Derived rules for finding proofs

- **Suppose the goal is to prove** $\{Precondition\}\ Command\ \{Postcondition\}$
- **If there were a rule of the form**

$$\frac{\vdash\ H_1,\ \cdots,\ \vdash\ H_n}{\vdash\ \{P\}\ C\ \{Q\}}$$

  then we could instantiate

  $P \mapsto Precondition$, $C \mapsto Command$, $Q \mapsto Postcondition$

  to get instances of $H_1,\ \cdots, H_n$ **as subgoals**

- **Some of the rules are already in this form e.g. the sequencing rule**
- **We will derive rules of this form for all commands**
- **Then we use these derived rules for mechanising Hoare Logic proofs**

Derived Rules

- **We will establish derived rules for all commands**

$$\frac{\cdots}{\vdash\ \{P\}\ \texttt{SKIP}\ \{Q\}}$$

$$\frac{\cdots}{\vdash\ \{P\}\ V\texttt{:=}E\ \{Q\}}$$

$$\frac{\cdots}{\vdash\ \{P\}\ C_1\texttt{;}C_2\ \{Q\}}$$

$$\frac{\cdots}{\vdash\ \{P\}\ \texttt{BEGIN VAR}\ V_1\texttt{;}\ \ldots;\ \texttt{VAR}\ V_n\texttt{;}\ C\ \texttt{END}\ \{Q\}}$$

$$\frac{\cdots}{\vdash\ \{P\}\ \texttt{IF}\ S\ \texttt{THEN}\ C_1\ \texttt{ELSE}\ C_2\ \{Q\}}$$

$$\frac{\cdots}{\vdash\ \{P\}\ \texttt{WHILE}\ S\ \texttt{DO}\ C\ \{Q\}}$$

- **These support 'backwards proof' starting from a goal** $\{P\}\ C\ \{Q\}$

The Derived Skip Rule

Derived Skip Rule

$$\frac{\vdash \; P \Rightarrow Q}{\vdash \; \{P\}\ \texttt{SKIP}\ \{Q\}}$$

- **Justifying proof schema:**

1.	$\vdash \; P \Rightarrow Q$	By assumption.
2.	$\vdash \; \{Q\}\ \texttt{SKIP}\ \{Q\}$	By the skip axiom.
3.	$\vdash \; \{P\}\ \texttt{SKIP}\ \{Q\}$	By precondition strengthening with 1 and 2.

- **This schematic proof shows that any inference with the Derived Skip Rule can be replaced by three inferences using primitive rules**
- **Derived rules are 'proof subroutines'**
  - **the schema metavariables (P, Q above) are the parameters of the subroutine**
  - **the derivation schema is the body of the subroutine**

The Derived Assignment Rule

- **An example proof**

1. $\vdash$ $\{\texttt{R=X} \wedge \texttt{0=0}\}$ $\texttt{Q:=0}$ $\{\texttt{R=X} \wedge \texttt{Q=0}\}$ By the assignment axiom.
2. $\vdash$ $\texttt{R=X} \ \Rightarrow\ \texttt{R=X} \wedge \texttt{0=0}$ By pure logic.
3. $\vdash$ $\{\texttt{R=X}\}$ $\texttt{Q:=0}$ $\{\texttt{R=X} \wedge \texttt{Q=0}\}$ By precondition strengthening.

- **Can generalise this proof to a proof schema:**

1. $\vdash$ $\{Q[E/V]\}\ V\texttt{:=}E\ \{Q\}$ By the assignment axiom.
2. $\vdash$ $P \ \Rightarrow\ Q[E/V]$ By assumption.
3. $\vdash$ $\{P\}\ C\ \{Q\}$ By precondition strengthening.

- **This proof schema justifies:**

Derived Assignment Rule

$$\frac{\vdash\ P \Rightarrow Q[E/V]}{\vdash\ \{P\}\ V\texttt{:=}E\ \{Q\}}$$

- **Note:** $Q[E/V]$ **is the weakest liberal precondition** $wlp(V\texttt{:=}E, Q)$

- **Example proof above can now be done in one less step**

1. $\vdash$ $\texttt{R=X} \ \Rightarrow\ \texttt{R=X} \wedge \texttt{0=0}$ By pure logic.
2. $\vdash$ $\{\texttt{R=X}\}$ $\texttt{Q:=0}$ $\{\texttt{R=X} \wedge \texttt{Q=0}\}$ By derived assignment.

Derived Sequenced Assignment Rule

- **The following rule will be useful later**

Derived Sequenced Assignment Rule

$$\frac{\vdash\ \{P\}\ C\ \{Q[E/V]\}}{\vdash\ \{P\}\ C;V:=E\ \{Q\}}$$

- **Exercise: give a proof schema to justify this**
- **Intuitively work backwards:**
  - **push Q 'through' $V:=E$, changing it to $Q[E/V]$**
- **Example: By the assignment axiom:**

  $\vdash$ {X=x∧Y=y} R:=X {R=x∧Y=y}

- **Hence by the sequenced assignment rule**

  $\vdash$ {X=x∧Y=y} R:=X; X:=Y {R=x∧X=y}

Backward Hoare & forward Floyd assignment axioms

- **Recall Hoare (backward) and Floyd (forward) assignment axioms**

  Hoare axiom: $\vdash \{P[E/V]\}\ V := E\ \{P\}$

  Floyd axiom: $\vdash \{P\}\ V := E\ \{\exists v.\ V = E[v/V]\ \wedge\ P[v/V]\}$

- **Exercise 1 (easy): derive forward axiom from Hoare axiom**
  - **hint:** $P \Rightarrow \exists v.\ E = E[v/V]\ \wedge\ P[v/V]$
- **Exercise 2 (a bit harder): derive Hoare axiom from forward axiom**
  - **hint: if v is a new variable then** $P[E/V][v/V] = P[E[v/V]/V]$
- **Exercise 3: devise and justify a derived assignment rule based on the Floyd assignment axiom**

The Derived While Rule

Derived While Rule

$$\frac{\vdash\ P \Rightarrow R \qquad \vdash\ \{R\ \wedge\ S\}\ \mathtt{C}\ \{R\} \qquad \vdash\ R \wedge \neg S\ \Rightarrow Q}{\vdash\ \{P\}\ \mathtt{WHILE\ S\ DO\ C}\ \{Q\}}$$

- **This follows from the While Rule and the rules of consequence**
- **Example: it is easy to show**

$\vdash\ \mathtt{R{=}X}\ \wedge\ \mathtt{Q{=}0}\ \Rightarrow\ \mathtt{X{=}R{+}(Y{\times}Q)}$

$\vdash\ \{\mathtt{X{=}R{+}(Y{\times}Q)}\wedge\mathtt{Y{\leq}R}\}\ \mathtt{R{:=}R{-}Y;\ Q{:=}Q{+}1}\ \{\mathtt{X{=}R{+}(Y{\times}Q)}\}$

$\vdash\ \mathtt{X{=}R{+}(Y{\times}Q)}\wedge\neg(\mathtt{Y{\leq}R})\ \Rightarrow\ \mathtt{X{=}R{+}(Y{\times}Q)}\wedge\neg(\mathtt{Y{\leq}R})$

- **Then, by the derived While rule**

```
⊢ {R=X ∧ Q=0}
    WHILE Y≤R DO
      R:=R-Y; Q:=Q+1
  {X=R+(Y×Q) ∧ ¬(Y≤R)}
```

The Derived Sequencing Rule

- **The rule below follows from the sequencing and consequence rules**

The Derived Sequencing Rule

$$\frac{\begin{array}{ll} & \vdash\ P \Rightarrow P_1 \\ \vdash\ \{P_1\}\ C_1\ \{Q_1\} & \vdash\ Q_1 \Rightarrow P_2 \\ \vdash\ \{P_2\}\ C_2\ \{Q_2\} & \vdash\ Q_2 \Rightarrow P_3 \\ . & . \\ . & . \\ . & . \\ \vdash\ \{P_n\}\ C_n\ \{Q_n\} & \vdash\ Q_n \Rightarrow Q \end{array}}{\vdash\ \{P\}\ C_1;\ \ldots\ ;\ C_n\ \{Q\}}$$

- **Exercise: why no derived conditional rule?**

Example

- **By the assignment axiom**

  (i) $\vdash$ {X=x∧Y=y} R:=X {R=x∧Y=y}
  (ii) $\vdash$ {R=x∧Y=y} X:=Y {R=x∧X=y}
  (iii) $\vdash$ {R=x∧X=y} Y:=R {Y=x∧X=y}

- **Using the derived sequencing rule, it can be deduced** *in one step* **from (i), (ii), (iii) and the fact that for any P: $\vdash\ P \Rightarrow P$**

  $\vdash$ {X=x ∧ Y=y} R:=X; X:=Y; Y:=R {Y=x ∧ X=y}

Derived Block Rule

- **From the derived sequencing rule and the block rule the following rule for blocks can be derived**

The Derived Block Rule

$$
\frac{\begin{array}{ll} & \vdash\ P \Rightarrow P_1 \\ \vdash\ \{P_1\}\ C_1\ \{Q_1\} & \vdash\ Q_1 \Rightarrow P_2 \\ \vdash\ \{P_2\}\ C_2\ \{Q_2\} & \vdash\ Q_2 \Rightarrow P_3 \\ \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \\ \vdash\ \{P_n\}\ C_n\ \{Q_n\} & \vdash\ Q_n \Rightarrow Q \end{array}}{\vdash\ \{P\}\ \texttt{BEGIN VAR}\ V_1;\ \ldots\ \texttt{VAR}\ V_m; C_1;\ \ldots\ ;\ C_n\ \{Q\}}
$$

where none of the variables $V_1, \ldots, V_m$ occur in P or Q.

Example

- **By the assignment axiom**

  (i) $\vdash$ {X=x∧Y=y} R:=X {R=x∧Y=y}
  (ii) $\vdash$ {R=x∧Y=y} X:=Y {R=x∧X=y}
  (iii) $\vdash$ {R=x∧X=y} Y:=R {Y=x∧X=y}

- **Using the derived block rule, it can be deduced** *in one step* **from (i), (ii) and (iii) that**

```
⊢  {X=x ∧ Y=y}
    BEGIN VAR R; R:=X; X:=Y; Y:=R END
   {Y=x ∧ X=y}
```

Deriving Rules for New Commands

- **Suppose we define a one-armed conditional by:**
  `IF` S `THEN` C $\quad\equiv\quad$ `IF` S `THEN` C `ELSE SKIP`
- **We can derive the following rule:**

The One-Armed Conditional Rule

$$\frac{\vdash\ \{P \wedge S\}\ C\ \{Q\} \qquad\qquad \vdash\ P \wedge \neg S \Rightarrow Q}{\vdash\ \{P\}\ \texttt{IF}\ S\ \texttt{THEN}\ C\ \{Q\}}$$

- **Derivation**

1.	$\vdash\ P \wedge \neg S\ \Rightarrow\ Q$	Hypothesis
2.	$\vdash\ \{P \wedge \neg S\}\ \texttt{SKIP}\ \{Q\}$	By derived `SKIP` rule
3.	$\vdash\ \{P \wedge S\}\ C\ \{Q\}$	Hypothesis
4.	$\vdash\ \{P\}\ \texttt{IF}\ S\ \texttt{THEN}\ C\ \texttt{ELSE SKIP}\ \{Q\}$	By conditional rule
5.	$\vdash\ \{P\}\ \texttt{IF}\ S\ \texttt{THEN}\ C\ \{Q\}$	By definition of one-armed conditional

Example

- **From**

  (i) $\vdash$ `{T ∧ X ≥ Y} Y:=X {Y=max(X,Y)}`
  (ii) $\vdash$ `T ∧ Y≥X ⇒ max(X,Y)=Y`

- **Then by the derived one-armed conditional rule it follows that**

  $\vdash$ `{T} IF X≥Y THEN Y:=X {Y=max(X,Y)}`

Aside: linear versus tree structured proofs

- **Tree structured proofs are fine (e.g. for exams)**
  - **advantages: familiar, maybe easier to construct**
  - **disadvantage: harder to present large proofs**

- **Examples (proof schemas, not concrete proofs)**

$$\dfrac{\vdash\ P \Rightarrow Q[E/V] \qquad \dfrac{}{\vdash\ \{Q[E/V]\}\ V := E\ \{Q\}}\,ASS}{\vdash\ \{P\}\ V := E\ \{Q\}}\,PRE$$

$$\dfrac{\vdash\ P \Rightarrow Q \qquad \dfrac{}{\vdash\ \{Q\}\ \texttt{SKIP}\ \{Q\}}\,SKP}{\vdash\ \{P\}\ \texttt{SKIP}\ \{Q\}}\,PRE$$

$$\dfrac{\vdash\ \{P\}C\{Q[E/V]\} \qquad \dfrac{}{\vdash\ \{Q[E/V]\}\ V := E\ \{Q\}}\,ASS}{\vdash\ \{P\}\ C;V := E\ \{Q\}}\,SEQ$$

$$\dfrac{\dfrac{\vdash\ \{P \wedge S\}\ C\ \{Q\} \qquad \dfrac{\vdash\ P \wedge \neg S\ \Rightarrow\ Q}{\vdash\ \{P \wedge \neg S\}\ \texttt{SKIP}\ \{Q\}}\,DSKP}{\vdash\ \{P\}\ \texttt{IF}\ S\ \texttt{THEN}\ C\ \texttt{ELSE}\ \texttt{SKIP}\ \{Q\}}\,IF}{\vdash\ \{P\}\ \texttt{IF}\ S\ \texttt{THEN}\ C\ \{Q\}}\,DEF$$

Forwards and backwards proof

- **Previously it was shown how to prove $\{P\}C\{Q\}$ by**
  - **proving properties of the components of C**
  - **and then putting these together, with the appropriate proof rule, to get the desired property of C**
- **For example, to prove $\vdash \{P\}C_1;C_2\{Q\}$**
- **First prove $\vdash \{P\}C_1\{R\}$ and $\vdash \{R\}C_2\{Q\}$**
- **then deduce $\vdash \{P\}C_1;C_2\{Q\}$ by sequencing rule**
- **This method is called** *forward proof*
  - **move forward from axioms via rules to conclusion**
- **The problem with forwards proof is that it is not always easy to see what you need to prove to get where you want to be**
- **It is more natural to work backwards**
  - **starting from the goal of showing $\{P\}C\{Q\}$**
  - **generate subgoals until problem solved**

Example

- **Suppose one wants to show**

  {X=x ∧ Y=y} R:=X; X:=Y; Y:=R {Y=x ∧ X=y}

- **By the assignment axiom and derived sequenced assignment rule it is sufficient to show the subgoal**

  {X=x ∧ Y=y} R:=X; X:=Y {R=x ∧ X=y}

- **Similarly this subgoal can be reduced to**

  {X=x ∧ Y=y} R:=X {R=x ∧ Y=y}

- **This clearly follows from the assignment axiom**

Backwards versus Forwards Proof

- **Backwards proof just involves using the rules backwards**
- **Given the rule**

$$\frac{\vdash\ S_1 \qquad \ldots \qquad \vdash\ S_n}{\vdash\ S}$$

- **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**

Example Backwards Proof

- **To prove**

```
⊢ {T}
    R:=X;
    Q:=0;
    WHILE Y≤R DO
      BEGIN R:=R-Y; Q:=Q+1 END
  {X=R+(Y×Q) ∧ R<Y}
```

- **By the sequencing rule, it is sufficient to prove**

(i) `⊢ {T} R:=X; Q:=0 {R=X ∧ Q=0}`

(ii)
```
⊢ {R=X ∧ Q=0}
    WHILE Y≤R DO
      BEGIN R:=R-Y; Q:=Q+1 END
  {X=R+(Y×Q) ∧ R<Y}
```

- **Where does** `{R=X ∧ Q=0}` **come from? (Answer later)**

Example Continued (1)

- **From previous slide:**

  (i) $\vdash$ {T} `R:=X; Q:=0` {R=X $\wedge$ Q=0}

- **To prove (i), by the sequenced assignment axiom, we must prove:**

  (iii) $\vdash$ {T} `R:=X` {R=X $\wedge$ 0=0}

- **To prove (iii), by the derived assignment rule, we must prove:**

  $\vdash$ T $\Rightarrow$ X=X $\wedge$ 0=0

- **This is true by pure logic**

Example continued (2)

- **From an earlier slide:**

  (ii) $\vdash$ {R=X $\wedge$ Q=0}
  ```
  WHILE Y≤R DO
    BEGIN R:=R-Y; Q:=Q+1 END
  ```
  {X=R+(Y×Q) $\wedge$ R<Y}

- **To prove (ii), by the derived while rule, we must prove:**

  (iv) R=X $\wedge$ Q=0 $\Rightarrow$ (X = R+(Y×Q))

  (v) X = R+Y×Q $\wedge$ $\neg$(Y≤R) $\Rightarrow$ (X = R+(Y×Q) $\wedge$ R<Y)

  and

  (vi) {X = R+(Y×Q) $\wedge$ (Y≤R)}
  ```
  BEGIN R:=R-Y; Q:=Q+1 END
  ```
  {X=R+(Y×Q)}

- **(iv) and (v) are proved by pure arithmetic**

Example Continued (3)

- To prove (vi), by the block rule, we must prove

$$\text{(vii)}\quad \begin{array}{l}\{\mathtt{X\ =\ R+(Y}\times\mathtt{Q)}\ \wedge\ \mathtt{(Y}\leq\mathtt{R)}\}\\ \ \mathtt{R:=R-Y;\ Q:=Q+1}\\ \{\mathtt{X=R+(Y}\times\mathtt{Q)}\}\end{array}$$

- To prove (vii), by the sequenced assignment rule, we must prove

$$\text{(viii)}\quad \begin{array}{l}\{\mathtt{X=R+(Y}\times\mathtt{Q)}\ \wedge\ \mathtt{(Y}\leq\mathtt{R)}\}\\ \ \mathtt{R:=R-Y}\\ \{\mathtt{X=R+(Y}\times\mathtt{(Q+1))}\}\end{array}$$

- To prove (viii), by the derived assignment rule, we must prove

$$\text{(ix)}\ \mathtt{X=R+(Y}\times\mathtt{Q)}\ \wedge\ \mathtt{Y}\leq\mathtt{R}\ \Rightarrow\ \mathtt{(X\ =\ (R-Y)+(Y}\times\mathtt{(Q+1)))}$$

- This is true by arithmetic
- Exercise: Construct the forwards proof that corresponds
  to this backwards proof

Annotations

- **The sequencing rule introduces a new statement** R

$$\frac{\vdash\ \{P\}\ C_1\ \{R\} \qquad \vdash\ \{R\}\ C_2\ \{Q\}}{\vdash\ \{P\}\ C_1 ; C_2\ \{Q\}}$$

- **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}
 BEGIN
   R:=X;
   Q:=0; {R=X ∧ Q=0} ←—P₁
   WHILE Y≤R DO {X = R+Y×Q} ←—P₂
     BEGIN R:=R-Y; Q:=Q+1 END
 END
{X = R+Y×Q ∧ R<Y}
```

Summary

- **We have looked at three ways of organizing proofs that make it easier for humans to apply them:**
  - **deriving “bigger step” rules**
  - **backwards proof**
  - **annotating programs**
- **Next we see how these techniques can be used to mechanize program verification**

NEW TOPIC: Mechanizing Program Verification

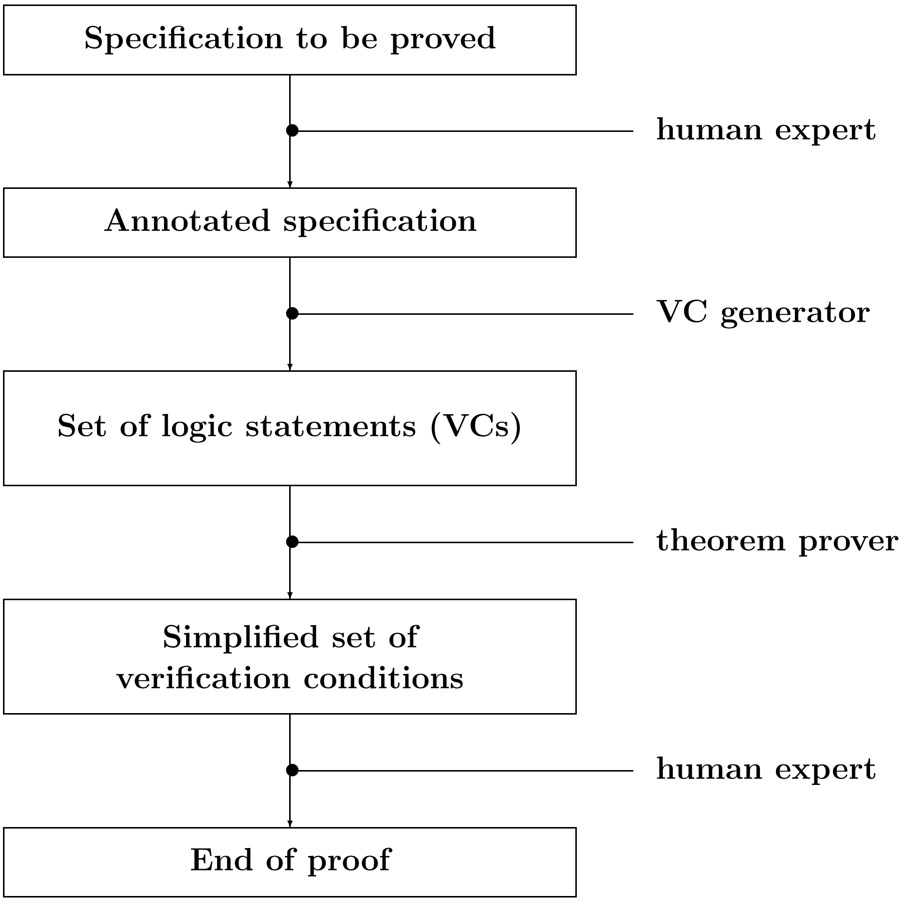
- **The architecture of a simple program verifier will be described**
- **Justified with respect to the rules of Floyd-Hoare logic**
- **It is clear that**
  - **proofs are long and boring, even if the program being verified is quite simple**
  - **lots of fiddly little details to get right, many of which are trivial, e.g.**

$$\vdash\ (\mathtt{R}{=}\mathtt{X}\ \wedge\ \mathtt{Q}{=}0)\ \Rightarrow\ (\mathtt{X} = \mathtt{R} + \mathtt{Y}\times\mathtt{Q})$$

Mechanization

- **Goal:** automate the routine bits of proofs in Floyd-Hoare logic
- Unfortunately, logicians have shown that it is impossible in principle to design a *decision procedure* to decide automatically the truth or falsehood of an arbitrary mathematical statement
- This does not mean that one cannot have procedures that will prove many *useful* theorems
  - the non-existence of a general decision procedure merely shows that one cannot hope to prove *everything* automatically
  - in practice, it is quite possible to build a system that will mechanize the boring and routine aspects of verification
- The standard approach to this will be described in the course
  - ideas very old (JC King's 1969 CMU PhD, Stanford verifier in 1970s)
  - used by program verifiers (e.g. Gypsy and SPARK verifier)
  - used to validate proof carrying code (PCC)
  - provides a verification front end to different provers (see *Why* system)

Architecture of a Verifier



Commentary

- **Input: a Hoare triple annotated with mathematical statements**
  - **these annotations describe relationships between variables**
- **The system generates a set of purely mathematical statements called** *verification conditions* **(or VCs)**
- **If the verification conditions are provable, then the original specification can be deduced from the axioms and rules of Hoare logic**
- **The verification conditions are passed to a** *theorem prover* **program which attempts to prove them automatically**
  - **if it fails, advice is sought from the user**

Verification conditions

- **The three steps in proving $\{P\}C\{Q\}$ with a verifier**
- **$\boxed{1}$ The program C is *annotated* by inserting statements (*assertions*) expressing conditions that are meant to hold at intermediate points**
  - **this step is tricky and needs intelligence and a good understanding of how the program works**
  - **automating it is an artificial intelligence problem**
- **$\boxed{2}$ 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**
- **$\boxed{3}$ 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**

Validity of Verification Conditions

- **It will be shown that**
  - **if one can prove all the verification conditions generated from** $\{P\}C\{Q\}$
  - **then** $\vdash \{P\}C\{Q\}$
- **Step** $\boxed{2}$ **converts a verification problem into a conventional mathematical problem**
- **The process will be illustrated with:**

```
{T}
 BEGIN
   R:=X;
   Q:=0;
   WHILE Y≤R DO
     BEGIN R:=R-Y; Q:=Q+1 END
 END
{X = R+Y×Q ∧ R<Y}
```

Example

- **Step $\boxed{1}$ is to insert annotations $\mathtt{P}_1$ and $\mathtt{P}_2$**

```
{T}
 BEGIN
   R:=X;
   Q:=0; {R=X ∧ Q=0} ←—P1
   WHILE Y≤R DO {X = R+Y×Q} ←—P2
     BEGIN R:=R-Y; Q:=Q+1 END
 END
{X = R+Y×Q ∧ R<Y}
```

- **The annotations $\mathtt{P}_1$ and $\mathtt{P}_2$ state conditions which are intended to hold *whenever* control reaches them**

Example Continued

```
{T}
 BEGIN
   R:=X;
   Q:=0; {R=X ∧ Q=0} ←—P₁
   WHILE Y≤R DO {X = R+Y×Q} ←—P₂
     BEGIN R:=R-Y; Q:=Q+1 END
 END
{X = R+Y×Q ∧ R<Y}
```

- **Control only reaches the point at which $\mathtt{P}_1$ is placed once**
- **It reaches $\mathtt{P}_2$ each time the `WHILE` body is executed**
  - **whenever this happens `X=R+Y×Q` holds, even though the values of `R` and `Q` vary**
  - **$\mathtt{P}_2$ is an *invariant* of the `WHILE`-command**

Generating and Proving Verification Conditions

- **Step $\boxed{2}$ will generate the following four verification conditions**

  (i) `T` $\Rightarrow$ `(X=X` $\wedge$ `0=0)`

  (ii) `(R=X` $\wedge$ `Q=0)` $\Rightarrow$ `(X = R+(Y`$\times$`Q))`

  (iii) `(X = R+(Y`$\times$`Q))` $\wedge$ `Y`$\leq$`R)` $\Rightarrow$ `(X = (R-Y)+(Y`$\times$`(Q+1)))`

  (iv) `(X = R+(Y`$\times$`Q))` $\wedge$ $\neg$`(Y`$\leq$`R)` $\Rightarrow$ `(X = R+(Y`$\times$`Q)` $\wedge$ `R<Y)`

- **Notice that these are statements of arithmetic**
  - **the constructs of our programming language have been 'compiled away'**

- **Step $\boxed{3}$ consists in proving the four verification conditions**
  - **easy with modern automatic theorem provers**

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 `DO` in `WHILE` (and `FOR`) 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)**

Annotation of Specifications

- **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**

```
{X=n}
 BEGIN
   Y:=1; ⟵①
   WHILE X≠0 DO ⟵②
     BEGIN Y:=Y×X; X:=X-1 END
 END
{X=0 ∧ Y=n!}
```

- **Suitable statements would be**

at ①: `{Y = 1 ∧ X = n}`
at ②: `{Y×X! = n!}`

Verification Condition Generation

- **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, ... , vc_n
  - **together with the VCs for $\mathtt{C}_1$ and those for $\mathtt{C}_2$**
- **Each VC rule corresponds to either a primitive or derived rule**

A VC Generation Program

- **The algorithm for generating verification conditions is** *recursive* **on the structure of commands**
- **The rule just given corresponds to the recursive program clause:**

  VC $(C(C_1, C_2))$ = `[`vc_1, ... ,vc_n`]@` (**VC** C_1) `@` (**VC** C_2)

- **The rules are chosen so that only one VC rule applies in each case**
  - **applying them is then purely mechanical**
  - **the choice is based on the syntax**
  - **only one rule applies in each case so VC generation is deterministic**

Justification of VCs

- **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**

Justification of Verification Conditions

- **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: skip and 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**

VC for SKIP

The Skip command

The single verification condition generated by

$$\{P\}\ \texttt{SKIP}\ \{Q\}$$

is

$$P \ \Rightarrow\ Q$$

- Example: The verification condition for

  $\{\texttt{X=0}\}\ \texttt{SKIP}\ \{\texttt{X=0}\}$

  is

  $\texttt{X=0} \ \Rightarrow\ \texttt{X=0}$

  (which is clearly true)

Justification of `SKIP` VC

- **We must show that if the VCs of $\{P\}$ `SKIP` $\{Q\}$ are provable then $\vdash \{P\}$ `SKIP` $\{Q\}$**

- **Proof:**
  - **Assume $P \Rightarrow Q$ is provable since it is the VC (i.e. assume $\vdash P \Rightarrow Q$)**
  - **From derived Skip rule it follows that $\vdash \{P\}$ `SKIP` $\{Q\}$ as required**

VC for Assignments

Assignment commands

The single verification condition generated by

$$\{P\}\ V\texttt{:=}E\ \{Q\}$$

is

$$P \ \Rightarrow\ Q\texttt{[}E/V\texttt{]}$$

- **Example: The verification condition for**

  $\{\texttt{X=0}\}\ \texttt{X:=X+1}\ \{\texttt{X=1}\}$

  is

  $\texttt{X=0} \ \Rightarrow\ \texttt{(X+1)=1}$

  (which is clearly true)

- **Note:** $Q\texttt{[}E/V\texttt{]} \ = \ \texttt{wlp}(\texttt{"}V\texttt{:=}E\texttt{"}, Q)$

Justification of Assignment VC

- **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** $\vdash\ \{P\}\ V := E\ \{Q\}$

VCs for Two-Armed Conditional

Two-armed conditional

The verification conditions generated from

$$\{P\}\ \texttt{IF}\ S\ \texttt{THEN}\ C_1\ \texttt{ELSE}\ C_2\ \{Q\}$$

are

(i) the verification conditions generated by

$$\{P\ \wedge\ S\}\ C_1\ \{Q\}$$

(ii) the verifications generated by

$$\{P\ \wedge\ \neg S\}\ C_2\ \{Q\}$$

- **Example: The verification conditions for**

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

  are

  (i) the VCs for $\{\texttt{T}\ \wedge\ \texttt{X}\geq\texttt{Y}\}$ `MAX:=X` $\{\texttt{MAX=max(X,Y)}\}$

  (ii) the VCs for $\{\texttt{T}\ \wedge\ \neg(\texttt{X}\geq\texttt{Y})\}$ `MAX:=Y` $\{\texttt{MAX=max(X,Y)}\}$

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** $\{P\ \wedge\ S\}\ C_1\ \{Q\}$ **and** $\{P\ \wedge\ \neg S\}\ C_2\ \{Q\}$
  - **The inductive hypotheses tell us that if these VCs are provable then the corresponding Hoare Logic theorems are provable**
  - **i.e. by induction** $\vdash\ \{P\ \wedge\ S\}\ C_1\ \{Q\}$ **and** $\vdash\ \{P\ \wedge\ \neg S\}\ C_2\ \{Q\}$
  - **Hence by the conditional rule** $\vdash\ \{P\}$ `IF` S `THEN` C_1 `ELSE` C_2 $\{Q\}$

Review of Annotated Sequences

- **If $C_1\,;C_2$ is properly annotated, then either**

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

  $\boxed{\text{Case 2:}}$ it is of the form $C\,;V := E$

- **And C, C_1 and C_2 are properly annotated**

VCs for Sequences

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**

$$\{P\}\ C\texttt{;}V\texttt{:=}E\ \{Q\}$$

are the verification conditions generated by $\{P\}\ C\ \{Q\texttt{[}E/V\texttt{]}\}$

Example

- **The verification conditions generated from**

  $\{$`X=x` $\wedge$ `Y=y`$\}$ `R:=X; X:=Y; Y:=R` $\{$`X=y` $\wedge$ `Y=x`$\}$

- **Are those generated by**

  $\{$`X=x` $\wedge$ `Y=y`$\}$ `R:=X; X:=Y` $\{$`(X=y` $\wedge$ `Y=x)[R/Y]`$\}$

- **This simplifies to**

  $\{$`X=x` $\wedge$ `Y=y`$\}$ `R:=X; X:=Y` $\{$`X=y` $\wedge$ `R=x`$\}$

- **The verification conditions generated by this are those generated by**

  $\{$`X=x` $\wedge$ `Y=y`$\}$ `R:=X` $\{$`(X=y` $\wedge$ `R=x)[Y/X]`$\}$

- **Which simplifies to**

  $\{$`X=x` $\wedge$ `Y=y`$\}$ `R:=X` $\{$`Y=y` $\wedge$ `R=x`$\}$

Example Continued

- **The only verification condition generated by**

  $\{$`X=x` $\wedge$ `Y=y`$\}$ `R:=X` $\{$`Y=y` $\wedge$ `R=x`$\}$

  is

  `X=x` $\wedge$ `Y=y` $\Rightarrow$ `(Y=y` $\wedge$ `R=x)[X/R]`

- **Which simplifies to**

  `X=x` $\wedge$ `Y=y` $\Rightarrow$ `Y=y` $\wedge$ `X=x`

- **Thus the single verification condition from**

  $\{$`X=x` $\wedge$ `Y=y`$\}$ `R:=X; X:=Y; Y:=R` $\{$`X=y` $\wedge$ `Y=x`$\}$

  is

  `X=x` $\wedge$ `Y=y` $\Rightarrow$ `Y=y` $\wedge$ `X=x`

Justification of VCs for Sequences (1)

- $\boxed{\text{Case 1:}}$ **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**

  $\vdash\ \{P\}\ C_1\ \{R\}$ **and** $\vdash\ \{R\}\ C_2\ \{Q\}$

- **Hence by the sequencing rule**

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

Justification of VCs for Sequences (2)

- $\boxed{\text{Case 2:}}$ **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**

  $\vdash\ \{P\}\ C\,;V := E\ \{Q\}$

VCs for Blocks

Blocks

The verification conditions generated by

$$\{P\}\ \texttt{BEGIN VAR}\ V_1;\ \ldots\ ;\texttt{VAR}\ V_n;C\ \texttt{END}\ \{Q\}$$

are

(i) the verification conditions generated by $\{P\}C\{Q\}$, and

(ii) none of V_1, $\ldots$, V_n occur in either P or Q.

Syntactic Conditions for Blocks

- **Generating verification conditions from blocks involves checking a syntactic condition**
  - **the local variables do not occur in the precondition or postcondition**
- **The need for this is clear from the side condition in the block rule**
- **Example: the verification conditions for**

```
{X=x ∧ Y=y}
BEGIN VAR R; R:=X; X:=Y; Y:=R END
{X=y ∧ Y=x}
```

- **Are the verification conditions generated by**

  `{X=x ∧ Y=y} R:=X; X:=Y; Y:=R {X=y ∧ Y=x}`

  and the syntactic check that `R` **not in** `{X=x ∧ Y=y}` **or** `{X=y ∧ Y=x}`

- **See previous example for verification conditions generated by this**

Justification of VCs for Blocks

- **If the verification conditions for**

  $\{P\}$ `BEGIN VAR` V_1`;` $\ldots$ `;VAR` V_n`;`C `END` $\{Q\}$

  are provable

- **Then the verification conditions for**

  $\{P\}\ C\ \{Q\}$

  are provable and V_1**,** $\ldots$**,** V_n **do not occur in** P **or** Q

- **By induction**

  $\vdash \{P\}\ C\ \{Q\}$

- **Hence by the block rule**

  $\vdash$ $\{P\}$ `BEGIN VAR` V_1`;` $\ldots$ `;VAR` V_n`;`C `END` $\{Q\}$

VCs for `WHILE`-Commands

- **A correctly annotated specification of a `WHILE`-command has the form**

  $\{P\}$ `WHILE` S `DO` $\{R\}$ C $\{Q\}$

- **The annotation R is called an invariant**

`WHILE`-commands

The verification conditions generated from

$$\{P\}\ \texttt{WHILE}\ S\ \texttt{DO}\ \{R\}\ C\ \{Q\}$$

are

(i) $P \ \Rightarrow\ R$

(ii) $R\ \wedge\ \neg S\ \Rightarrow\ Q$

(iii) the verification conditions generated by $\{R\ \wedge\ S\}\ C\{R\}$

Example

- **The verification conditions for**

  $\{$`R=X` $\wedge$ `Q=0`$\}$
   `WHILE Y`$\leq$`R DO` $\{$`X=R+Y`$\times$`Q`$\}$
     `BEGIN R:=R-Y; Q:=Q+1 END`
  $\{$`X = R+(Y`$\times$`Q)` $\wedge$ `R<Y`$\}$

  are:

  (i) `R=X` $\wedge$ `Q=0` $\Rightarrow$ `(X = R+(Y`$\times$`Q))`

  (ii) `X = R+Y`$\times$`Q` $\wedge$ $\neg$`(Y`$\leq$`R)` $\Rightarrow$ `(X = R+(Y`$\times$`Q)` $\wedge$ `R<Y)`

  together with the verification condition for

  $\{$`X = R+(Y`$\times$`Q)` $\wedge$ `(Y`$\leq$`R)`$\}$
   `BEGIN R:=R-Y; Q:=Q+1 END`
  $\{$`X=R+(Y`$\times$`Q)`$\}$

  which consists of the single condition

  (iii) `X = R+(Y`$\times$`Q)` $\wedge$ `(Y`$\leq$`R)` $\Rightarrow$ `X = (R-Y)+(Y`$\times$`(Q+1))`

Example Summarised

- **By previous transparency**

$$\vdash \{\texttt{R=X} \wedge \texttt{Q=0}\}$$
```
WHILE Y≤R DO
  BEGIN R:=R-Y; Q:=Q+1 END
```
$$\{\texttt{X = R+(Y}\times\texttt{Q)} \wedge \texttt{R<Y}\}$$

if

$$\vdash \texttt{R=X} \wedge \texttt{Q=0} \Rightarrow \texttt{(X = R+(Y}\times\texttt{Q))}$$

and

$$\vdash \texttt{X = R+(Y}\times\texttt{Q)} \wedge \neg\texttt{(Y}\leq\texttt{R)} \Rightarrow \texttt{(X = R+(Y}\times\texttt{Q)} \wedge \texttt{R<Y)}$$

and

$$\vdash \texttt{X = R+(Y}\times\texttt{Q)} \wedge \texttt{(Y}\leq\texttt{R)} \Rightarrow \texttt{X = (R-Y)+(Y}\times\texttt{(Q+1))}$$

Justification of `WHILE` VCs

- **If the verification conditions for**

  $\{P\}$ `WHILE` S `DO` $\{R\}$ C $\{Q\}$

  are provable, then

  $\vdash\ P \Rightarrow R$

  $\vdash\ (R\ \wedge\ \neg S)\ \Rightarrow\ Q$

  and the verification conditions for

  $\{R\ \wedge\ S\}\ C\ \{R\}$

  are provable

- **By induction**

  $\vdash\ \{R\ \wedge\ S\}\ C\ \{R\}$

- **Hence by the derived `WHILE`-rule**

  $\vdash\ \{P\}$ `WHILE` S `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

Dijkstra's weakest preconditions

- **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 Dijksta'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:**
  - $\texttt{wlp}(C, Q)$ = '*The weakest predicate P such that $\{P\}\ C\ \{Q\}$*'
  - $\texttt{wp}(C, Q)$ = '*The weakest predicate P such that $[P]\ C\ [Q]$*'
- **If P and Q are predicates then $Q \Rightarrow P$ means P is 'weaker' than Q**

Rules for weakest preconditions

- **Relation with Hoare specifications:**

$$\begin{array}{lcl} \{P\}\ C\ \{Q\} & \Leftrightarrow & P \Rightarrow \texttt{wlp}(C, Q) \\ [P]\ C\ [Q] & \Leftrightarrow & P \Rightarrow \texttt{wp}(C, Q) \end{array}$$

- **Dijkstra gives rules for computing weakest preconditions:**

$$\begin{array}{lcl} \texttt{wp}(\texttt{SKIP}, Q) & = & Q \\ \texttt{wp}(V \texttt{:=} E, Q) & = & Q[E/V] \\ \texttt{wp}(C_1 \texttt{;} C_2,\ Q) & = & \texttt{wp}(C_1, \texttt{wp}(C_2,\ Q)) \\ \texttt{wp}(\texttt{IF}\ B\ \texttt{THEN}\ C_1\ \texttt{ELSE}\ C_2,\ Q) & = & (B\ \Rightarrow \texttt{wp}(C_1, Q))\ \wedge\ (\neg B\ \Rightarrow\ \texttt{wp}(C_2, Q)) \end{array}$$

  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** $P\ \Rightarrow\ \texttt{wlp}(C, Q)$
  - **VCs use annotations to ensure first order formulas can be generated**

Sequencing example

- **Swapping variables:**

$$
\begin{aligned}
&\texttt{wlp(R:=X; X:=Y; Y:=R}, (\texttt{Y} = x \wedge \texttt{X} = y))\\
&\quad = \texttt{wlp}(\texttt{R:=X},\ \texttt{wlp}(\texttt{X:=Y},\ \texttt{wlp}(\texttt{Y:=R},\ (\texttt{Y} = x \wedge \texttt{X} = y))))\\
&\quad = \texttt{wlp}(\texttt{R:=X},\ \texttt{wlp}(\texttt{X:=Y},\ (\texttt{Y} = x \wedge \texttt{X} = y)\texttt{[R/Y]}))\\
&\quad = \texttt{wlp}(\texttt{R:=X},\ \texttt{wlp}(\texttt{X:=Y},\ (\texttt{R} = x \wedge \texttt{X} = y)))\\
&\quad = \texttt{wlp}(\texttt{R:=X},\ (\texttt{R} = x \wedge \texttt{Y} = y))\\
&\quad = (\texttt{X} = x \wedge \texttt{Y} = y)
\end{aligned}
$$

- **So since** $\{P\}\ C\ \{Q\} \ \Leftrightarrow\ P \ \Rightarrow\ \texttt{wlp}(C,Q)$

  to prove

  $\{\texttt{X} = x \wedge \texttt{Y} = y\}\ \texttt{R:=X; X:=Y; Y:=R}\ \{\texttt{Y} = x \wedge \texttt{X} = y\}$

  just need to prove:

  $(\texttt{X} = x \wedge \texttt{Y} = y) \Rightarrow (\texttt{X} = x \wedge \texttt{Y} = y)$

  which is clearly true (instance of $S \Rightarrow S$**)**

Conditional example

- **Compute `wlp` of the maximum program:**

$$\begin{array}{l}\texttt{wlp}(\texttt{IF X<Y THEN MAX:=Y ELSE MAX:=X}, (\texttt{MAX} = max(x,y)) \\ \quad = \ (\texttt{X<Y} \ \Rightarrow \ \texttt{wlp}(\texttt{MAX:=Y}, \ (\texttt{MAX} = max(x,y)))) \\ \qquad \wedge \\ \qquad (\neg(\texttt{X<Y}) \ \Rightarrow \ \texttt{wlp}(\texttt{MAX:=X}, \ (\texttt{MAX} = max(x,y)))) \\ \quad = \ (\texttt{X<Y} \ \Rightarrow \ \texttt{Y} = max(x,y)) \ \wedge \ (\neg(\texttt{X<Y}) \ \Rightarrow \ \texttt{X} = max(x,y)) \\ \quad = \ \mathit{if} \ \texttt{X<Y} \ \mathit{then} \ \texttt{Y} = max(x,y) \ \mathit{else} \ \texttt{X} = max(x,y)\end{array}$$

- **So to prove**

  $\{\texttt{X} = x \wedge \texttt{Y} = y\}$ `IF X<Y THEN MAX:=X ELSE MAX:=Y` $\{\texttt{MAX} = max(x,y)\}$

  just prove:

  $(\texttt{X} = x \wedge \texttt{Y} = y) \Rightarrow (\texttt{X<Y} \ \Rightarrow \ \texttt{Y} = max(x,y)) \ \wedge \ (\neg(\texttt{X<Y}) \ \Rightarrow \ \texttt{X} = max(x,y))$

  which follows from the defining property of max

  $\vdash \ \ \forall x \ y. \ (x \geq y \Rightarrow x = max(x,y)) \ \wedge \ (\neg(x \geq y) \Rightarrow y = max(x,y))$

Using wlp to improve verification condition method

- If C is loop-free then VC for $\{P\}\ C\ \{Q\}$ is $P \Rightarrow \texttt{wlp}(C, Q)$
  - no annotations needed in sequences!
- Cannot in general compute a finite formula for $\texttt{wlp}(\texttt{WHILE}\ S\ \texttt{DO}\ C,\ Q)$
- The following holds

  $\texttt{wlp}(\texttt{WHILE}\ S\ \texttt{DO}\ C,\ Q)\ =\ \mathit{if}\ S\ \mathit{then}\ \texttt{wlp}(C,\ \texttt{wlp}(\texttt{WHILE}\ S\ \texttt{DO}\ C,\ Q))\ \mathit{else}\ Q$
- Above doesn't define $\texttt{wlp}(C, Q)$ as a finite statement
- We will describe a hybrid VC and wlp method

$\texttt{wlp}$-based verification condition method

- **We define $\texttt{awp}(C, Q)$ and $\texttt{wvc}(C, Q)$**
  - $\texttt{awp}(C, Q)$ **is a statement sort of approximating** $\texttt{wlp}(C, Q)$
  - $\texttt{wvc}(C, Q)$ **is a set of verification conditions**
- **If C is loop-free then**
  - $\texttt{awp}(C, Q) \;=\; \texttt{wlp}(C, Q)$
  - $\texttt{wvc}(C, Q) \;=\; \{\}$
- **Denote by $\wedge\mathcal{S}$ the conjunction of all the statements in $\mathcal{S}$**
  - $\bigwedge\{\} = \texttt{T}$
  - $\bigwedge(\mathcal{S}_1 \cup \mathcal{S}_2) = (\bigwedge\mathcal{S}_1) \wedge (\bigwedge\mathcal{S}_2)$
- **It will follow that $\wedge\texttt{wvc}(C, Q) \Rightarrow \{\texttt{awp}(C, Q)\}\ C\ \{Q\}$**
- **Hence to prove $\{P\}C\{Q\}$ it is sufficient to prove all the statements in $\texttt{wvc}(C, Q)$ and $P \Rightarrow \texttt{awp}(C, Q)$**

Definition of awp

- **Assume all** `WHILE`**-commands are annotated:** `WHILE` S `DO` $\{R\}$ C
- **Define** `awp` **recursively by:**

$$
\begin{array}{ll}
\texttt{awp}(\texttt{SKIP},\ Q) & = Q \\
\texttt{awp}(V\ \texttt{:=}\ E,\ Q) & = Q[E/V] \\
\texttt{awp}(C_1\ \texttt{;}\ C_2,\ Q) & = \texttt{awp}(C_1,\ \texttt{awp}(C_2,\ Q)) \\
\texttt{awp}(\texttt{IF}\ S\ \texttt{THEN}\ C_1\ \texttt{ELSE}\ C_2,\ Q) & = (S\ \wedge \texttt{awp}(C_1,\ Q)) \vee (\neg S \wedge \texttt{awp}(C_2,\ Q)) \\
\texttt{awp}(\texttt{WHILE}\ S\ \texttt{DO}\ \{R\}\ C,\ Q) & = R
\end{array}
$$

- **Note:**
$(S \wedge \texttt{awp}(C_1,\ Q)) \vee (\neg S \wedge \texttt{awp}(C_2,\ Q)) \ = \ \mathit{if}\ S\ \mathit{then}\ \texttt{awp}(C_1,\ Q)\ \mathit{else}\ \texttt{awp}(C_2,\ Q)$

Definition of `wvc`

- **Assume all `WHILE`-commands are annotated:** `WHILE` S `DO` $\{R\}$ C
- **Define `wvc` recursively by:**

$$
\begin{array}{ll}
\texttt{wvc}(\texttt{SKIP},\ Q) & = \{\} \\
\texttt{wvc}(V\ \texttt{:=}\ E,\ Q) & = \{\} \\
\texttt{wvc}(C_1\ \texttt{;}\ C_2,\ Q) & = \texttt{wvc}(C_1, \texttt{awp}(C_2, Q)) \cup \texttt{wvc}(C_2, Q) \\
\texttt{wvc}(\texttt{IF}\ S\ \texttt{THEN}\ C_1\ \texttt{ELSE}\ C_2,\ Q) & = \texttt{wvc}(C_1,\ Q) \cup \texttt{wvc}(C_2,\ Q) \\
\texttt{wvc}(\texttt{WHILE}\ S\ \texttt{DO}\ \{R\}\ C,\ Q) & = \{R \wedge \neg S \Rightarrow Q,\ R \wedge S \Rightarrow \texttt{awp}(C, R)\} \\
 & \quad \cup\ \texttt{wvc}(C, R)
\end{array}
$$

Correctness of wlp-based verification conditions

- **Theorem:** $\bigwedge \texttt{wvc}(C, Q) \Rightarrow \{\texttt{awp}(C, Q)\}\ C\ \{Q\}$. **Proof by Induction on** C
  - $\bigwedge\texttt{wvc}(\texttt{SKIP}, Q) \Rightarrow \{\texttt{awp}(C, Q)\}\ C\ \{Q\}$ **is** $\texttt{T} \Rightarrow \{Q\}\ \texttt{SKIP}\ \{Q\}$
  - $\bigwedge\texttt{wvc}(V\texttt{:=}E, Q) \Rightarrow \{\texttt{awp}(C, Q)\}\ C\ \{Q\}$ **is** $\texttt{T} \Rightarrow \{Q[E/V]\}\ V\ \texttt{:=}\ E\ \{Q\}$
  - $\bigwedge\texttt{wvc}(C_1\texttt{;}C_2, Q) \Rightarrow \{\texttt{awp}(C_1\texttt{;}C_2, Q)\}\ C_1\texttt{;}C_2\ \{Q\}$ **is**
    $\bigwedge(\texttt{wvc}(C_1, \texttt{awp}(C_2, Q)) \cup \texttt{wvc}(C_2, Q)) \Rightarrow \{\texttt{awp}(C_1,\ \texttt{awp}(C_2, Q))\}\ C_1\texttt{;}C_2\ \{Q\}$.
    By induction $\bigwedge\texttt{wvc}(C_2, Q) \Rightarrow \{\texttt{awp}(C_2, Q)\}\ C_1\ \{Q\}$
    and $\bigwedge\texttt{wvc}(C_1, \texttt{awp}(C_2, Q)) \Rightarrow \{\texttt{awp}(C_1, \texttt{awp}(C_2, Q))\}\ C_2\ \{\texttt{awp}(C_2, Q)\}$,
    hence result by the Sequencing Rule.
  - $\bigwedge\texttt{wvc}(\texttt{IF}\ S\ \texttt{THEN}\ C_1\ \texttt{ELSE}\ C_2, Q)$
    $\Rightarrow \{\texttt{awp}(\texttt{IF}\ S\ \texttt{THEN}\ C_1\ \texttt{ELSE}\ C_2, Q)\}\ \texttt{IF}\ S\ \texttt{THEN}\ C_1\ \texttt{ELSE}\ C_2\ \{Q\}$
    is $\bigwedge(\texttt{wvc}(C_1,\ Q) \cup \texttt{wvc}(C_2,\ Q))$
    $\Rightarrow \{(S\ \wedge \texttt{awp}(C_1,\ Q)) \vee (\neg S \wedge \texttt{awp}(C_2,\ Q))\}\ \texttt{IF}\ S\ \texttt{THEN}\ C_1\ \texttt{ELSE}\ C_2\ \{Q\}$.
    By induction $\bigwedge\texttt{wvc}(C_1, Q) \Rightarrow \{\texttt{awp}(C_1, Q)\}\ C_1\ \{Q\}$
    and $\bigwedge\texttt{wvc}(C_2, Q) \Rightarrow \{\texttt{awp}(C_2, Q)\}\ C_2\ \{Q\}$. **Strengthening preconditions**
    gives $\bigwedge\texttt{wvc}(C_1, Q) \Rightarrow \{\texttt{awp}(C_1, Q) \wedge S\}\ C_1\ \{Q\}$
    and $\bigwedge\texttt{wvc}(C_2, Q) \Rightarrow \{\texttt{awp}(C_2, Q) \wedge \neg S\}\ C_2\ \{Q\}$, **hence**
    $\bigwedge\texttt{wvc}(C_1, Q) \Rightarrow \{((S\ \wedge \texttt{awp}(C_1,\ Q)) \vee (\neg S \wedge \texttt{awp}(C_2,\ Q))) \wedge S\}\ C_1\ \{Q\}$
    and $\bigwedge\texttt{wvc}(C_2, Q) \Rightarrow \{((S\ \wedge \texttt{awp}(C_1,\ Q)) \vee (\neg S \wedge \texttt{awp}(C_2,\ Q))) \wedge \neg S\}\ C_2\ \{Q\}$,
    hence result by the Conditional Rule.
  - $\bigwedge\texttt{wvc}(\texttt{WHILE}\ S\ \texttt{DO}\ \{R\}\ C, Q) \Rightarrow \{\texttt{awp}(\texttt{WHILE}\ S\ \texttt{DO}\ \{R\}\ C, Q)\}\ \texttt{WHILE}\ S\ \texttt{DO}\ \{R\}\ C\ \{Q\}$
    is $\bigwedge(\{R \wedge \neg S \Rightarrow Q,\ R \wedge S \Rightarrow \texttt{awp}(C, R)\} \cup \texttt{wvc}(C, R)) \Rightarrow \{R\}\ \texttt{WHILE}\ S\ \texttt{DO}\ \{R\}\ C\ \{Q\}$.
    By induction $\bigwedge\texttt{wvc}(C, R) \Rightarrow \{\texttt{awp}(C, R)\}\ C\ \{R\}$, **hence result by** WHILE**-Rule.**

Example

$\boxed{\texttt{wvc}(\texttt{WHILE}\ S\ \texttt{DO}\ \{R\}\ C,\ Q)\ =\ (R,\ \{R \wedge \neg S \Rightarrow Q,\ R \wedge S \Rightarrow \texttt{awp}(C,R)\} \cup \texttt{wvc}(C,R))}$

$\texttt{wvc}(\texttt{R:=R-Y;Q:=Q+1},\ \texttt{X} = \texttt{R} + \texttt{Y} \times \texttt{Q})$ **(analyse body)**

$= (\texttt{wlp}(\texttt{R:=R-Y;Q:=Q+1},\ \texttt{X} = \texttt{R} + \texttt{Y} \times \texttt{Q}), \{\})$

$= (\texttt{X} = \texttt{R-Y} + \texttt{Y} \times \texttt{Q+1},\ \{\})$

$\texttt{wvc}(\texttt{WHILE Y} \leq \texttt{R DO}\ \{\texttt{X} = \texttt{R} + \texttt{Y} \times \texttt{Q}\}\ \texttt{R:=R-Y;Q:=Q+1},\ \texttt{X} = \texttt{R+Y}\times\texttt{Q} \wedge \texttt{R<Y})$

$= (\texttt{X} = \texttt{R} + \texttt{Y} \times \texttt{Q},$

$\{\texttt{X} = \texttt{R} + \texttt{Y} \times \texttt{Q} \wedge \neg(\texttt{Y} \leq \texttt{R}) \Rightarrow \texttt{X} = \texttt{R+Y}\times\texttt{Q} \wedge \texttt{R<Y},$

$\texttt{X} = \texttt{R} + \texttt{Y} \times \texttt{Q} \wedge \texttt{Y} \leq \texttt{R} \Rightarrow \texttt{X} = \texttt{R-Y} + \texttt{Y} \times \texttt{Q+1}\} \cup \{\})$

$\texttt{wvc}(\texttt{Q:=0;WHILE Y} \leq \texttt{R DO}\ \{\texttt{X} = \texttt{R} + \texttt{Y} \times \texttt{Q}\}\ \texttt{R:=R-Y; Q:=Q+1},\ \texttt{X} = \texttt{R+Y}\times\texttt{Q} \wedge \texttt{R<Y})$

$= (\texttt{X} = \texttt{R} + \texttt{Y} \times \texttt{0},$

$\{\} \cup \{\texttt{X} = \texttt{R} + \texttt{Y} \times \texttt{Q} \wedge \neg(\texttt{Y} \leq \texttt{R}) \Rightarrow \texttt{X} = \texttt{R+Y}\times\texttt{Q} \wedge \texttt{R<Y},$

$\texttt{X} = \texttt{R} + \texttt{Y} \times \texttt{Q} \wedge \texttt{Y} \leq \texttt{R} \Rightarrow \texttt{X} = \texttt{R-Y} + \texttt{Y} \times \texttt{Q+1}\})$

$\texttt{wvc}(\texttt{R=X;Q:=0;WHILE Y} \leq \texttt{R DO}\ \{\texttt{X} = \texttt{R} + \texttt{Y} \times \texttt{Q}\}\ \texttt{R:=R-Y; Q:=Q+1},\ \texttt{X} = \texttt{R+Y}\times\texttt{Q} \wedge \texttt{R<Y})$

$= (\texttt{X} = \texttt{X} + \texttt{Y} \times \texttt{0},$

$\{\} \cup \{\texttt{X} = \texttt{R} + \texttt{Y} \times \texttt{Q} \wedge \neg(\texttt{Y} \leq \texttt{R}) \Rightarrow \texttt{X} = \texttt{R+Y}\times\texttt{Q} \wedge \texttt{R<Y},$

$\texttt{X} = \texttt{R} + \texttt{Y} \times \texttt{Q} \wedge \texttt{Y} \leq \texttt{R} \Rightarrow \texttt{X} = \texttt{R-Y} + \texttt{Y} \times \texttt{Q+1}\})$

As $\texttt{X} = \texttt{X} + \texttt{Y} \times \texttt{0}$ is $\texttt{T}$ the correctness theorem for $\texttt{wvc}$ gives

$\texttt{X} = \texttt{R} + \texttt{Y} \times \texttt{Q} \wedge \neg(\texttt{Y} \leq \texttt{R}) \Rightarrow \texttt{X} = \texttt{R+Y}\times\texttt{Q} \wedge \texttt{R<Y}\ \wedge\ \texttt{X} = \texttt{R} + \texttt{Y} \times \texttt{Q} \wedge \texttt{Y} \leq \texttt{R} \Rightarrow \texttt{X} = \texttt{R-Y} + \texttt{Y} \times \texttt{Q+1}$

$\Rightarrow$

$\{\texttt{T}\}\ \texttt{R=X;Q:=0;WHILE Y} \leq \texttt{R DO}\ \{\texttt{X} = \texttt{R} + \texttt{Y} \times \texttt{Q}\}\ \{\texttt{X} = \texttt{R+Y}\times\texttt{Q} \wedge \texttt{R<Y}\}$

Strongest postconditions

- **Define $\texttt{sp}(C, P)$ to be 'strongest' Q such that $\{P\}\ C\ \{Q\}$**
  - **partial correctness: $\{P\}\ C\ \{\texttt{sp}(C, P)\}$**
  - **strongest means if $\{P\}\ C\ \{Q\}$ then $\texttt{sp}(C, P) \Rightarrow Q$**
- **Note that `wlp` goes 'backwards', but `sp` goes 'forwards'**
  - **verification condition for $\{P\}\ C\ \{Q\}$ is: $\texttt{sp}(C, P) \Rightarrow Q$**
- **By 'strongest' and Hoare logic postcondition weakening**
  - $\{P\}\ C\ \{Q\}$ **if and only if** $\texttt{sp}(C, P) \Rightarrow Q$

Strongest postconditions for loop-free code

- **Only consider loop-free code**
- $\texttt{sp}(\texttt{SKIP},\ P)\ =\ P$
- $\texttt{sp}(V\ \texttt{:=}\ E,\ P)\ =\ \exists v.\ V = E\texttt{[}v/V\texttt{]} \wedge P\texttt{[}v/V\texttt{]}$
- $\texttt{sp}(C_1\ \texttt{;}\ C_2,\ P)\ =\ \texttt{sp}(C_2,\ \texttt{sp}(C_1,\ P))$
- $\texttt{sp}(\texttt{IF}\ S\ \texttt{THEN}\ C_1\ \texttt{ELSE}\ C_2,\ P)\ =\ \texttt{sp}(C_1,\ P \wedge S)\ \vee\ \texttt{sp}(C_2,\ P \wedge \neg S)$

- $\texttt{sp}(V\texttt{:=}E,\ P)$ **corresponds to Floyd assignment axiom**
- **Can dynamically prune conditionals because** $\texttt{sp}(C, \texttt{F}) = \texttt{F}$

Sequencing example

- $\mathtt{sp}(\mathtt{R:=X;\ X:=Y;\ Y:=R},\ \mathtt{X} = x \wedge \mathtt{Y} = y)$
$$
\begin{array}{cl}
= & \mathtt{sp}(\mathtt{Y:=R},\ \mathtt{sp}(\mathtt{X:=Y},\ \mathtt{sp}(\mathtt{R:=X},\ \mathtt{X} = x \wedge \mathtt{Y} = y))) \\
= & \mathtt{sp}(\mathtt{Y:=R},\ \mathtt{sp}(\mathtt{X:=Y},\ (\exists v.\ \mathtt{R} = \mathtt{X}[v/\mathtt{R}] \wedge (\mathtt{X} = x \wedge \mathtt{Y} = y)[v/\mathtt{R}]))) \\
= & \mathtt{sp}(\mathtt{Y:=R},\ \mathtt{sp}(\mathtt{X:=Y},\ (\exists v.\ \mathtt{R} = \mathtt{X} \wedge (\mathtt{X} = x \wedge \mathtt{Y} = y)))) \\
= & \mathtt{sp}(\mathtt{Y:=R},\ \mathtt{sp}(\mathtt{X:=Y},\ (\mathtt{R} = \mathtt{X} \wedge \mathtt{X} = x \wedge \mathtt{Y} = y))) \\
= & \mathtt{sp}(\mathtt{Y:=R},\ (\exists v.\ \mathtt{X} = \mathtt{Y}[v/\mathtt{X}] \wedge (\mathtt{R} = \mathtt{X} \wedge \mathtt{X} = x \wedge \mathtt{Y} = y)[v/\mathtt{X}])) \\
= & \mathtt{sp}(\mathtt{Y:=R},\ (\exists v.\ \mathtt{X} = \mathtt{Y} \wedge (\mathtt{R} = v \wedge v = x \wedge \mathtt{Y} = y))) \\
= & \mathtt{sp}(\mathtt{Y:=R},\ (\exists v.\ \mathtt{X} = \mathtt{Y} \wedge (\mathtt{R} = x \wedge v = x \wedge \mathtt{Y} = y))) \\
= & \mathtt{sp}(\mathtt{Y:=R},\ (\mathtt{X} = \mathtt{Y} \wedge (\mathtt{R} = x \wedge (\exists v.\ v = x) \wedge \mathtt{Y} = y))) \\
= & \mathtt{sp}(\mathtt{Y:=R},\ (\mathtt{X} = \mathtt{Y} \wedge (\mathtt{R} = x \wedge \mathtt{T} \wedge \mathtt{Y} = y))) \\
= & \mathtt{sp}(\mathtt{Y:=R},\ (\mathtt{X} = \mathtt{Y} \wedge \mathtt{R} = x \wedge \mathtt{Y} = y)) \\
= & \exists v.\ \mathtt{Y} = \mathtt{R}[v/\mathtt{Y}] \wedge (\mathtt{X} = \mathtt{Y} \wedge \mathtt{R} = x \wedge \mathtt{Y} = y)[v/\mathtt{Y}] \\
= & \exists v.\ \mathtt{Y} = \mathtt{R} \wedge (\mathtt{X} = v \wedge \mathtt{R} = x \wedge v = y) \\
= & \exists v.\ \mathtt{Y} = \mathtt{R} \wedge (\mathtt{X} = y \wedge \mathtt{R} = x \wedge v = y) \\
= & \mathtt{Y} = \mathtt{R} \wedge (\mathtt{X} = y \wedge \mathtt{R} = x \wedge (\exists v.\ v = y)) \\
= & \mathtt{Y} = \mathtt{R} \wedge (\mathtt{X} = y \wedge \mathtt{R} = x \wedge \mathtt{T}) \\
= & \mathtt{Y} = \mathtt{R} \wedge \mathtt{X} = y \wedge \mathtt{R} = x \\
= & \mathtt{Y} = x \wedge \mathtt{X} = y \wedge \mathtt{R} = x
\end{array}
$$

- **So to prove** $\{\mathtt{X} = x \wedge \mathtt{Y} = y\}\ \mathtt{R:=X;\ X:=Y;\ Y:=R}\ \{\mathtt{Y} = x \wedge \mathtt{X} = y\}$
just prove: $(\mathtt{Y} = x \wedge \mathtt{X} = y \wedge \mathtt{R} = x) \Rightarrow \mathtt{Y} = x \wedge \mathtt{X} = y$

Conditional example

- **Compute sp of the maximum program:**

$$
\begin{array}{ll}
\multicolumn{2}{l}{\mathtt{sp}(\mathtt{IF\ X<Y\ THEN\ MAX:=Y\ ELSE\ MAX:=X},\ (\mathtt{X} = x \wedge \mathtt{Y} = y))} \\
= & \mathtt{sp}(\mathtt{MAX:=Y},\ ((\mathtt{X} = x \wedge \mathtt{Y} = y) \wedge \mathtt{X} < \mathtt{Y})) \\
 & \vee \\
 & \mathtt{sp}(\mathtt{MAX:=X},\ ((\mathtt{X} = x \wedge \mathtt{Y} = y) \wedge \neg(\mathtt{X} < \mathtt{Y}))) \\
= & \exists v.\ \mathtt{MAX} = \mathtt{Y}[v/\mathtt{MAX}] \wedge ((\mathtt{X} = x \wedge \mathtt{Y} = y) \wedge \mathtt{X} < \mathtt{Y})[v/\mathtt{MAX}] \\
 & \vee \\
 & \exists v.\ \mathtt{MAX} = \mathtt{X}[v/\mathtt{MAX}] \wedge ((\mathtt{X} = x \wedge \mathtt{Y} = y) \wedge \neg(\mathtt{X} < \mathtt{Y}))[v/\mathtt{MAX}] \\
= & \exists v.\ \mathtt{MAX} = \mathtt{Y} \wedge ((\mathtt{X} = x \wedge \mathtt{Y} = y) \wedge \mathtt{X} < \mathtt{Y}) \\
 & \vee \\
 & \exists v.\ \mathtt{MAX} = \mathtt{X} \wedge \mathtt{X} = x \wedge \mathtt{Y} = y \wedge \neg(\mathtt{X} < \mathtt{Y})) \\
= & (\mathtt{MAX} = \mathtt{Y} \wedge \mathtt{X} = x \wedge \mathtt{Y} = y \wedge \mathtt{X} < \mathtt{Y}) \vee (\mathtt{MAX} = \mathtt{X} \wedge \mathtt{X} = x \wedge \mathtt{Y} = y \wedge \neg(\mathtt{X} < \mathtt{Y}) \\
= & (\mathtt{MAX} = y \wedge \mathtt{X} = x \wedge \mathtt{Y} = y \wedge x < y) \vee (\mathtt{MAX} = x \wedge \mathtt{X} = x \wedge \mathtt{Y} = y \wedge \neg(x < y) \\
= & \mathit{if}\ \ x < y\ \ \mathit{then}\ (\mathtt{MAX} = y \wedge \mathtt{X} = x \wedge \mathtt{Y} = y)\ \ \mathit{else}\ (\mathtt{MAX} = x \wedge \mathtt{X} = x \wedge \mathtt{Y} = y) \\
= & \mathtt{MAX} = (\mathit{if}\ \ x < y\ \ \mathit{then}\ \ y\ \ \mathit{else}\ \ x) \wedge \mathtt{X} = x \wedge \mathtt{Y} = y \\
= & \mathtt{MAX} = \mathit{max}(x, y) \wedge \mathtt{X} = x \wedge \mathtt{Y} = y
\end{array}
$$

Computing sp as symbolic execution: assignment (1)

- **Floyd assignment formula makes computing** sp **messy in general**
- **For a special case it becomes like symbolic execution**
- **Symbolic state:** $\mathtt{X}_1 = E_1 \wedge \cdots \wedge \mathtt{X}_i = E_i \wedge \cdots \wedge \mathtt{X}_n = E_n \wedge R$
- **Suppose** E_1, $\ldots$, E_n **or** R **doesn't contain** $\mathtt{X}_1$, $\ldots$, $\mathtt{X}_n$ **then**

$$
\begin{aligned}
&\mathtt{sp}(\mathtt{X}_i\mathtt{:=}E,\ (\mathtt{X}_1 = E_1 \wedge \cdots \wedge \mathtt{X}_i = E_i \wedge \cdots \wedge \mathtt{X}_n = E_n \wedge R)) \\
&\quad = \exists v.\ \mathtt{X}_i = E\,[v/\mathtt{X}_i] \wedge (\mathtt{X}_1 = E_1 \wedge \cdots \wedge \mathtt{X}_i = E_i \wedge \cdots \wedge \mathtt{X}_n = E_n \wedge R)[v/\mathtt{X}_i] \\
&\quad = \exists v.\ \mathtt{X}_i = E\,[v/\mathtt{X}_i] \wedge (\mathtt{X}_1 = E_1 \wedge \cdots \wedge v = E_i \wedge \cdots \wedge \mathtt{X}_n = E_n \wedge R) \\
&\quad = \mathtt{X}_i = E\,[E_i/\mathtt{X}_i] \wedge (\mathtt{X}_1 = E_1 \wedge \cdots \wedge (\exists v.\ v = E_i) \wedge \cdots \wedge \mathtt{X}_n = E_n \wedge R) \\
&\quad = \mathtt{X}_i = E\,[E_i/\mathtt{X}_i] \wedge (\mathtt{X}_1 = E_1 \wedge \cdots \wedge \mathtt{T} \wedge \cdots \wedge \mathtt{X}_n = E_n \wedge R) \\
&\quad = \mathtt{X}_i = E\,[E_i/\mathtt{X}_i] \wedge \mathtt{X}_1 = E_1 \wedge \cdots \wedge \mathtt{X}_n = E_n \wedge R \\
&\quad = \mathtt{X}_i = E\,[E_1/\mathtt{X}_1]\cdots[E_n/\mathtt{X}_n] \wedge \mathtt{X}_1 = E_1 \wedge \cdots \wedge \mathtt{X}_n = E_n \wedge R \\
&\quad = \mathtt{X}_1 = E_1 \wedge \cdots \wedge \mathtt{X}_i = E\,[E_1/\mathtt{X}_1]\cdots[E_n/\mathtt{X}_n] \wedge \cdots \wedge \mathtt{X}_n = E_n \wedge R
\end{aligned}
$$

- **Note** $E\,[E_1/\mathtt{X}_1]\cdots[E_n/\mathtt{X}_n]$ **doesn't contain** $\mathtt{X}_1$, $\ldots$, $\mathtt{X}_n$

Computing sp as symbolic execution: assignment (2)

- **Suppose $\mathtt{X} \neq \mathtt{X}_i$ and $\mathtt{X}$ doesn't occur in E_i or E for $1 \leq i \leq n$, then**

$$\begin{aligned}
&\mathtt{sp}(\mathtt{X}\texttt{:=}E,\ (\mathtt{X}_1 = E_1 \wedge \cdots \wedge \mathtt{X}_n = E_n \wedge R)) \\
&\quad = \exists v.\ \mathtt{X} = E\,\texttt{[}v/\mathtt{X}\texttt{]} \wedge (\mathtt{X}_1 = E_1 \wedge \cdots \wedge \mathtt{X}_n = E_n \wedge R)\texttt{[}v/\mathtt{X}\texttt{]} \\
&\quad = \exists v.\ \mathtt{X} = E \wedge \mathtt{X}_1 = E_1 \wedge \cdots \wedge \mathtt{X}_n = E_n \wedge R \\
&\quad = \mathtt{X} = E\,\texttt{[}E_1/\mathtt{X}_1\texttt{]}\cdots\texttt{[}E_n/\mathtt{X}_n\texttt{]} \wedge \mathtt{X}_1 = E_1 \wedge \cdots \wedge \mathtt{X}_n = E_n \wedge R
\end{aligned}$$

- **Summarising: we have two symbolic computation rules:**

  1. if E_1, ..., E_n or R doesn't contain $\mathtt{X}_1$, ..., $\mathtt{X}_n$ then:

$$\begin{aligned}
&\mathtt{sp}(\mathtt{X}_i\texttt{:=}E,\ (\mathtt{X}_1 = E_1 \wedge \cdots \wedge \mathtt{X}_i = E_i \wedge \cdots \wedge \mathtt{X}_n = E_n \wedge R)) \\
&\quad = (\mathtt{X}_1 = E_1 \wedge \cdots \wedge \mathtt{X}_i = E\,\texttt{[}E_1/\mathtt{X}_1\texttt{]}\cdots\texttt{[}E_n/\mathtt{X}_n\texttt{]} \wedge \cdots \wedge \mathtt{X}_n = E_n \wedge R)
\end{aligned}$$

  2. if $\mathtt{X} \neq \mathtt{X}_i$ and $\mathtt{X}$ doesn't occur in E_i or E for $1 \leq i \leq n$, then

$$\begin{aligned}
&\mathtt{sp}(\mathtt{X}\texttt{:=}E,\ (\mathtt{X}_1 = E_1 \wedge \cdots \wedge \mathtt{X}_n = E_n \wedge R)) \\
&\quad = (\mathtt{X} = E\,\texttt{[}E_1/\mathtt{X}_1\texttt{]}\cdots\texttt{[}E_n/\mathtt{X}_n\texttt{]} \wedge \mathtt{X}_1 = E_1 \wedge \cdots \wedge \mathtt{X}_n = E_n \wedge R)
\end{aligned}$$

Computing $\texttt{sp}$ as symbolic execution: conditional (1)

- **Suppose if none of** $E_1, \ldots, E_n, R_1, R_2$ **contain** $\texttt{X}_1, \ldots, \texttt{X}_n$ **then**

$$\texttt{sp}(C_1,\ (\texttt{X}_1 = E_1 \wedge \cdots \wedge \texttt{X}_n = E_n \wedge R_1)) \ = \ (\texttt{X}_1 = E_{11} \wedge \cdots \wedge \texttt{X}_n = E_{1n} \wedge R_1)$$

$$\texttt{sp}(C_2,\ (\texttt{X}_1 = E_1 \wedge \cdots \wedge \texttt{X}_n = E_n \wedge R_2)) \ = \ (\texttt{X}_1 = E_{21} \wedge \cdots \wedge \texttt{X}_n = E_{2n} \wedge R_2)$$

- **Conditional notation:** $(B \rightarrow E_1 \mid E_2) \ = \ \mathit{if}\ B\ \mathit{then}\ E_1\ \mathit{else}\ E_2$

- **Then for** $E_1, \ldots, E_n, R$ **not containing** $\texttt{X}_1, \ldots, \texttt{X}_n$

$$\begin{array}{l}
\texttt{sp}(\texttt{IF}\ S\ \texttt{THEN}\ C_1\ \texttt{ELSE}\ C_2,\ (\texttt{X}_1 = E_1 \wedge \cdots \wedge \texttt{X}_n = E_n \wedge R)) \\
\quad = \ \texttt{sp}(C_1,\ (\texttt{X}_1 = E_1 \wedge \cdots \wedge \texttt{X}_n = E_n \wedge R) \wedge S) \\
\qquad \vee \\
\qquad \texttt{sp}(C_2,\ (\texttt{X}_1 = E_1 \wedge \cdots \wedge \texttt{X}_n = E_n \wedge R) \wedge \neg S) \\
\quad = \ \texttt{sp}(C_1,\ (\texttt{X}_1 = E_1 \wedge \cdots \wedge \texttt{X}_n = E_n \wedge (R \wedge S\texttt{[}E_1/\texttt{X}_1\texttt{]}\cdots\texttt{[}E_n/\texttt{X}_n\texttt{]}))) \\
\qquad \vee \\
\qquad \texttt{sp}(C_2,\ (\texttt{X}_1 = E_1 \wedge \cdots \wedge \texttt{X}_n = E_n \wedge (R \wedge \neg S\texttt{[}E_1/\texttt{X}_1\texttt{]}\cdots\texttt{[}E_n/\texttt{X}_n\texttt{]}))) \\
\quad = \ (\texttt{X}_1 = E_{11} \wedge \cdots \wedge \texttt{X}_n = E_{1n} \wedge (R \wedge S\texttt{[}E_1/\texttt{X}_1\texttt{]}\cdots\texttt{[}E_n/\texttt{X}_n\texttt{]})) \qquad \textbf{(by assumption)} \\
\qquad \vee \\
\qquad (\texttt{X}_1 = E_{21} \wedge \cdots \wedge \texttt{X}_n = E_{2n} \wedge (R \wedge \neg S\texttt{[}E_1/\texttt{X}_1\texttt{]}\cdots\texttt{[}E_n/\texttt{X}_n\texttt{]})) \\
\quad = \ (\texttt{X}_1{=}(S\texttt{[}E_1/\texttt{X}_1\texttt{]}\cdots\texttt{[}E_n/\texttt{X}_n\texttt{]}{\rightarrow}E_{11}|E_{21})){\wedge}\cdots{\wedge}(\texttt{X}_n{=}(S\texttt{[}E_1/\texttt{X}_1\texttt{]}\cdots\texttt{[}E_n/\texttt{X}_n\texttt{]}{\rightarrow}E_{1n}|E_{2n})) \wedge R
\end{array}$$

Computing $\mathtt{sp}$ as symbolic execution: conditional (2)

- **From last slide if $E_1, \ldots, E_n$, R do not contain $\mathtt{X}_1, \ldots, \mathtt{X}_n$**

  $\mathtt{sp}(\mathtt{IF}\ S\ \mathtt{THEN}\ C_1\ \mathtt{ELSE}\ C_2,\ (\mathtt{X}_1 = E_1 \wedge \cdots \wedge \mathtt{X}_n = E_n \wedge R))$

  $= (\mathtt{X}_1{=}(S\,[E_1/\mathtt{X}_1]\cdots[E_n/\mathtt{X}_n]{\rightarrow}E_{11}|E_{21}))\wedge\cdots\wedge(\mathtt{X}_n{=}(S\,[E_1/\mathtt{X}_1]\cdots[E_n/\mathtt{X}_n]{\rightarrow}E_{1n}|E_{2n})) \wedge R$

  where

  $\mathtt{sp}(C_1,\ (\mathtt{X}_1 = E_1 \wedge \cdots \wedge \mathtt{X}_n = E_n \wedge R_1)) = (\mathtt{X}_1 = E_{11} \wedge \cdots \wedge \mathtt{X}_n = E_{1n} \wedge R_1)$

  $\mathtt{sp}(C_2,\ (\mathtt{X}_1 = E_1 \wedge \cdots \wedge \mathtt{X}_n = E_n \wedge R_2)) = (\mathtt{X}_1 = E_{21} \wedge \cdots \wedge \mathtt{X}_n = E_{2n} \wedge R_2)$

- **If C_1 or C_2 don't assign to $\mathtt{X}_i$ then $E_i = E_{1i} = E_{2i}$ so**
  $(S\,[E_1/\mathtt{X}_1]\cdots[E_n/\mathtt{X}_n]{\rightarrow}E_{1i}|E_{2i}) = E_i$
  so formula above can be further simplified

- **If R determines the value of $S\,[E_1/\mathtt{X}_1]\cdots[E_n/\mathtt{X}_n]$ then can simplify**
  $(\mathtt{X}_i = (S\,[E_1/\mathtt{X}_1]\cdots[E_n/\mathtt{X}_n] \rightarrow E_{1i} \mid E_{2i}))$

Summary of $\texttt{sp}$ loop-free code symbolic execution

- **Symbolic state:** $(\texttt{X}_1 = E_1 \wedge \cdots \wedge \texttt{X}_n = E_n \wedge R)$
- $\texttt{sp}(\texttt{SKIP},\ (\texttt{X}_1 = E_1 \wedge \cdots \wedge \texttt{X}_n = E_n \wedge R)) = (\texttt{X}_1 = E_1 \wedge \cdots \wedge \texttt{X}_n = E_n \wedge R))$
- **If E_1, $\ldots$, E_n or R doesn't contain $\texttt{X}_1$, $\ldots$, $\texttt{X}_n$ then:**

$$\begin{aligned}&\texttt{sp}(\texttt{X}_i\texttt{:=}E,\ (\texttt{X}_1 = E_1 \wedge \cdots \wedge \texttt{X}_i = E_i \wedge \cdots \wedge \texttt{X}_n = E_n \wedge R))\\ &\ = \ (\texttt{X}_1 = E_1 \wedge \cdots \wedge \texttt{X}_i = E\texttt{[}E_1/\texttt{X}_1\texttt{]}\cdots\texttt{[}E_n/\texttt{X}_n\texttt{]} \wedge \cdots \wedge \texttt{X}_n = E_n \wedge R)\end{aligned}$$

- **If $\texttt{X} \neq \texttt{X}_i$ and $\texttt{X}$ doesn't occur in E_i or E for $1 \leq i \leq n$, then**

$$\begin{aligned}&\texttt{sp}(\texttt{X:=}E,\ (\texttt{X}_1 = E_1 \wedge \cdots \wedge \texttt{X}_n = E_n \wedge R))\\ &\ = \ (\texttt{X} = E\texttt{[}E_1/\texttt{X}_1\texttt{]}\cdots\texttt{[}E_n/\texttt{X}_n\texttt{]} \wedge \texttt{X}_1 = E_1 \wedge \cdots \wedge \texttt{X}_n = E_n \wedge R)\end{aligned}$$

- **If E_1, $\ldots$,E_n, R does not contain $\texttt{X}_1$, $\ldots$, $\texttt{X}_n$**

$$\begin{aligned}&\texttt{sp}(\texttt{IF}\ S\ \texttt{THEN}\ C_1\ \texttt{ELSE}\ C_2,\ (\texttt{X}_1 = E_1 \wedge \cdots \wedge \texttt{X}_n = E_n \wedge R))\\ &\ = \ (\texttt{X}_1{=}(S\texttt{[}E_1/\texttt{X}_1\texttt{]}\cdots\texttt{[}E_n/\texttt{X}_n\texttt{]}{\rightarrow}E_{11}|E_{21}))\wedge\cdots\wedge(\texttt{X}_n{=}(S\texttt{[}E_1/\texttt{X}_1\texttt{]}\cdots\texttt{[}E_n/\texttt{X}_n\texttt{]}{\rightarrow}E_{1n}|E_{2n})) \wedge R\end{aligned}$$

where

$$\begin{aligned}\texttt{sp}(C_1,\ (\texttt{X}_1 = E_1 \wedge \cdots \wedge \texttt{X}_n = E_n \wedge R_1)) &= (\texttt{X}_1 = E_{11} \wedge \cdots \wedge \texttt{X}_n = E_{1n} \wedge R_1)\\ \texttt{sp}(C_2,\ (\texttt{X}_1 = E_1 \wedge \cdots \wedge \texttt{X}_n = E_n \wedge R_2)) &= (\texttt{X}_1 = E_{21} \wedge \cdots \wedge \texttt{X}_n = E_{2n} \wedge R_2)\end{aligned}$$

Example symbolic computation using sp (1)

- $\mathtt{sp}(C_1\ ;\ C_2,\ P)\ =\ \mathtt{sp}(C_2,\ \mathtt{sp}(C_1,\ P))$ **hence**

$$
\begin{array}{l}
\mathtt{sp}(\mathtt{R} := \mathtt{0};\\
\quad \mathtt{K} := \mathtt{0};\\
\quad \mathtt{IF\ I} < \mathtt{J\ THEN\ K} := \mathtt{K} + \mathtt{1\ ELSE\ SKIP};\\
\quad \mathtt{IF\ K} = \mathtt{1} \wedge \neg(\mathtt{I} = \mathtt{J})\ \mathtt{THEN\ R} := \mathtt{J} - \mathtt{I\ ELSE\ R} := \mathtt{I} - \mathtt{J},\\
\quad (\mathtt{I} = i \wedge \mathtt{J} = j \wedge i < j))\ =
\end{array}
$$

$$
\begin{array}{l}
\mathtt{sp}(\mathtt{K} := \mathtt{0};\\
\quad \mathtt{IF\ I} < \mathtt{J\ THEN\ K} := \mathtt{K} + \mathtt{1\ ELSE\ SKIP};\\
\quad \mathtt{IF\ K} = \mathtt{1} \wedge \neg(\mathtt{I} = \mathtt{J})\ \mathtt{THEN\ R} := \mathtt{J} - \mathtt{I\ ELSE\ R} := \mathtt{I} - \mathtt{J},\\
\quad (\mathtt{R} = 0 \wedge \mathtt{I} = i \wedge \mathtt{J} = i \wedge i < j))\ =
\end{array}
$$

$$
\begin{array}{l}
\mathtt{sp}(\mathtt{IF\ I} < \mathtt{J\ THEN\ K} := \mathtt{K} + \mathtt{1\ ELSE\ SKIP};\\
\quad \mathtt{IF\ K} = \mathtt{1} \wedge \neg(\mathtt{I} = \mathtt{J})\ \mathtt{THEN\ R} := \mathtt{J} - \mathtt{I\ ELSE\ R} := \mathtt{I} - \mathtt{J},\\
\quad (\mathtt{K} = 0 \wedge \mathtt{R} = 0 \wedge \mathtt{I} = i \wedge \mathtt{J} = j \wedge i < j))
\end{array}
$$

Example symbolic computation using `sp` (2)

Considering each conditional branch and merging gives something like:

$$
\begin{array}{l}
\texttt{sp}(\texttt{IF I} < \texttt{J THEN K} := \texttt{K} + 1 \texttt{ ELSE SKIP};\\
\quad \texttt{IF K} = 1 \wedge \neg(\texttt{I} = \texttt{J}) \texttt{ THEN R} := \texttt{J} - \texttt{I} \texttt{ ELSE R} := \texttt{I} - \texttt{J},\\
\quad (\texttt{K} = 0 \wedge \texttt{R} = 0 \wedge \texttt{I} = i \wedge \texttt{J} = j \wedge i < j)) \ = \\
(\texttt{K}{=}(i{<}j \to E_{11} \mid E_{12}) \wedge \texttt{R}{=}(i{<}j \to E_{21} \mid E_{22}) \wedge \texttt{I}{=}(i{<}j \to E_{31} \mid E_{32}) \wedge \texttt{J}{=}(i{<}j \to E_{41} \mid E_{42}) \wedge i{<}j)
\end{array}
$$

Noting '$\wedge\ i < j$' it follows that this simplifies to:

$$
\begin{array}{l}
\texttt{sp}(\texttt{IF I} < \texttt{J THEN K} := \texttt{K} + 1 \texttt{ ELSE SKIP};\\
\quad \texttt{IF K} = 1 \wedge \neg(\texttt{I} = \texttt{J}) \texttt{ THEN R} := \texttt{J} - \texttt{I} \texttt{ ELSE R} := \texttt{I} - \texttt{J},\\
\quad (\texttt{K} = 0 \wedge \texttt{R} = 0 \wedge \texttt{I} = i \wedge \texttt{J} = j \wedge i < j)) \ = \\
(\texttt{K} = E_{11} \wedge \texttt{R} = E_{21} \wedge \texttt{I} = E_{31} \wedge \texttt{J} = E_{41} \wedge i < j)
\end{array}
$$

so need only consider the `I < J` branch

Example symbolic computation using sp (3)

$\mathtt{sp}(\mathtt{R} := 0;$
$\quad \mathtt{K} := 0;$
$\quad \mathtt{IF}\ \mathtt{I} < \mathtt{J}\ \mathtt{THEN}\ \mathtt{K} := \mathtt{K}+1\ \mathtt{ELSE}\ \mathtt{SKIP};$
$\quad \mathtt{IF}\ \mathtt{K} = 1 \wedge \neg(\mathtt{I} = \mathtt{J})\ \mathtt{THEN}\ \mathtt{R} := \mathtt{J} - \mathtt{I}\ \mathtt{ELSE}\ \mathtt{R} := \mathtt{I} - \mathtt{J},$
$\quad (\mathtt{I} = i \wedge \mathtt{J} = j \wedge i < j)) \ =$

$\mathtt{sp}(\mathtt{IF}\ \mathtt{K} = 1 \wedge \neg(\mathtt{I} = \mathtt{J})\ \mathtt{THEN}\ \mathtt{R} := \mathtt{J} - \mathtt{I}\ \mathtt{ELSE}\ \mathtt{R} := \mathtt{I} - \mathtt{J},$
$\quad (\mathtt{K} = (\mathtt{K}+1)[0/\mathtt{K}] \wedge \mathtt{R} = 0 \wedge \mathtt{I} = i \wedge \mathtt{J} = j \wedge i < j)) \ =$

$\mathtt{sp}(\mathtt{IF}\ \mathtt{K} = 1 \wedge \neg(\mathtt{I} = \mathtt{J})\ \mathtt{THEN}\ \mathtt{R} := \mathtt{J} - \mathtt{I}\ \mathtt{ELSE}\ \mathtt{R} := \mathtt{I} - \mathtt{J},$
$\quad (\mathtt{K} = 1 \wedge \mathtt{R} = 0 \wedge \mathtt{I} = i \wedge \mathtt{J} = j \wedge i < j)) \ =$ **(by a similar argument)**

$(\mathtt{K} = 1 \wedge \mathtt{R} = j{-}i \wedge \mathtt{I} = i \wedge \mathtt{J} = j \wedge i < j)$

Hence by $\{P\}\ C\ \{Q\} \ = \ \mathtt{sp}(C, P) \Rightarrow Q$ it follows that

$\{(\mathtt{I} = i \wedge \mathtt{J} = j \wedge i < j\}$
$\quad \mathtt{R} := 0;$
$\quad \mathtt{K} := 0;$
$\quad \mathtt{IF}\ \mathtt{I} < \mathtt{J}\ \mathtt{THEN}\ \mathtt{K} := \mathtt{K}+1\ \mathtt{ELSE}\ \mathtt{SKIP};$
$\quad \mathtt{IF}\ \mathtt{K} = 1 \wedge \neg(\mathtt{I} = \mathtt{J})\ \mathtt{THEN}\ \mathtt{R} := \mathtt{J} - \mathtt{I}\ \mathtt{ELSE}\ \mathtt{R} := \mathtt{I} - \mathtt{J})$
$\{R = j{-}i\}$

Same example using wlp

$$
\begin{array}{l}
\texttt{wlp}(\texttt{R} := 0; \\
\quad \texttt{K} := 0; \\
\quad \texttt{IF I} < \texttt{J THEN K} := \texttt{K} + 1 \texttt{ ELSE SKIP}; \\
\quad \texttt{IF K} = 1 \wedge \neg(\texttt{I} = \texttt{J}) \texttt{ THEN R} := \texttt{J} - \texttt{I} \texttt{ ELSE R} := \texttt{I} - \texttt{J}, \\
\quad (\texttt{R} = \texttt{J}{-}\texttt{I})) \ =
\end{array}
$$

$$
\begin{array}{l}
\texttt{wlp}(\texttt{R} := 0; \\
\quad \texttt{K} := 0; \\
\quad \texttt{IF I} < \texttt{J THEN K} := \texttt{K} + 1 \texttt{ ELSE SKIP}, \\
\quad (\texttt{K} = 1 \wedge \neg(\texttt{I} = \texttt{J}) \ \rightarrow \ \texttt{J}{-}\texttt{I} = \texttt{J}{-}\texttt{I} \ \mid \ \texttt{J}{-}\texttt{I} = \texttt{I}{-}\texttt{J}))
\end{array}
$$

$$
\begin{array}{l}
\texttt{wlp}(\texttt{R} := 0; \\
\quad \texttt{K} := 0; \\
\quad (\texttt{I} < \texttt{J} \ \rightarrow \ (\texttt{K} + 1 = 1 \wedge \neg(\texttt{I} = \texttt{J}) \ \rightarrow \ \texttt{J}{-}\texttt{I} = \texttt{J}{-}\texttt{I} \ \mid \ \texttt{J}{-}\texttt{I} = \texttt{I}{-}\texttt{J}) \\
\qquad\qquad \mid \ (\texttt{K} = 1 \wedge \neg(\texttt{I} = \texttt{J}) \ \rightarrow \ \texttt{J}{-}\texttt{I} = \texttt{J}{-}\texttt{I} \ \mid \ \texttt{J}{-}\texttt{I} = \texttt{I}{-}\texttt{J})) \ =
\end{array}
$$

$$
\begin{array}{l}
(\texttt{I} < \texttt{J} \ \rightarrow \ (0{+}1 = 1 \wedge \neg(\texttt{I} = \texttt{J}) \rightarrow \texttt{J}{-}\texttt{I} = \texttt{J}{-}\texttt{I} \mid \texttt{J}{-}\texttt{I} = \texttt{I}{-}\texttt{J}) \\
\qquad\qquad \mid \ (0 = 1 \wedge \neg(\texttt{I} = \texttt{J}) \rightarrow \texttt{J}{-}\texttt{I} = \texttt{J}{-}\texttt{I} \mid \texttt{J}{-}\texttt{I} = \texttt{I}{-}\texttt{J})) \ =
\end{array}
$$

$$
(\texttt{I} < \texttt{J} \ \rightarrow \ (\neg(\texttt{I} = \texttt{J}) \rightarrow \texttt{J}{-}\texttt{I} = \texttt{J}{-}\texttt{I} \mid \texttt{J}{-}\texttt{I} = \texttt{I}{-}\texttt{J}) \ \mid \ \texttt{J}{-}\texttt{I} = \texttt{I}{-}\texttt{J}) \ =
$$

$$
(\texttt{I} < \texttt{J} \ \rightarrow \ \texttt{J}{-}\texttt{I} = \texttt{J}{-}\texttt{I} \ \mid \ \texttt{J}{-}\texttt{I} = \texttt{I}{-}\texttt{J}) \ =
$$

$$
(\texttt{I} < \texttt{J})
$$

Example comparing sp with wlp

- **Going forward (simplifying on-the-fly)**

$$\begin{array}{l}\texttt{sp}(\texttt{R} := 0;\\ \quad \texttt{K} := 0;\\ \quad \texttt{IF I} < \texttt{J THEN K} := \texttt{K} + 1 \texttt{ ELSE SKIP};\\ \quad \texttt{IF K} = 1 \wedge \neg(\texttt{I} = \texttt{J}) \texttt{ THEN R} := \texttt{J} - \texttt{I} \texttt{ ELSE R} := \texttt{I} - \texttt{J},\\ \quad (\texttt{I} = i \wedge \texttt{J} = j \wedge i < j)) \;=\\ (\texttt{K} = 1 \wedge \texttt{R} = j{-}i \wedge \texttt{I} = i \wedge \texttt{J} = j \wedge i < j)\end{array}$$

- **Going backwards**

$$\begin{array}{l}\texttt{wlp}(\texttt{R} := 0;\\ \quad \texttt{K} := 0;\\ \quad \texttt{IF I} < \texttt{J THEN K} := \texttt{K} + 1 \texttt{ ELSE SKIP};\\ \quad \texttt{IF K} = 1 \wedge \neg(\texttt{I} = \texttt{J}) \texttt{ THEN R} := \texttt{J} - \texttt{I} \texttt{ ELSE R} := \texttt{I} - \texttt{J},\\ \quad (\texttt{R} = \texttt{J}{-}\texttt{I})) \;=\\ (\texttt{I} < \texttt{J} \;\rightarrow\; (0{+}1 = 1 \wedge \neg(\texttt{I} = \texttt{J}) \rightarrow \texttt{J}{-}\texttt{I} = \texttt{J}{-}\texttt{I} \mid \texttt{J}{-}\texttt{I} = \texttt{I}{-}\texttt{J})\end{array}$$

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'
    (see Specification and Verification II, especially 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
    (see later slides)

Exercises

- **Compute**

$\mathtt{sp}(\mathtt{R} := 0;$
$\quad \mathtt{K} := 0;$
$\quad \mathtt{IF\ I} < \mathtt{J\ THEN\ K} := \mathtt{K} + 1\ \mathtt{ELSE\ SKIP};$
$\quad \mathtt{IF\ K} = 1 \wedge \neg(\mathtt{I} = \mathtt{J})\ \mathtt{THEN\ R} := \mathtt{J} - \mathtt{I}\ \mathtt{ELSE\ R} := \mathtt{I} - \mathtt{J},$
$\quad (\mathtt{I} = i \wedge \mathtt{J} = j \wedge j \leq i))$

- **Hence show**

$\{(\mathtt{I} = i \wedge \mathtt{J} = j \wedge j \leq i\}$
$\quad \mathtt{R} := 0;$
$\quad \mathtt{K} := 0;$
$\quad \mathtt{IF\ I} < \mathtt{J\ THEN\ K} := \mathtt{K} + 1\ \mathtt{ELSE\ SKIP};$
$\quad \mathtt{IF\ K} = 1 \wedge \neg(\mathtt{I} = \mathtt{J})\ \mathtt{THEN\ R} := \mathtt{J} - \mathtt{I}\ \mathtt{ELSE\ R} := \mathtt{I} - \mathtt{J})$
$\{R = i - j\}$

- **Do same example use `wlp`**

Using sp to generate verification conditions

- **If C is loop-free then VC for $\{P\}\ C\ \{Q\}$ is $\texttt{sp}(C,\ P) \Rightarrow Q$**
- **Cannot in general compute a finite formula for $\texttt{sp}(\texttt{WHILE}\ S\ \texttt{DO}\ C,\ P)$**
- **The following holds**

  $\texttt{sp}(\texttt{WHILE}\ S\ \texttt{DO}\ C,\ P)\ = \texttt{sp}(\texttt{WHILE}\ S\ \texttt{DO}\ C,\ \texttt{sp}(C,\ (P \wedge S)))\ \vee\ (P \wedge \neg S)$
- **Above doesn't define $\texttt{sp}(C, P)$ to be a finite statement**
- **As with `wlp`, we will describe a hybrid VC and `sp` method**

sp-based verification conditions

- **Define** $\texttt{asp}(C, P)$ **to be an approximate strongest postcondition**
- **Define** $\texttt{svc}(C, P)$ **to be a set of verification conditions**
- **Idea is that if** $\bigwedge\texttt{svc}(C, P) \Rightarrow \{P\}\ C\ \{\texttt{asp}(C, P)\}$
- **If** C **is loop-free then**
  - $\texttt{asp}(C, P) = \texttt{sp}(C, P)$
  - $\texttt{svc}(C, P) = \{\}$

Definition of asp

- **Define asp recursively by:**

$$
\begin{array}{ll}
\texttt{asp}(P,\ \texttt{SKIP}) & = P \\
\texttt{asp}(P,\ V\ \texttt{:=}\ E) & = \exists v.\ V = E\texttt{[}v/V\texttt{]} \wedge P\texttt{[}v/V\texttt{]} \\
\texttt{asp}(P,\ C_1\ \texttt{;}\ C_2) & = \texttt{asp}(\texttt{asp}(P, C_1), C_2) \\
\texttt{asp}(P,\ \texttt{IF}\ S\ \texttt{THEN}\ C_1\ \texttt{ELSE}\ C_2) & = \texttt{asp}(P \wedge S,\ C_1)\ \vee\ \texttt{asp}(P \wedge \neg S,\ C_2) \\
\texttt{asp}(P,\ \texttt{WHILE}\ S\ \texttt{DO}\ \{R\}\ C) & = R \wedge \neg S
\end{array}
$$

Definition of svc

- **Define** svc **recursively by:**

$$
\begin{array}{ll}
\texttt{svc}(P,\ \texttt{SKIP}) & = \{\} \\
\texttt{svc}(P,\ V\ \texttt{:=}\ E) & = \{\} \\
\texttt{svc}(P,\ C_1\ \texttt{;}\ C_2) & = \texttt{svc}(P, C_1) \cup \texttt{svc}(\texttt{svc}_1(P, C_1), C_2) \\
\texttt{svc}(P,\ \texttt{IF}\ S\ \texttt{THEN}\ C_1\ \texttt{ELSE}\ C_2) & = \texttt{svc}(P \wedge S,\ C_1)\ \cup\ \texttt{svc}(P \wedge \neg S,\ C_2) \\
\texttt{svc}(P,\ \texttt{WHILE}\ S\ \texttt{DO}\ \{R\}\ C) & = \{P \Rightarrow R,\ \texttt{asp}(R \wedge S,\ C) \Rightarrow R\} \\
 & \quad \cup\ \texttt{svc}(R \wedge S,\ C)
\end{array}
$$

- **Theorem:** $\bigwedge\texttt{svc}(P, C) \Rightarrow \{P\}\ C\ \{\texttt{asp}(P, C)\}$
- **Proof by induction on** C **(exercise)**

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*