• Backwards proof just involves using the rules backwards

• Given the rule

\[
\vdash S_1 \quad \ldots \quad \vdash S_n
\]
\[\vdash S\]

• Forwards proof says:
  - if we have proved \( \vdash S_1 \ldots \vdash S_n \) we can deduce \( \vdash S \)

• Backwards proof says:
  - to prove \( \vdash S \) it is sufficient to prove \( \vdash S_1 \ldots \vdash S_n \)

• Having proved a theorem by backwards proof, it is simple to extract a forwards proof
• The sequencing rule introduces a new statement $R$

$$\vdash \{P\} \ C_1 \ \{R\} \quad \vdash \{R\} \ C_2 \ \{Q\}$$

$$\vdash \{P\} \ C_1;C_2 \ \{Q\}$$

• To apply this backwards, one needs to find a suitable statement $R$

• If $C_2$ is $V := E$ then sequenced assignment gives $Q[E/V]$ for $R$

• If $C_2$ isn’t an assignment then need some other way to choose $R$

• Similarly, to use the derived While rule, must invent an invariant
Annotate First

- It is helpful to think up these statements before you start the proof and then annotate the program with them
  - the information is then available when you need it in the proof
  - this can help avoid you being bogged down in details
  - the annotation should be true whenever control reaches that point

- Example, the following program could be annotated at the points $P_1$ and $P_2$ indicated by the arrows

\[
\{ T \}
R := X;
Q := 0; \; \{ R = X \land Q = 0 \} \leftarrow P_1
\]
\[
\text{WHILE } Y \leq R \text{ DO } \{ X = R + Y \times Q \} \leftarrow P_2
\]
\[
(R := R - Y; \; Q := Q + 1)
\]
\[
\{ X = R + Y \times Q \land R < Y \}\]
NEW TOPIC: Mechanizing Program Verification

- The architecture of a simple program verifier will be described
- Justified with respect to the rules of Floyd-Hoare logic
- It is clear that
  - proofs are long and boring, even if the program being verified is quite simple
  - lots of fiddly little details to get right, many of which are trivial, e.g.
  \[
  \vdash (R=X \land Q=0) \Rightarrow (X = R + Y \times Q)
  \]
Architecture of a Verifier

- Specification to be proved
  - human expert
- Annotated specification
  - VC generator
- Set of logic statements (VCs)
  - theorem prover
- Simplified set of verification conditions
  - human expert
- End of proof
The three steps in proving \( \{P\}C\{Q\} \) with a verifier

1. The program \( C \) is annotated by inserting statements (assertions) expressing conditions that are meant to hold at intermediate points
   - tricky: needs intelligence and good understanding of how the program works
   - automating it is an artificial intelligence problem

2. A set of logic statements called verification conditions (VCs) is then generated from the annotated specification
   - this is purely mechanical and easily done by a program

3. The verification conditions are proved
   - needs automated theorem proving (i.e. more artificial intelligence)

To improve automated verification one can try to
- reduce the number and complexity of the annotations required
- increase the power of the theorem prover
- still a research area
• It will be shown that
  
  • if one can prove all the verification conditions generated from \( \{P\} C \{Q\} \)
  
  • then \( \vdash \{P\} C \{Q\} \)

• Step 2 converts a verification problem into a conventional mathematical problem

• The process will be illustrated with:

\[
\{T\}
\begin{align*}
R & := X; \\
Q & := 0; \\
\text{WHILE } Y \leq R \text{ DO} & \\
& \quad (R := R - Y; \ Q := Q + 1) \cr
\{X = R + Y \times Q \land R < Y\}
\end{align*}
\]
• Step 1 is to insert annotations $P_1$ and $P_2$

$$
\{T\} \\
R := X; \\
Q := 0; \{R = X \land Q = 0\} \leftarrow P_1 \\
\text{WHILE } Y \leq R \text{ DO } \{X = R + Y \times Q\} \leftarrow P_2 \\
(R := R - Y; \ Q := Q + 1) \\
\{X = R + Y \times Q \land R < Y\}
$$

• The annotations $P_1$ and $P_2$ state conditions which are intended to hold whenever control reaches them
Example Continued

\{ T \}
\begin{align*}
R &:= X; \\
Q &:= 0; \quad \{ R=X \land Q=0 \} \leftarrow P_1 \\
\text{WHILE } Y \leq R \text{ DO } \{ X = R+Y \times Q \} \leftarrow P_2 \\
\quad (R := R-Y; \quad Q := Q+1) \\
\{ X = R+Y \times Q \land R < Y \}
\end{align*}

- Control only reaches the point at which $P_1$ is placed once
- It reaches $P_2$ each time the WHILE body is executed
  - whenever this happens $X=R+Y\times Q$ holds, even though the values of $R$ and $Q$ vary
  - $P_2$ is an invariant of the WHILE-command
Step 2 will generate the following four verification conditions

(i) \( T \Rightarrow (X = X \land 0 = 0) \)

(ii) \( (R = X \land Q = 0) \Rightarrow (X = R + (Y \times Q)) \)

(iii) \( (X = R + (Y \times Q) \land Y \leq R) \Rightarrow (X = (R - Y) + (Y \times (Q+1))) \)

(iv) \( (X = R + (Y \times Q)) \land \neg (Y \leq R) \Rightarrow (X = R + (Y \times Q) \land R < Y) \)

Notice that these are statements of arithmetic

- the constructs of our programming language have been ‘compiled away’

Step 3 consists in proving the four verification conditions

- easy with modern automatic theorem provers
An annotation of commands

- An annotated command is a command with statements (assertions) embedded within it.

- A command is properly annotated if statements have been inserted at the following places:
  
  (i) before \( C_2 \) in \( C_1; C_2 \) if \( C_2 \) is not an assignment command.
  
  (ii) after the word \texttt{DO} in \texttt{WHILE} commands.

- The inserted assertions should express the conditions one expects to hold whenever control reaches the point at which the assertion occurs.

- Can reduce number of annotations using weakest preconditions (see later).
A properly annotated specification is a specification \( \{P\}C\{Q\} \) where \( C \) is a properly annotated command.

Example: To be properly annotated, assertions should be at points ① and ② of the specification below

\[
\begin{align*}
\{X=n\} \\
Y:=1; & \quad \leftarrow ① \\
\text{WHILE } X \neq 0 \text{ DO } & \quad \leftarrow ② \\
\quad (Y:=Y \times X; \ X:=X-1) \\
\{X=0 \land Y=n!\}
\end{align*}
\]

Suitable statements would be

at ①: \( \{Y = 1 \land X = n\} \)
at ②: \( \{Y \times X! = n!\} \)
• The VCs generated from an annotated specification \( \{P\}C\{Q\} \) are obtained by considering the various possibilities for \( C \).

• We will describe it command by command using rules of the form:

• The VCs for \( C(C_1, C_2) \) are
  
  • \( vc_1, \ldots, vc_n \)
  
  • together with the VCs for \( C_1 \) and those for \( C_2 \)

• Each VC rule corresponds to either a primitive or derived rule.
• This process will be justified by showing that $\vdash \{P\}C\{Q\}$ if all the verification conditions can be proved.

• We will prove that for any $C$
  
  • assuming the VCs of $\{P\}C\{Q\}$ are provable
  
  • then $\vdash \{P\}C\{Q\}$ is a theorem of the logic.
• The argument that the verification conditions are sufficient will be by induction on the structure of $C$

• Such inductive arguments have two parts
  
  • show the result holds for atomic commands, i.e. assignments
  
  • show that when $C$ is not an atomic command, then if the result holds for the constituent commands of $C$ (this is called the induction hypothesis), then it holds also for $C$

• The first of these parts is called the basis of the induction

• The second is called the step

• The basis and step entail that the result holds for all commands
Assignment commands
The single verification condition generated by
\[
\{P\} \ V := E \ \{Q\}
\]
is
\[
P \implies Q[E/V]
\]

• Example: The verification condition for

\[
\{X=0\} \ X := X+1 \ \{X=1\}
\]
is
\[
X=0 \implies (X+1)=1
\]
(which is clearly true)

• Note: \(Q[E/V] = \text{wlp}("V:=E", Q)\)
• We must show that if the VCs of \( \{P\} V := E \{Q\} \) are provable then \( \vdash \{P\} V := E \{Q\} \)

• Proof:
  
  • Assume \( \vdash P \Rightarrow Q[E/V] \) as it is the VC
  
  • From derived assignment rule it follows that \( \vdash \{P\} V := E \{Q\} \)
Conditionals

The verification conditions generated from

\( \{P\} \text{ IF } S \text{ THEN } C_1 \text{ ELSE } C_2 \{Q\} \)

are

(i) the verification conditions generated by

\( \{P \land S\} C_1 \{Q\} \)

(ii) the verifications generated by

\( \{P \land \neg S\} C_2 \{Q\} \)

- Example: The verification conditions for

\( \{T\} \text{ IF } X \geq Y \text{ THEN } \text{MAX:=}X \text{ ELSE } \text{MAX:=}Y \{\text{MAX=}\max(X,Y)\} \)

are

(i) the VCs for \( \{T \land X \geq Y\} \text{ MAX:=}X \{\text{MAX=}\max(X,Y)\} \)

(ii) the VCs for \( \{T \land \neg(X \geq Y)\} \text{ MAX:=}Y \{\text{MAX=}\max(X,Y)\} \)
Justification for the Conditional VCs (1)

Must show that if VCs of
\{P\} IF S THEN \(C_1\) ELSE \(C_2\) \{Q\}
are provable, then
\(\vdash \{P\} IF S THEN \(C_1\) ELSE \(C_2\) \{Q\}\)

Proof:

- Assume the VCs \(\{P \land S\} \ C_1 \ {Q}\) and \(\{P \land \neg S\} \ C_2 \ {Q}\)

- The inductive hypotheses tell us that if these VCs are provable then the corresponding Hoare Logic theorems are provable

  - i.e. by induction \(\vdash \{P \land S\} \ C_1 \ {Q}\) and \(\vdash \{P \land \neg S\} \ C_2 \ {Q}\)

  - Hence by the conditional rule \(\vdash \{P\} IF S THEN \(C_1\) ELSE \(C_2\) \{Q\}\)
• If $C_1; C_2$ is properly annotated, then either

  **Case 1:** it is of the form $C_1; \{R\}C_2$ and $C_2$ is not an assignment

  **Case 2:** it is of the form $C; V := E$

• And $C$, $C_1$ and $C_2$ are properly annotated
Sequences

1. The verification conditions generated by

\[ \{P\} C_1 \{R\} C_2 \{Q\} \]

(where \(C_2\) is not an assignment) are the union of:

(a) the verification conditions generated by \(\{P\} C_1 \{R\}\)

(b) the verifications generated by \(\{R\} C_2 \{Q\}\)

2. The verification conditions generated by

\[ \{P\} C; V := E \{Q\} \]

are the verification conditions generated by \(\{P\} C \{Q[E/V]\}\)
• **Case 1:** If the verification conditions for

\[
\{P\} C_1 ; \{R\} C_2 \{Q\}
\]

are provable

• Then the verification conditions for

\[
\{P\} C_1 \{R\}
\]

and

\[
\{R\} C_2 \{Q\}
\]

must both be provable

• Hence by induction

\[
\vdash \{P\} C_1 \{R\} \ \text{and} \ \vdash \{R\} C_2 \{Q\}
\]

• Hence by the sequencing rule

\[
\vdash \{P\} C_1 ; C_2 \{Q\}\]
Case 2: If the verification conditions for

\(\{P\} C; V := E \{Q\}\)

are provable, then the verification conditions for

\(\{P\} C \{Q[E/V]\}\)

are also provable

Hence by induction

\(\vdash \{P\} C \{Q[E/V]\}\)

Hence by the derived sequenced assignment rule

\(\vdash \{P\} C; V := E \{Q\}\)
A correctly annotated specification of a **WHILE**-command has the form

\[
\{P\} \text{WHILE } S \text{ DO } \{R\} \text{ C } \{Q\}
\]

The annotation \(R\) is called an invariant.

**WHILE-commands**

The verification conditions generated from

\[
\{P\} \text{WHILE } S \text{ DO } \{R\} \text{ C } \{Q\}
\]

are

(i) \(P \Rightarrow R\)

(ii) \(R \land \neg S \Rightarrow Q\)

(iii) the verification conditions generated by \(\{R \land S\} \text{C}\{R\}\)
Justification of \texttt{WHILE} VCs

- If the verification conditions for

\[
\{P\} \texttt{WHILE } S \texttt{ DO } \{R\} \texttt{ C } \{Q\}
\]

are provable, then

\[
\vdash P \Rightarrow R
\]

\[
\vdash (R \land \neg S) \Rightarrow Q
\]

and the verification conditions for

\[
\{R \land S\} \texttt{ C } \{R\}
\]

are provable

- By induction

\[
\vdash \{R \land S\} \texttt{ C } \{R\}
\]

- Hence by the derived \texttt{WHILE}-rule

\[
\vdash \{P\} \texttt{WHILE } S \texttt{ DO } C \{Q\}
\]
Summary

- Have outlined the design of an automated program verifier
- Annotated specifications compiled to mathematical statements
  - if the statements (VCs) can be proved, the program is verified
- Human help is required to give the annotations and prove the VCs
- The algorithm was justified by an inductive proof
  - it appeals to the derived rules
- All the techniques introduced earlier are used
  - backwards proof
  - derived rules
  - annotation
• Weakest preconditions is a theory of refinement
  • idea is to calculate a program to achieve a postcondition
  • not a theory of post hoc verification

• Non-determinism a key idea in Dijkstra’s presentation
  • start with a non-deterministic high level pseudo-code
  • refine to deterministic and efficient code

• Weakest preconditions (wp) are for total correctness

• Weakest liberal preconditions (wlp) for partial correctness

• If $C$ is a command and $Q$ a predicate, then informally:
  • $\text{wlp}(C, Q) = \text{‘The weakest predicate } P \text{ such that } \{P\} C \{Q\}\text{’}$
  • $\text{wp}(C, Q) = \text{‘The weakest predicate } P \text{ such that } [P] C [Q]\text{’}$

• If $P$ and $Q$ are predicates then $Q \Rightarrow P$ means $P$ is ‘weaker’ than $Q$
Rules for weakest preconditions

- Relation with Hoare specifications:
  \[
  \{P\} \ C \ \{Q\} \iff P \Rightarrow \wp(C, Q)
  \]
  \[
  [P] \ C \ [Q] \iff P \Rightarrow \wp(C, Q)
  \]

- Dijkstra gives rules for computing weakest preconditions:
  \[
  \wp(V := E, Q) = Q[E/V]
  \]
  \[
  \wp(C_1; C_2, Q) = \wp(C_1, \wp(C_2, Q))
  \]
  \[
  \wp(\text{IF } S \ \text{THEN } C_1 \ \text{ELSE } C_2, Q) = (S \Rightarrow \wp(C_1, Q)) \land (\neg S \Rightarrow \wp(C_2, Q))
  \]
  for deterministic loop-free code the same equations hold for \(\wlp\)

- Rule for WHILE-commands doesn’t give a first order result

- Weakest preconditions closely related to verification conditions

- VCs for \(\{P\} \ C \ \{Q\}\) are related to \(P \Rightarrow \wlp(C, Q)\)
  - VCs use annotations to ensure first order formulas can be generated
Using wlp to improve verification condition method

• If $C$ is **loop-free** then VC for $\{P\} C \{Q\}$ is $P \Rightarrow wlp(C, Q)$
  - no annotations needed in sequences!

• Cannot in general compute a **finite** formula for $wlp(\text{WHILE } S \text{ DO } C, Q)$

• The following holds

  \[
  wlp(\text{WHILE } S \text{ DO } C, Q) = \begin{cases} 
  wlp(C, wlp(\text{WHILE } S \text{ DO } C, Q)) & \text{if } S \\
  \text{else } Q \end{cases}
  \]

• Above doesn’t define $wlp(C, Q)$ as a finite statement

• Could use a hybrid VC and $wlp$ method
Strongest postconditions

• Define \( sp(C, P) \) to be ‘strongest’ \( Q \) such that \( \{P\} C \{Q\} \)
  
  • partial correctness: \( \{P\} C \{sp(C, P)\} \)
  
  • strongest means if \( \{P\} C \{Q\} \) then \( sp(C, P) \Rightarrow Q \)

• Note that \( wlp \) goes ‘backwards’, but \( sp \) goes ‘forwards’
  
  • verification condition for \( \{P\} C \{Q\} \) is: \( sp(C, P) \Rightarrow Q \)

• By ‘strongest’ and Hoare logic postcondition weakening
  
  • \( \{P\} C \{Q\} \) if and only if \( sp(C, P) \Rightarrow Q \)
Strongest postconditions for loop-free code

- Only consider loop-free code
- \( \text{sp}(V := E, P) = \exists v. V = E[v/V] \land P[v/V] \)
- \( \text{sp}(C_1 ; C_2, P) = \text{sp}(C_2, \text{sp}(C_1, P)) \)
- \( \text{sp}(<\text{IF } S \text{ THEN } C_1 \text{ ELSE } C_2, P>) = \text{sp}(C_1, P \land S) \lor \text{sp}(C_2, P \land \neg S) \)
- \( \text{sp}(V := E, P) \) corresponds to Floyd assignment axiom
- Can dynamically prune conditionals because \( \text{sp}(C, F) = F \)
- Computer strongest postconditions is symbolic execution
Computing $sp$ versus $wlp$

- **Computing $sp$ is like execution**
  - can simplify as one goes along with the ‘current state’
  - may be able to resolve branches, so can avoid executing them
  - Floyd assignment rule complicated in general
  - $sp$ used for symbolically exploring ‘reachable states’
    (related to model checking)

- **Computing $wlp$ is like backwards proof**
  - don’t have ‘current state’, so can’t simplify using it
  - can’t determine conditional tests, so get big if-then-else trees
  - Hoare assignment rule simpler for arbitrary formulae
  - $wlp$ used for improved verification conditions
Using sp to generate verification conditions

- If $C$ is loop-free then VC for $\{P\} C \{Q\}$ is $\text{sp}(C, P) \Rightarrow Q$

- Cannot in general compute a \textbf{finite} formula for $\text{sp}(\text{WHILE } S \text{ DO } C, P)$

- The following holds
  \[
  \text{sp}(\text{WHILE } S \text{ DO } C, P) = \text{sp}(\text{WHILE } S \text{ DO } C, \text{sp}(C, (P \land S)))) \lor (P \land \neg S)
  \]

- Above doesn’t define $\text{sp}(C, P)$ to be a finite statement

- As with $\text{wlp}$, can use a hybrid VC and sp method
Summary

- Annotate then generate VCs is the classical method
  - practical tools: Gypsy (1970s), SPARK (current)
  - weakest preconditions are alternative explanation of VCs
  - wlp needs fewer annotations than VC method described earlier
  - wlp also used for refinement

- VCs and wlp go backwards, sp goes forward
  - sp provides verification method based on symbolic simulation
  - widely used for loop-free code
  - current research potential for forwards full proof of correctness
  - probably need mixture of forwards and backwards methods (Hoare’s view)
Range of methods for proving $\{P\}C\{Q\}$

- Bounded model checking (BMC)
  - unwind loops a finite number of times
  - then symbolically execute
  - check states reached satisfy decidable properties

- Full proof of correctness
  - add invariants to loops
  - generate verification conditions
  - prove verification conditions with a theorem prover

- Research goal: unifying framework for a spectrum of methods

decidable checking proof of correctness