < 0\), \(x \neq y \land z = 0\...\)

**While** – \texttt{while \ b \ do \ S}
A Note on (the lack of) Aliasing

**Assignments** $x := y$ *copy values*

No Aliasing, i.e. $x$ and $y$ point to the same region in memory
Syntax of Hoare Logic: Assertions

**Q.** How do we describe *properties* of states?

- *states* are determined by the values of program variables
- here: states will store *numbers* only.

**Properties of States.** propositional formulae built from variables, numbers and basic arithmetic:

- \( x = 3 \);
- \( x = y \);
- \( x \neq y \);
- \( x > 0 \);
- \( x \leq (y^2 + 1\frac{3}{4}) \);
- etc...
Propositional Logic to combine simple assertions, e.g.

- $x = 4 \land y = 2$;
- $x < 0 \lor y < 0$;
- $x > y \rightarrow x = 2 \times y$;
- $True$;
- $False$.

The last two logical constructions - $True$ and $False$ - will prove particularly useful, as we'll later see.

Alternative. Could use first order logic – more expressive power.
Anatomy of a Hoare Triple

\{P\} \text{ program } \{Q\}

- program is a simple program written using assignments, conditionals, while and sequencing
- pre/post conditions are propositional formulae built from arithmetical relations

**Semantics.** A Hoare triple \{P\} \text{ program } \{Q\} is valid if
  - whenever we run program in a state that satisfies \(P\)
  - \textit{and} the program terminates, then the post-state satisfies \(Q\)
Example Statements in Hoare Logic

\[\{x > 0\} \quad y := 0 - x \quad \{y < 0 \land x \neq y\}\]

If \((x > 0)\) is true before \(y := 0 - x\) is executed
then \((y < 0 \land x \neq y)\) is true afterwards.

Here:

- \((x > 0)\) is the precondition;
- \(y := 0 - x\) is a (very simple) code fragment;
- \((y < 0 \land x \neq y)\) is the postcondition.

Hoare logic will provide the rules to prove this.
Hoare’s Notation – the Definition

The **Hoare triple**:

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

means:

- **If** \( P \) is true in the initial state
- **and** \( S \) terminates
- **then** \( Q \) will hold in the final state.

**Examples:**

1. \[ \{ x = 2 \} \; x := x+1 \; \{ x = 3 \} \]
2. \[ \{ x = 2 \} \; x := x+1 \; \{ x = 5000 \} \]
3. \[ \{ x > 0 \} \; y := 0 - x \; \{ y < 0 \land x \neq y \} \]

(Hoare Triples can be true or false)
Some Hoare Triples

Q. Under what conditions are the following Hoare Triples valid?

1. \{True\} program \{True\}
2. \{True\} program \{False\}
3. \{False\} program \{True\}
4. \{False\} program \{False\}
Some Hoare Triples

Q. Under what conditions are the following Hoare Triples valid?

1. \{True\} program \{True\}
2. \{True\} program \{False\}
3. \{False\} program \{True\}
4. \{False\} program \{False\}

A. Consider (precondition) \land (termination) \rightarrow (postcondition)

1. is always true (as RHS of \rightarrow is true)
2. true if program never terminates
3. always true (as RHS of \rightarrow is true)
4. always true (as LHS of \rightarrow is false)
A Larger Hoare Triple

\{n \geq 0\}

\text{fact} := 1;
\text{k} := n;
\text{while (k>0)}
  \text{fact} := \text{fact} \times \text{k};
  \text{k} := \text{k}-1

\{\text{fact} = n!\}

**Q1.** is this Hoare triple true or false?
A Larger Hoare Triple

\{n \geq 0\}

\text{fact} := 1;
\text{k} := n;
\text{while (k>0)}
  \text{fact} := \text{fact} \times k;
  \text{k} := \text{k}-1

\{\text{fact} = n!\}

Q1. is this Hoare triple true or false?

Q2. what if \( n < 0 \) initially?
Partial Correctness

Partial Correctness.
A program is *partially correct* if it gives the right answer whenever it terminates.

Hoare Logic (in the form discussed now) (only) proves partial correctness.

Total Correctness.
A program is * totalement correct* if it always terminates and gives the right answer.

Example.

\[ \{ x = 1 \} \text{ while } x=1 \text{ do } y:=2 \quad \{ x = 3 \} \]

is *true* in Hoare logic semantics (just because the loop never terminates).
Partial Correctness is OK

Why not insist on termination?

- We *may not want* termination.

\[ \{ True \} \text{ webserv} \{ \text{very good reason} \} \]

- Not accounting for termination makes things simpler.
- We can add termination assertions (next week)
Specification vs Verification

Hoare triples allow us to say something about the *intended effect* of the code.

**Q.** How do we *make sure* that the code validates these assertions?

**A1.** Testing. For example, for \( \{P\} \) program \( \{Q\} \):

```plaintext
assert (P);
program;
assert (Q);
```

- does this catch *all* possible errors?
- How to structure test cases? Changes of variable values?

**A2.** Proving. Show that \( \{P\} \) program \( \{Q\} \) is true *for all* states.

**Hoare Calculus.**

- a collection of *rules and procedures* for (formally) manipulating the (language of) triples.

(Just like ND for classical propositional logic . . .)
The Assignment Axiom (Rule 1/6)

Rules for proving correctness of programs:
- one rule per construct (assignment, sequencing, if, while)
- two rules to glue things together

Assignment Rule.
- assignment \( x := e \) change state
- reflect this in pre/post condition

Terminology
- Suppose \( Q(x) \) is a predicate involving a variable \( x \),
- Then \( Q(e) \) indicates the same formula with all occurrences of \( x \) replaced by the expression \( e \).

The Rule.
\[
\{ Q(e) \} \ x := e \ \{ Q(x) \}
\]
The Assignment Axiom – Intuition

\[ \{ Q(e) \} \ x := e \ \{ Q(x) \} \]

Before / After

- want \( x \) to have property \( Q \) after assignment
- then property \( Q \) must hold for the value \( e \) before assignment

Q. Why is this backwards? Shouldn’t it be

\[ \{ Q(x) \} \ x := e \ \{ Q(e) \} \]

Counterexample. precondition \( x = 0 \), assignment \( x := 1 \)

\[ \{ x = 0 \} \ x := 1 \ \{ 1 = 0 \} \]

which says “if \( x = 0 \) initially and \( x := 1 \) terminates then \( 1 = 0 \) finally”
Work from the Goal, ‘Backwards’

**Forward Reasoning.** Not usually helpful
- start at the precondition, work your way down to the postcondition
- not the best way – cf. e.g. doing natural deduction proofs

**Backwards Reasoning**
- start with the goal (postcondition)
- work your way back up to the precondition

**Example.**

\[
\{ Q(e) \} \ x := e \ \{ Q(x) \}
\]

- start with postcondition, *copy* it over to precondition
- *replace* all occurrences of \( x \) with \( e \).

postcondition may have no, one, or many occurrences of \( x \) in it; all get replaced
Example 1 of \( \{ Q(e) \} \) \( x := e \ \{ Q(x) \} \)

**Code Fragment.** \( x := 2 \), postcondition \( y = x \).
- copy \( y = x \) over to the precondition
- replace all occurrences of \( x \) with 2

**Formally.**

\[
\{ y = 2 \} \ x := 2 \ \{ y = x \}
\]

is an instance of the assignment axiom.
Example 2 of \( \{ Q(e) \} \ x := e \ \{ Q(x) \} \)

**Code Fragment.** \( x := x + 1 \), postcondition \( y = x \).

As before.

\[ \{ y = x + 1 \} \ x := x + 1 \ \{ y = x \} \]

is an instance of the assignment axiom.
Example 3 of \{Q(e)\} \ x := e \ \{Q(x)\}

Q. How do we prove

\{y > 0\} \ x := y + 3 \ \{x > 3\} ?

A.

1. Start with the postcondition \(x > 3\) and apply the axiom:

\{y + 3 > 3\} \ x := y + 3 \ \{x > 3\}

2. Use the fact that \(y + 3 > 3\) is equivalent to \(y > 0\) to get our result.

Equivalent Predicates.

Can always replace predicates by equivalent predicates, label with \textit{precondition equivalence}, or \textit{postcondition equivalence}.
Proving the Assignment Axiom sound w.r.t. semantics

Assignment Axiom.

\[ \{ Q(e) \} \ x := e \ \{ Q(x) \} \]

Justification.

- Let \( v \) be the value of expression \( e \) in the initial state.
- If \( Q(e) \) is true initially, then so is \( Q(v) \).
- Since the variable \( x \) has value \( v \) after the assignment (\textit{and nothing else is changed in the state}), \( Q(x) \) must be true after that assignment.
The Assignment Axiom is Optimal

Proof Strength. The assignment axiom is as strong as possible.

\[
\{Q(e)\} \ x := e \ \{Q(x)\}
\]

Meaning?
If \(Q(x)\) holds after the assignment then \(Q(e)\) MUST have held before.

- Suppose \(Q(x)\) is true after the assignment.
- If \(v\) is the value assigned, \(Q(v)\) is true after the assignment.
- Since it is only the value of \(x\) that is changed, and the predicate \(Q(v)\) does not involve \(x\), \(Q(v)\) must also be true before the assignment.
- Since \(v\) was the value of \(e\) before the assignment, \(Q(e)\) is true initially.
A non-example

What if we wanted to prove

\[
\{y = 2\} \ x := y \ \{x > 0\}
\]

This is clearly true. But our assignment axiom doesn’t get us there:

\[
\{y > 0\} \ x := y \ \{x > 0\}
\]

**Problem.**

cannot just replace \(y > 0\) with \(y = 2\) either - they are not equivalent.

**Solution.**

Need a new Hoare logic rule that allows for manipulation of pre (and post) conditions.
Weak and Strong Predicates

Stronger.
A predicate $P$ is *stronger* than $Q$ if $P$ implies $Q$.

Weaker.
$Q$ is *weaker* than $P$ if $P$ is stronger than $Q$.

Intuition. If $P$ is stronger than $Q$, then
- $P$ is *more restrictive*, i.e. holds in fewer situations
- $Q$ holds in *more* cases than $P$, including all cases where $P$ holds.
- stronger predicates convey *more* information than weaker predicates.

Q. Can you give me an example of a *really strong* predicate?

Example.
- *I will keep unemployment below 3%* is stronger than
- *I will keep unemployment below 15%*.
- The *strongest* possible statement is *False* (unemployment below 0%)
- The *weakest* possible statement is *True* (unemployment at or below 100%)
Weak and Strong in Pictures

weak, e.g. animal

strong, e.g. marsupial
Strong Postconditions

**Example.**

- \((x = 6) \implies (x > 0)\), so \((x = 6)\) is *stronger* than \((x > 0)\)
- The Hoare triple:

  \[
  \{x = 5\} \ x := x + 1 \ {\{x = 6\}}
  \]

  says *more* about the code than does:

  \[
  \{x = 5\} \ x := x + 1 \ {\{x > 0\}}
  \]

**Strong Postconditions** in general

- if postcondition \(Q_1\) is *stronger* than \(Q_2\), then \(\{P\} S \{Q_1\}\) is a *stronger* statement than \(\{P\} S \{Q_2\}\).
- if postcondition \(x = 6\) is *stronger* than postcondition \(x > 0\), then \(\{P\} S \{x = 6\}\) is a *stronger* statement than \(\{P\} S \{x > 0\}\)
Weak Preconditions

Formula Example.
- condition \((x > 0)\) says *less* about a state than \(x = 5\).
- so \(x > 0\) is a *weaker* condition than \(x = 5\) since \(x = 5\) implies \(x > 0\).

Hoare Triple Example
- the Hoare triple \(\{x > 0\} \ x := x + 1 \ {x > 1}\) says *more* about the code than \(\{x = 5\} \ x := x + 1 \ {x > 1}\)
- this is because it says something about *more* situations

Weak Preconditions
- If precondition \(P_1\) is *weaker* than \(P_2\), then \(\{P_1\} S \{Q\}\) is *stronger* than \(\{P_2\} S \{Q\}\).
- if precondition \(x > 0\) is *weaker* than precondition \(x = 5\), then \(\{x > 0\} S \{Q\}\) is *stronger* than \(\{x = 5\} S \{Q\}\).
Weak/Strong Pre/Postconditions

Precondition Strengthening. If $P_2$ is stronger than $P_1$, then $\{P_2\} S \{Q\}$ is true whenever $\{P_1\} S \{Q\}$ is true.

Proof. Assume that $\{P_1\} S \{Q\}$ is true.
- Assume that we run $S$ in a state that satisfies $P_2$
- but since $P_2$ is stronger than $P_1$, we have $P_2 \rightarrow P_1$
- hence $S$ also satisfies $P_1$ so that $Q$ is true afterwards.

Postcondition Weakening. If $Q_1$ is a stronger postcondition than $Q_2$, then $\{P\} S \{Q_2\}$ is true whenever $\{P\} S \{Q_1\}$ is true.

Proof. Assume that $\{P\} S \{Q_1\}$ is true.
- assumes that we run $S$ in a state that satisfies $P$ and that $S$ terminates
- this will lead to a post-state that satisfies $Q_1$
- but because $Q_1$ is stronger than $Q_2$, we have $Q_1 \rightarrow Q_2$
- hence the post-state will also satisfy $Q_2$. 
Proof rule for Strengthening Preconditions (Rule 2/6)

Q. How do we reflect this in the Hoare calculus?

A. We codify this in terms of proof rules that we can apply.

Precondition Strengthening. Interpretation. If the premises are provable then so is the conclusion

\[
\frac{P_s \rightarrow P_w}{\{P_w\} S \{Q\}}
\]

\[
\{P_s\} S \{Q\}
\]

Example by pattern matching

\[
y = 2 \rightarrow y > 0 \quad \{y > 0\} x := y \quad \{x > 0\}
\]

\[
\{y = 2\} x := y \quad \{x > 0\}
\]

Precondition Equivalence. If \( P_1 \leftrightarrow P_2 \) then both \( P_1 \rightarrow P_2 \) and \( P_2 \rightarrow P_1 \).
Proof rule for Weakening Postconditions (Rule 3/6)

Postcondition Weakening.

Interpretation. If the premises are provable then so is the conclusion

\[
\begin{align*}
\{P\} &\implies\{Q_s\} & Q_s &\implies Q_w \\
\{P\} &\implies\{Q_w\}
\end{align*}
\]

Example by pattern matching

\[
\begin{align*}
\{x > 2\} &\text{ } x := x + 1 \text{ } \{x > 3\} & x > 3 &\implies x > 0 \\
\{x > 2\} &\text{ } x := x + 1 \text{ } \{x > 0\}
\end{align*}
\]

Postcondition Equivalence. If \(Q_1 \leftrightarrow Q_2\) then \(Q_1 \implies Q_2\) and \(Q_2 \implies Q_1\).

i.e. \(Q_s \implies Q_w \land Q_w \implies Q_s\)
Sequencing (Rule 4/6)

Sequencing.
- execute commands one after another, each one manipulates the state
- need to think about the overall effect of state change

Sequencing as a proof rule

*Interpretation.* If the premises are provable then so is the conclusion

\[
\begin{array}{c}
\{P\} S_1 \{Q\} \\
\{Q\} S_2 \{R\}
\end{array}
\]

\[
\{P\} S_1; S_2 \{R\}
\]

Example.

\[
\begin{array}{c}
\{x > 2\} x := x + 1 \{x > 3\} \\
\{x > 3\} x := x + 2 \{x > 5\}
\end{array}
\]

\[
\{x > 2\} x := x + 1; x := x + 2 \{x > 5\}
\]
Interlude: Laying out a proof

**Layout Problem.** Assertions may depend on more than one premise.

**Linear Layout.**

1. \( \{ x + 2 > 5 \} \quad x := x + 2 \quad \{ x > 5 \} \) \hfill \text{(Assignment)}

2. \( \{ x > 3 \} \quad x := x + 2 \quad \{ x > 5 \} \) \hfill \text{(Precondition Equivalence, 1)}

3. \( \{ x + 1 > 3 \} \quad x := x + 1 \quad \{ x > 3 \} \) \hfill \text{(Assignment)}

4. \( \{ x > 2 \} \quad x := x + 1 \quad \{ x > 3 \} \) \hfill \text{(Precondition Equivalence, 1)}

5. \( \{ x > 2 \} \quad x := x + 1; x := x + 2 \quad \{ x > 5 \} \) \hfill \text{(Sequence, 4, 2)}

Note the *numbered proof steps* and *justifications.*
Finding a Proof

**Q.** Where do we get the “condition in the middle” from?

\[
\begin{array}{c}
\{P\} S_1 \{Q\} \quad \{Q\} S_2 \{R\} \\
\{P\} S_1 ; S_2 \{R\}
\end{array}
\]

- overall precondition \( P \) and overall postcondition \( R \) are given
- sequencing requires us to find a gluing condition \( Q \)

**A.** Start with the goal \( R \) and work backwards (as usual)

\[
\begin{array}{c}
\{x > 2\} \ x := x + 1 \quad \{Q\} \\
\{x > 2\} \ x := x + 1 ; x := x + 2 \quad \{x > 5\}
\end{array}
\]

\[
\begin{array}{c}
\{Q\} \quad \{Q\} \ x := x + 2 \quad \{x > 5\}
\end{array}
\]
An example with precondition strengthening

**Goal.** Prove that the following is true:

\[
\{x = 3\} \quad x := x + 1; x := x + 2 \quad \{x > 5\}
\]

**First Steps** in linear layout

5. \(\{x > 2\} \quad x := x + 1; x := x + 2 \quad \{x > 5\}\)  
   (See earlier slide)

**Add the following**

6. \(x = 3 \rightarrow x > 2\)  
   (Basic arithmetic)

7. \(\{x = 3\} \quad x := x + 1; x := x + 2 \quad \{x > 5\}\)  
   (Prec. Strength. 5, 6)
Soundness of Rule for Sequences

**Lemma.** If the premises of Sequencing rule are true then so is the conclusion

**Proof.** Suppose the premises \{P\}S_1\{Q\} and \{Q\}S_2\{R\} are true and let \(\sigma_0\) be an arbitrary state that satisfies \(P\).

- if we run \(S_1\) in state \(\sigma_0\) we get a state \(\sigma_1\) that satisfies \(Q\)
- if we run \(S_2\) in state \(\sigma_1\) we get a state \(\sigma_2\) that satisfies \(R\)
- but executing \(S_1; S_2\) just means execute \(S_1\) first and then \(S_2\)
- hence we end up in a state that satisfies \(R\)

**Q.** What about termination?
Proof Rule for Conditionals (Rule 5/6)

**Conditionals.**

\[
\text{if } b \text{ then } S_1 \text{ else } S_2
\]

- \(b\) is a *boolean condition* that evaluates to *true* or *false*
- the value of \(b\) may depend on the *program state*

**Informal Reasoning.** Case split
- if \(b\) evaluates to *true*, then run \(S_1\)
- if \(b\) evaluates to *false*, then run \(S_2\).

**Additional Precondition.**
- in the *if*-branch, additionally know that \(b\) is *true*
- in the *then*-branch, additionally know that \(b\) is *false*

**Q.** What is / are the “right” premise(s) for the *if*-rule

\[
\frac{}{\{P\} \text{ if } b \text{ then } S_1 \text{ else } S_2 \{Q\}}
\]
Proof Rule for Conditionals

Proof Rule

\[
\begin{align*}
{P \land b} & \quad S_1 \quad \{Q\} \quad {P \land \neg b} & \quad S_2 \quad \{Q\} \\
\{P\} & \quad \text{if } b \text{ then } S_1 \text{ else } S_2 \quad \{Q\}
\end{align*}
\]

Justification.

- When a conditional is executed, either \( S_1 \) or \( S_2 \) is executed.
- Therefore, if the \textit{conditional} is to establish \( Q \), \textit{both} \( S_1 \) and \( S_2 \) must establish \( Q \).
- Similarly, if the precondition for the \textit{conditional} is \( P \), then it must also be a precondition for the two branches \( S_1 \) and \( S_2 \).
- The choice between \( S_1 \) and \( S_2 \) depends on evaluating \( b \) \textit{in the initial state}, so we can also assume \( b \) to be a precondition for \( S_1 \) and \( \neg b \) to be a precondition for \( S_2 \).
Example of Conditional Rule

$$\begin{align*}
\{P \land b\} & \quad S_1 \quad \{Q\} \quad \{P \land \neg b\} \quad S_2 \quad \{Q\} \\
\{P\} & \quad \text{if } b \text{ then } S_1 \quad \text{else } S_2 \quad \{Q\}
\end{align*}$$

Example. We want to show that the following is true

$$\{x > 2\} \quad \text{if } x > 2 \text{ then } y := 1 \quad \text{else } y := -1 \quad \{y > 0\}$$

Using the conditional rule (pattern matching)

$$\begin{align*}
\{x > 2 \land x > 2\} & \quad y := 1 \quad \{y > 0\} \quad \{x > 2 \land \neg(x > 2)\} \quad y := -1 \quad \{y > 0\} \\
\{x > 2\} & \quad \text{if } x > 2 \text{ then } y := 1 \quad \text{else } y := -1 \quad \{y > 0\}
\end{align*}$$

Precondition Equivalence means that we need to show:

(1)  $$\{x > 2\} \quad y := 1 \quad \{y > 0\}$$  
(2)  $$\{\text{False}\} \quad y := -1 \quad \{y > 0\}$$
Example In Full

Show. \{x > 2\} \text{ if } x > 2 \text{ then } y := 1 \text{ else } y := -1 \{y > 0\}

Proof in linear layout:

1. \{1 > 0\} y := 1 \{y > 0\} \quad \text{(Assignment)}
2. (1 > 0) \leftrightarrow True \quad \text{(Prop. Logic)}
3. \{True\} y := 1 \{y > 0\} \quad \text{(Prec. Equivalence, 2, 1)}
4. (x > 2) \rightarrow True \quad \text{(Prop. Logic)}
5. \{x > 2\} y := 1 \{y > 0\} \quad \text{(premise (1))} \quad \text{(Prec. Stre., 3, 4)}
6. \{-1 > 0\} y := -1 \{y > 0\} \quad \text{(Assignment)}
7. False \leftrightarrow (-1 > 0) \quad \text{(Prop. Logic)}
8. \{False\} y := -1 \{y > 0\} \quad \text{(premise(2))} \quad \text{(Prec. Eq)}
9. \{x > 2\} \text{ if } x > 2 \text{ then } y := 1 \text{ else } y := -1 \{y > 0\} \quad \text{(Conditional, 5, 8)}
Interlude: Conditionals Without ‘Else’

**Conditionals** are complete in the sense that they include an else-branch:

```
if b then S₁ else S₂
```

**Q.** Our language *could* have statements of the form

```
if b then S
```

What would be the rule?
Interlude: Conditionals Without ‘Else’

**Conditionals** are complete in the sense that they include an else-branch:

\[
\text{if } b \text{ then } S_1 \text{ else } S_2
\]

**Q.** Our language *could* have statements of the form

\[
\text{if } b \text{ then } S
\]

What would be the rule?

**A.** Conditionals without else are equivalent to

\[
\text{if } b \text{ then } S \text{ else (do nothing)}
\]

**Conditional Rule.**

\[
\frac{
\{ P \land b \} \quad S \quad \{ Q \} \\
\{ P \land \neg b \} \quad \text{do nothing} \quad \{ Q \}
}{
\{ P \} \quad \text{if } b \text{ then } P \quad \{ Q \}
} 
\]
Q. How do we establish the following? **Conditional Rule.**

\[
\begin{align*}
\{P \land P\} & \quad S \quad \{Q\} \\
\{P \land \neg b\} & \quad \text{do nothing} \quad \{Q\} \\
\{P\} & \quad \text{if } b \text{ then } P \quad \{Q\}
\end{align*}
\]

Q1. How about do nothing?

A. Easy: \{P\} do nothing \{P\} is always true.

**Precondition Strengthening** to the rescue:

\[
\begin{align*}
\{P \land b\} & \quad S \quad \{Q\} \\
(P \land \neg b) & \quad \rightarrow \quad Q \\
\{P\} & \quad \text{if } b \text{ then } S \quad \text{else } x := x \quad \{Q\}
\end{align*}
\]
Finding a Proof

**Q.** How do we prove that

\[
\{ x = 3 \} \ x := x + 1; x := x + 2 \quad \{ x > 5 \}
\]

**A.** Use sequencing rule

\[
\begin{array}{c}
\{ P \} \\
S_1 \\
\{ Q \} \\
\{ Q \} \\
S_2 \\
\{ R \}
\end{array}
\]

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

\[
\{ P \} \quad S_1 \quad ; \quad S_2 \quad \{ R \}
\]

**Concrete Instance.**

\[
\begin{align*}
\{ x = 3 \} & \quad x := x + 1 \quad \{ Q \} \\
\{ x = 3 \} & \quad x := x + 1; x := x + 2 \quad \{ x > 5 \}
\end{align*}
\]

\[
\{ x = 3 \} \quad x := x + 1; x := x + 2 \quad \{ x > 5 \}
\]

\[
\text{Seq}
\]
Finding a Proof

**Goal.** Prove that the following is true.

\[ \{x = 3\} \; x := x + 1; \; x := x + 2 \; \{x > 5\} \]

**First Take.** Apply assignment axiom \( \{Q(e)\} x := e \{Q(x)\} \)

**Q.** What rule could (?) be?

\[
\begin{align*}
\{x = 3\} \; x := x + 1 & \quad \{Q\} \\
\{x = 3\} \; x := x + 1; \; x := x + 2 & \quad \text{Seq}
\end{align*}
\]
Finding a Proof

**Goal.** Prove that the following is true:

\[
\{ x = 3 \} \ x := x + 1; x := x + 2 \quad \{ x > 5 \}
\]

**A.** Putting \( Q = x > 3 \) would mean that (?) is precondition equivalence

\[
\frac{\{ x = 3 \} \ x := x + 1 \quad \{ x > 3 \} \quad \{ x + 2 > 5 \} \ x := x + 2 \quad \{ x > 5 \} \quad \{ x > 3 \} \ x := x + 2 \quad \{ x > 5 \}}{\{ x = 3 \} \ x := x + 1; x := x + 2 \quad \{ x > 5 \}} \quad \text{PreEq} \quad \text{Seq}
\]
**Goal.** Prove that the following is true:

\[
\{x = 3\} \quad x := x + 1; x := x + 2 \quad \{x > 5\}
\]

**Second Take.** Can apply the assignment axiom \(\{Q(e)\}x := e\{Q(x)\}\)

**Q.** What rule could (\(\_\)) be?

\[
\frac{\{x + 1 > 3\} \quad x := x + 1 \quad \{x > 3\}}{\{x = 3\} \quad x := x + 1 \quad \{x > 3\}} \quad ? \quad \frac{\{x + 2 > 5\} \quad x := x + 2 \quad \{x > 5\}}{\{x > 3\} \quad x := x + 2 \quad \{x > 5\}}
\]

PreEq

\[
\frac{\{x = 3\} \quad x := x + 1; x := x + 2 \quad \{x > 5\}}{\{x > 5\}}
\]

Seq
A. Let’s try precondition equivalence again:

\[ x > 2 \iff x + 1 > 3 \]

\[
\frac{\{x + 1 > 3\} \quad x := x + 1 \quad \{x > 3\}}{
\{x > 2\} \quad x := x + 1 \quad \{x > 3\}} \quad \text{PreEq}
\]

\[
\frac{\{x = 3\} \quad x := x + 1 \quad \{x > 3\}}{
\{x > 2 > 5\} \quad x := x + 2 \quad \{x > 5\}} \quad ?
\]

\[
\frac{\{x > 3\} \quad x := x + 2 \quad \{x > 5\}}{
\{x > 3\} \quad x := x + 2 \quad \{x > 5\}} \quad \text{Seq}
\]

\[
\{x = 3\} \quad x := x + 1; x := x + 2 \quad \{x > 5\}
\]

Q. There’s still something missing. What is (?) now?
Finding a Proof

A. $x = 3$ implies $x > 2$ so “?” can be precondition strengthening.

Precondition Strengthening.

$$
P_s \rightarrow P_w \quad \{ P_w \} \downarrow \{ Q \} \quad \{ P_s \} \downarrow \{ Q \}
$$

Complete Proof as a tree

\[
\begin{array}{c}
\{ x + 1 > 3 \} \ x := x + 1 \quad \{ x > 3 \} \\
\{ x > 2 \} \ x := x + 1 \quad \{ x > 3 \} \\
\{ x = 3 \} \ x := x + 1 \quad \{ x > 3 \}
\end{array}
\]

PreEq

\[
\begin{array}{c}
\{ x + 2 > 5 \} \ x := x + 2 \quad \{ x > 5 \} \\
\{ x > 3 \} \ x := x + 2 \quad \{ x > 5 \}
\end{array}
\]

PreE

\[
\begin{array}{c}
\{ x = 3 \} \ x := x + 1; x := x + 2 \quad \{ x > 5 \}
\end{array}
\]

Seq
The Same Proof in Linear Form

1. \( \{x + 1 > 3\} \ x := x + 1 \ \{x > 3\} \)  
   (Assignment)

2. \( x > 2 \iff x + 1 > 3 \)  
   (Basic arithmetic)

3. \( \{x > 2\} \ x := x + 1 \ \{x > 3\} \)  
   (Prec. Equi. 1, 2)

4. \( x = 3 \rightarrow x > 2 \)  
   (Basic arithmetic)

5. \( \{x = 3\} \ x := x + 1 \ \{x > 3\} \)  
   (Prec. Stren. 3, 4)

6. \( \{x + 2 > 5\} \ x := x + 2 \ \{x > 5\} \)  
   (Assignment)

7. \( x > 3 \iff x + 2 > 5 \)  
   (Basic arithmetic)

8. \( \{x > 3\} \ x := x + 2 \ \{x > 5\} \)  
   (Prec. Equiv. 6, 7)

9. \( \{x = 3\} \ x := x + 1; x := x + 2 \ \{x > 5\} \)  
   (Seq. 5, 8)

(sections separated by horizontal lines are both premises of the sequencing rule)