# Hoare Calculus and Predicate Transformers

Wolfgang Schreiner
Wolfgang.Schreiner@risc.uni-linz.ac.at

Research Institute for Symbolic Computation (RISC)
Johannes Kepler University, Linz, Austria
http://www.risc.uni-linz.ac.at



1. **The Hoare Calculus for Non-Loop Programs**
2. Predicate Transformers
3. Partial Correctness of Loop Programs
4. Total Correctness of Loop Programs
5. Abortion
6. Procedures



## The Hoare Calculus

Calculus for reasoning about imperative programs.

- "Hoare triple": $\{P\}\ c\ \{Q\}$
  - Logical propositions $P$ and $Q$, program command $c$.
  - The Hoare triple is itself a logical proposition.
  - The Hoare calculus gives rules for constructing true Hoare triples.
- Partial correctness interpretation of $\{P\}\ c\ \{Q\}$:

  "If $c$ is executed in a state in which $P$ holds, then it terminates in a state in which $Q$ holds unless it aborts or runs forever."
  - Program does not produce wrong result.
  - But program also need not produce any result.
    - Abortion and non-termination are not ruled out.
- Total correctness interpretation of $\{P\}\ c\ \{Q\}$:

  "If $c$ is executed in a state in which $P$ holds, then it terminates in a state in which $Q$ holds."
  - Program produces the correct result.

We will use the partial correctness interpretation for the moment.



## General Rules

$$\frac{P \Rightarrow Q}{\{P\}\ \{Q\}} \qquad \frac{P \Rightarrow P' \quad \{P'\}\ c\ \{Q'\} \quad Q' \Rightarrow Q}{\{P\}\ c\ \{Q\}}$$

- Logical derivation: $\dfrac{A_1 \ A_2}{B}$
  - Forward: If we have shown $A_1$ and $A_2$, then we have also shown $B$.
  - Backward: To show $B$, it suffices to show $A_1$ and $A_2$.
- Interpretation of above sentences:
  - To show that, if $P$ holds in a state, then $Q$ holds in the same state (no command is executed), it suffices to show $P$ implies $Q$.
    - Hoare triples are ultimately reduced to classical logic.
  - To show that, if $P$ holds, then $Q$ holds after executing $c$, it suffices to show this for a $P'$ weaker than $P$ and a $Q'$ stronger than $Q$.
    - Precondition may be weakened, postcondition may be strengthened.

## Special Commands

Commands modeling “emptiness” and abortion.

$$\{P\}\ \textbf{skip}\ \{P\} \qquad \{\text{true}\}\ \textbf{abort}\ \{\text{false}\}$$

- The **skip** command does not change the state; if $P$ holds before its execution, then $P$ thus holds afterwards as well.
- The **abort** command aborts execution and thus trivially satisfies partial correctness.
  - Axiom implies $\{P\}\ \textbf{abort}\ \{Q\}$ for arbitrary $P, Q$.

Useful commands for reasoning and program transformations.



## Scalar Assignments

$$\{Q[e/x]\}\ x := e\ \{Q\}$$

- Syntax
  - Variable $x$, expression $e$.
  - $Q[e/x] \ldots Q$ where every free occurrence of $x$ is replaced by $e$.
- Interpretation
  - To make sure that $Q$ holds for $x$ after the assignment of $e$ to $x$, it suffices to make sure that $Q$ holds for $e$ before the assignment.
- Partial correctness
  - Evaluation of $e$ may abort.

$$\begin{array}{rcc} \{x+3<5\} & x := x+3 & \{x<5\} \\ \{x<2\} & x := x+3 & \{x<5\} \end{array}$$



## Array Assignments

$$\{Q[a[i \mapsto e]/a]\}\ a[i] := e\ \{Q\}$$

- An array is modelled as a function $a : I \to V$
  - Index set $I$, value set $V$.
  - $a[i] = e \ldots$ a holds at index $i$ the value $e$.
- Updated array $a[i \mapsto e]$
  - Array that is constructed from $a$ by mapping index $i$ to value $e$.
  - Axioms (for all $a : I \to V, i \in I, j \in I, e \in V$):

$$\begin{array}{l} i = j \Rightarrow a[i \mapsto e][j] = e \\ i \neq j \Rightarrow a[i \mapsto e][j] = a[j] \end{array}$$

$$\begin{array}{rcc} \{a[i \mapsto x][1] > 0\} & a[i] := x & \{a[1] > 0\} \\ \{(i = 1 \Rightarrow x > 0) \wedge (i \neq 1 \Rightarrow a[1] > 0)\} & \text{a[i]} := \text{x} & \{a[1] > 0\} \end{array}$$

Index violations and pointer semantics of arrays not yet considered.



## Command Sequences

$$\frac{\{P\}\ c_1\ \{R_1\} \quad R_1 \Rightarrow R_2 \quad \{R_2\}\ c_2\ \{Q\}}{\{P\}\ c_1; c_2\ \{Q\}}$$

- Interpretation
  - To show that, if $P$ holds before the execution of $c_1; c_2$, then $Q$ holds afterwards, it suffices to show for some $R_1$ and $R_2$ with $R_1 \Rightarrow R_2$ that
    - if $P$ holds before $c_1$, that $R_1$ holds afterwards, and that
    - if $R_2$ holds before $c_2$, then $Q$ holds afterwards.
- Problem: find suitable $R_1$ and $R_2$
  - Easy in many cases (see later).

$$\frac{\{x+y-1>0\}\ y := y-1\ \{x+y>0\} \quad \{x+y>0\}\ x := x+y\ \{x>0\}}{\{x+y-1>0\}\ y := y-1; x := x+y\ \{x>0\}}$$

## Conditionals

$$\frac{\{P \land b\}\ c_1\ \{Q\} \quad \{P \land \neg b\}\ c_2\ \{Q\}}{\{P\}\ \textbf{if}\ b\ \textbf{then}\ c_1\ \textbf{else}\ c_2\ \{Q\}}$$

$$\frac{\{P \land b\}\ c\ \{Q\} \quad (P \land \neg b) \Rightarrow Q}{\{P\}\ \textbf{if}\ b\ \textbf{then}\ c\ \{Q\}}$$

- Interpretation
  - To show that, if $P$ holds before the execution of the conditional, then $Q$ holds afterwards,
  - it suffices to show that the same is true for each conditional branch, under the additional assumption that this branch is executed.

$$\frac{\{x \neq 0 \land x \geq 0\}\ y := x\ \{y > 0\} \quad \{x \neq 0 \land x \not\geq 0\}\ y := -x\ \{y > 0\}}{\{x \neq 0\}\ \textbf{if}\ x \geq 0\ \textbf{then}\ y := x\ \textbf{else}\ y := -x\ \{y > 0\}}$$



1. The Hoare Calculus for Non-Loop Programs

2. **Predicate Transformers**

3. Partial Correctness of Loop Programs

4. Total Correctness of Loop Programs

5. Abortion

6. Procedures



## Backward Reasoning

Implication of rule for command sequences and rule for assignments:

$$\frac{\{P\}\ c\ \{Q[e/x]\}}{\{P\}\ c; x := e\ \{Q\}}$$

- Interpretation
  - If the last command of a sequence is an assignment, we can remove the assignment from the proof obligation.
  - By multiple application, assignment sequences can be removed from the back to the front.

| | | | | |
|---|---|---|---|---|
| $\{P\}$ | $\{P\}$ | $\{P\}$ | $\{P\}$ | $P \Rightarrow x = 4$ |
| x := x+1; | x := x+1; | x := x+1; | $\{x + 1 = 5\}$ | |
| y := 2*x; | y := 2*x; | $\{x + 2x = 15\}$ | $(\Leftrightarrow x = 4)$ | |
| z := x+y | $\{x + y = 15\}$ | $(\Leftrightarrow 3x = 15)$ | | |
| $\{z = 15\}$ | | $(\Leftrightarrow x = 5)$ | | |



## Weakest Preconditions

A calculus for "backward reasoning".

- Predicate transformer wp
  - Function "wp" that takes a command $c$ and a postcondition $Q$ and returns a precondition.
  - Read $\text{wp}(c, Q)$ as "the weakest precondition of $c$ w.r.t. $Q$".
- $\text{wp}(c, Q)$ is a precondition for $c$ that ensures $Q$ as a postcondition.
  - Must satisfy $\{\text{wp}(c, Q)\}\ c\ \{Q\}$.
- $\text{wp}(c, Q)$ is the weakest such precondition.
  - Take any $P$ such that $\{P\}\ c\ \{Q\}$.
  - Then $P \Rightarrow \text{wp}(P, Q)$.
- Consequence: $\{P\}\ c\ \{Q\}$ iff $(P \Rightarrow \text{wp}(c, Q))$
  - We want to prove $\{P\}\ c\ \{Q\}$.
  - We may prove $P \Rightarrow \text{wp}(c, Q)$ instead.

Verification is reduced to the calculation of weakest preconditions.

## Weakest Preconditions

The weakest precondition of each program construct.

$$\begin{aligned}
&\mathrm{wp}(\textbf{skip}, Q) \Leftrightarrow Q\\
&\mathrm{wp}(\textbf{abort}, Q) \Leftrightarrow \text{true}\\
&\mathrm{wp}(x := e, Q) \Leftrightarrow Q[e/x]\\
&\mathrm{wp}(c_1; c_2, Q) \Leftrightarrow \mathrm{wp}(c_1, \mathrm{wp}(c_2, Q))\\
&\mathrm{wp}(\textbf{if } b \textbf{ then } c_1 \textbf{ else } c_2, Q) \Leftrightarrow (b \Rightarrow \mathrm{wp}(c_1, Q)) \land (\neg b \Rightarrow \mathrm{wp}(c_2, Q))\\
&\mathrm{wp}(\textbf{if } b \textbf{ then } c, Q) \Leftrightarrow (b \Rightarrow \mathrm{wp}(c, Q)) \land (\neg b \Rightarrow Q)
\end{aligned}$$

Alternative formulation of a program calculus.



## Forward Reasoning

Sometimes, we want to derive a postcondition from a given precondition.

$$\{P\}\ x := e\ \{\exists x_0 : P[x_0/x] \land x = e[x_0/x]\}$$

- Forward Reasoning
  - What is the maximum we know about the post-state of an assignment $x := e$, if the pre-state satisfies $P$?
  - We know that $P$ holds for some value $x_0$ (the value of $x$ in the pre-state) and that $x$ equals $e[x_0/x]$.

$$\begin{gathered}
\{x \geq 0 \land y = a\}\\
x := x + 1\\
\{\exists x_0 : x_0 \geq 0 \land y = a \land x = x_0 + 1\}\\
(\Leftrightarrow (\exists x_0 : x_0 \geq 0 \land x = x_0 + 1) \land y = a)\\
(\Leftrightarrow x > 0 \land y = a)
\end{gathered}$$



## Strongest Postcondition

A calculus for forward reasoning.

- Predicate transformer sp
  - Function "sp" that takes a precondition $P$ and a command $c$ and returns a postcondition.
  - Read $\mathrm{sp}(P, c)$ as "the strongest postcondition of $c$ w.r.t. $P$".
- $\mathrm{sp}(P, c)$ is a postcondition for $c$ that is ensured by precondition $P$.
  - Must satisfy $\{P\}\ c\ \{\mathrm{sp}(P, c)\}$.
- $\mathrm{sp}(P, c)$ is the strongest such postcondition.
  - Take any $P, Q$ such that $\{P\}\ c\ \{Q\}$.
  - Then $\mathrm{sp}(P, c) \Rightarrow Q$.
- Consequence: $\{P\}\ c\ \{Q\}$ iff $(\mathrm{sp}(P, c) \Rightarrow Q)$.
  - We want to prove $\{P\}\ c\ \{Q\}$.
  - We may prove $\mathrm{sp}(P, c) \Rightarrow Q$ instead.

Verification is reduced to the calculation of strongest postconditions.



## Strongest Postconditions

The strongest postcondition of each program construct.

$$\begin{aligned}
&\mathrm{sp}(P, \textbf{skip}) \Leftrightarrow P\\
&\mathrm{sp}(P, \textbf{abort}) \Leftrightarrow \text{false}\\
&\mathrm{sp}(P, x := e) \Leftrightarrow \exists x_0 : P[x_0/x] \land x = e[x_0/x]\\
&\mathrm{sp}(P, c_1; c_2) \Leftrightarrow sp(sp(P, c_1), c_2)\\
&\mathrm{sp}(P, \textbf{if } b \textbf{ then } c_1 \textbf{ else } c_2) \Leftrightarrow (b \Rightarrow \mathrm{sp}(P, c_1)) \land (\neg b \Rightarrow \mathrm{sp}(P, c_2))\\
&\mathrm{sp}(P, \textbf{if } b \textbf{ then } c) \Leftrightarrow (b \Rightarrow \mathrm{sp}(P, c)) \land (\neg b \Rightarrow P)
\end{aligned}$$

The use of predicate transformers is an alternative/supplement to the Hoare calculus; this view is due to Dijkstra.

1. The Hoare Calculus for Non-Loop Programs
2. Predicate Transformers
3. **Partial Correctness of Loop Programs**
4. Total Correctness of Loop Programs
5. Abortion
6. Procedures



## The Hoare Calculus and Loops

$$\{\text{true}\}\ \textbf{loop}\ \{\text{false}\} \qquad \frac{P \Rightarrow I \quad \{I \wedge b\}\ c\ \{I\} \quad (I \wedge \neg b) \Rightarrow Q}{\{P\}\ \textbf{while}\ b\ \textbf{do}\ c\ \{Q\}}$$

- Interpretation:
  - The **loop** command does not terminate and thus trivially satisfies partial correctness.
    - Axiom implies $\{P\}$ **loop** $\{Q\}$ for arbitrary $P, Q$.
  - To show that, if before the execution of a **while**-loop the property $P$ holds, after its termination the property $Q$ holds, it suffices to show for some property $I$ (the loop invariant) that
    - $I$ holds before the loop is executed (i.e. that $P$ implies $I$),
    - if $I$ holds when the loop body is entered (i.e. if also $b$ holds), that after the execution of the loop body $I$ still holds,
    - when the loop terminates (i.e. if $b$ does not hold), $I$ implies $Q$.
- Problem: find appropriate loop invariant $I$.
  - Strongest relationship between all variables modified in loop body.



## Example

$$I :\Leftrightarrow (n \geq 0 \Rightarrow 1 \leq i \leq n+1) \wedge s = \textstyle\sum_{j=1}^{i-1} j$$

$$\frac{\begin{array}{c}(i = 1 \wedge s = 0) \Rightarrow I \\ \{I \wedge i \leq 0\}\ s := s+i; i := i+1\ \{I\} \\ (I \wedge i \not\leq n) \Rightarrow s = \sum_{j=1}^{n} j\end{array}}{\{i = 1 \wedge s = 0\}\ \textbf{while}\ i \leq n\ \textbf{do}\ (s := s+i; i := i+1)\ \{s = \sum_{j=1}^{n} j\}}$$

The invariant captures the "essence" of a loop; only by giving its invariant, a true understanding of a loop is demonstrated.



## Practical Aspects

We want to verify the following program:

$$\{P\}\ c_1; \textbf{while}\ b\ \textbf{do}\ c; c_2\ \{Q\}$$

- Assume $c_1$ and $c_2$ do not contain loop commands.
- It suffices to prove

$$\{\text{sp}(P, c_1)\}\ \textbf{while}\ b\ \textbf{do}\ c\ \{\text{wp}(c_2, Q)\}$$

Verification of loops is the core of most program verifications.

## Weakest Liberal Preconditions for Loops

$$\mathrm{wp}(\textbf{loop}, Q) \Leftrightarrow \text{true}$$
$$\mathrm{wp}(\textbf{while } b \textbf{ do } c, Q) \Leftrightarrow \forall i \in \mathbb{N} : L_i(Q)$$

$$L_0(Q) :\Leftrightarrow \text{true}$$
$$L_{i+1}(Q) :\Leftrightarrow (\neg b \Rightarrow Q) \wedge (b \Rightarrow \mathrm{wp}(c, L_i(Q)))$$

- Interpretation
  - Weakest precondition that ensures that loops stops in a state satisfying $Q$, unless it aborts or runs forever.
- Infinite sequence of predicates $L_i(Q)$:
  - Weakest precondition that ensures that loops stops after less than $i$ iterations in a state satisfying $Q$, unless it aborts or runs forever.
- Alternative view: $L_i(Q) \Leftrightarrow \mathrm{wp}(\mathrm{if}_i, Q)$

  $\mathrm{if}_0 := \textbf{loop}$

  $\mathrm{if}_{i+1} := \textbf{if } b \textbf{ then } (c; \mathrm{if}_i)$



## Example

$$\mathrm{wp}(\textbf{while } i < n \textbf{ do } i := i+1, Q)$$

$$\begin{aligned}
L_0(Q) &\Leftrightarrow \text{true}\\
L_1(Q) &\Leftrightarrow (i \not< n \Rightarrow Q) \wedge (i < n \Rightarrow \mathrm{wp}(i := i+1, \text{true}))\\
&\Leftrightarrow (i \not< n \Rightarrow Q) \wedge (i < n \Rightarrow \text{true})\\
&\Leftrightarrow (i \not< n \Rightarrow Q)\\
L_2(Q) &\Leftrightarrow (i \not< n \Rightarrow Q) \wedge (i < n \Rightarrow \mathrm{wp}(i := i+1, i \not< n \Rightarrow Q))\\
&\Leftrightarrow (i \not< n \Rightarrow Q) \wedge\\
&\qquad (i < n \Rightarrow (i+1 \not< n \Rightarrow Q[i+1/i]))\\
L_3(Q) &\Leftrightarrow (i \not< n \Rightarrow Q) \wedge (i < n \Rightarrow \mathrm{wp}(i := i+1,\\
&\qquad (i \not< n \Rightarrow Q) \wedge (i < n \Rightarrow (i+1 \not< n \Rightarrow Q[i+1/i]))))\\
&\Leftrightarrow (i \not< n \Rightarrow Q) \wedge\\
&\qquad (i < n \Rightarrow ((i+1 \not< n \Rightarrow Q[i+1/i]) \wedge\\
&\qquad\quad (i+1 < n \Rightarrow (i+2 \not< n \Rightarrow Q[i+2/i]))))
\end{aligned}$$



## Weakest Liberal Preconditions for Loops

- Sequence $L_i(Q)$ is monotonically increasing in strength:
  - $\forall i \in \mathbb{N} : L_{i+1}(Q) \Rightarrow L_i(Q)$.
- The weakest precondition is the "lowest upper bound":
  - $\mathrm{wp}(\textbf{while } b \textbf{ do } c, Q) \Rightarrow \forall i \in \mathbb{N} : L_i(Q)$.
  - $\forall P : (P \Rightarrow \forall i \in \mathbb{N} : L_i(Q)) \Rightarrow (P \Rightarrow \mathrm{wp}(\textbf{while } b \textbf{ do } c, Q))$.
- We can only compute weaker approximation $L_i(Q)$.
  - $\mathrm{wp}(\textbf{while } b \textbf{ do } c, Q) \Rightarrow L_i(Q)$.
- We want to prove $\{P\}$ **while** $b$ **do** $c$ $\{Q\}$.
  - This is equivalent to proving $P \Rightarrow \mathrm{wp}(\textbf{while } b \textbf{ do } c, Q)$.
  - Thus $P \Rightarrow L_i(Q)$ must hold as well.
- If we can prove $\neg(P \Rightarrow L_i(Q))$, ...
  - $\{P\}$ **while** $b$ **do** $c$ $\{Q\}$ does not hold.
  - If we fail, we may try the easier proof $\neg(P \Rightarrow L_{i+1}(Q))$.

Falsification is possible by use of approximation $L_i$, but verification is not.



1. The Hoare Calculus for Non-Loop Programs
2. Predicate Transformers
3. Partial Correctness of Loop Programs
**4. Total Correctness of Loop Programs**
5. Abortion
6. Procedures

## Total Correctness of Loops

Hoare rules for **loop** and **while** are replaced as follows:

$$\{\text{false}\}\ \textbf{loop}\ \{\text{false}\} \qquad \frac{P \Rightarrow I \quad I \wedge b \Rightarrow t > 0 \qquad \{I \wedge b \wedge t = N\}\ c\ \{I \wedge t < N\} \quad (I \wedge \neg b) \Rightarrow Q}{\{P\}\ \textbf{while}\ b\ \textbf{do}\ c\ \{Q\}}$$

- New interpretation of $\{P\}\ c\ \{Q\}$.
  - If execution of $c$ starts in a state where $P$ holds, then execution terminates in a state where $Q$ holds, unless it aborts.
  - Non-termination is ruled out, abortion not (yet).
  - The **loop** command thus does not satisfy total correctness.
- Termination term $t$.
  - Denotes a natural number before and after every loop iteration.
  - If $t = N$ before an iteration, then $t < N$ after the iteration.
  - Consequently, if term denotes zero, loop must terminate.

Instead of the natural numbers, any *well-founded ordering* may be used for the domain of $t$.



## Example

$$I :\Leftrightarrow (n \geq 0 \Rightarrow 1 \leq i \leq n+1) \wedge s = \textstyle\sum_{j=1}^{i-1} j$$

$$\frac{(i = 1 \wedge s = 0) \Rightarrow I \quad I \wedge i \leq n \Rightarrow n - i + 1 > 0 \qquad \{I \wedge i \leq 0 \wedge n - i + 1 = N\}\ s := s + i; i := i + 1\ \{I \wedge n - i + 1 < N\} \qquad (I \wedge i \not\leq n) \Rightarrow s = \sum_{j=1}^{n} j}{\{i = 1 \wedge s = 0\}\ \textbf{while}\ i \leq n\ \textbf{do}\ (s := s + i; i := i + 1)\ \{s = \sum_{j=1}^{n} j\}}$$

In practice, termination is easy to show (compared to partial correctness).



## Weakest Preconditions for Loops

$$\begin{aligned}
&\text{wp}(\textbf{loop}, Q) \Leftrightarrow \text{false}\\
&\text{wp}(\textbf{while}\ b\ \textbf{do}\ c, Q) \Leftrightarrow \exists i \in \mathbb{N} : L_i(Q)\\[1em]
&L_0(Q) :\Leftrightarrow \text{false}\\
&L_{i+1}(Q) :\Leftrightarrow (\neg b \Rightarrow Q) \wedge (b \Rightarrow \text{wp}(c, L_i(Q)))
\end{aligned}$$

- New interpretation
  - Weakest precondition that ensures that the loop terminates in a state in which $Q$ holds, unless it aborts.
- New interpretation of $L_i(Q)$
  - Weakest precondition that ensures that the loop terminates after less than $i$ iterations in a state in which $Q$ holds, unless it aborts.
- Preserves property: $\{P\}\ c\ \{Q\}$ iff $(P \Rightarrow \text{wp}(c, Q))$
  - Now for total correctness interpretation of Hoare calculus.
- Preserves alternative view: $L_i(Q) \Leftrightarrow \text{wp}(\text{if}_i, Q)$

  $\text{if}_0 := \textbf{loop}$

  $\text{if}_{i+1} := \textbf{if}\ b\ \textbf{then}\ (c; \text{if}_i)$



## Example

$$\text{wp}(\textbf{while}\ i < n\ \textbf{do}\ i := i + 1, Q)$$

$$\begin{aligned}
L_0(Q) &:\Leftrightarrow \text{false}\\
L_1(Q) &:\Leftrightarrow (i \not< n \Rightarrow Q) \wedge (i < n \Rightarrow \ wp(i := i+1, L_0(Q)))\\
&\Leftrightarrow (i \not< n \Rightarrow Q) \wedge (i < n \Rightarrow \ \text{false})\\
&\Leftrightarrow i \not< n \wedge Q\\
L_2(Q) &:\Leftrightarrow (i \not< n \Rightarrow Q) \wedge (i < n \Rightarrow \ wp(i := i+1, L_1(Q)))\\
&\Leftrightarrow (i \not< n \Rightarrow Q) \wedge\\
&\qquad i < n \Rightarrow (i + 1 \not< n \wedge Q[i+1/i]))\\
L_3(Q) &:\Leftrightarrow (i \not< n \Rightarrow Q) \wedge (i < n \Rightarrow \ \text{wp}(i := i+1, L_2(Q)))\\
&\Leftrightarrow (i \not< n \Rightarrow Q) \wedge\\
&\qquad (i < n \Rightarrow \ ((i + 1 \not< n \Rightarrow Q[i+1/i]) \wedge\\
&\qquad\qquad (i + 1 < n \Rightarrow (i + 2 \not< n \wedge Q[i+2/i]))))
\end{aligned}$$

...

## Weakest Preconditions for Loops

- Sequence $L_i(Q)$ is now monotonically decreasing in strength:
  - $\forall i \in \mathbb{N} : L_i(Q) \Rightarrow L_{i+1}(Q)$.
- The weakest precondition is the "greatest lower bound":
  - $(\forall i \in \mathbb{N} : L_i(Q)) \Rightarrow \text{wp}(\textbf{while } b \textbf{ do } c, Q)$.
  - $\forall P : ((\forall i \in \mathbb{N} : L_i(Q)) \Rightarrow P) \Rightarrow (\text{wp}(\textbf{while } b \textbf{ do } c, Q) \Rightarrow P)$.
- We can only compute a stronger approximation $L_i(Q)$.
  - $L_i(Q) \Rightarrow \text{wp}(\textbf{while } b \textbf{ do } c, Q)$.
- We want to prove $\{P\}\ c\ \{Q\}$.
  - It suffices to prove $P \Rightarrow \text{wp}(\textbf{while } b \textbf{ do } c, Q)$.
  - It thus also suffices to prove $P \Rightarrow L_i(Q)$.
  - If proof fails, we may try the easier proof $P \Rightarrow L_{i+1}(Q)$

Verifications are typically not successful with finite approximation of weakest precondition.



1. The Hoare Calculus for Non-Loop Programs

2. Predicate Transformers

3. Partial Correctness of Loop Programs

4. Total Correctness of Loop Programs

5. Abortion

6. Procedures



## Abortion

New rules to prevent abortion.

$$\{\text{false}\}\ \textbf{abort}\ \{\text{true}\}$$
$$\{Q[e/x] \land D(e)\}\ x := e\ \{Q\}$$
$$\{Q[a[i \mapsto e]/a] \land D(e) \land 0 \leq i < \text{length}(a)\}\ a[i] := e\ \{Q\}$$

- New interpretation of $\{P\}\ c\ \{Q\}$.
  - If execution of $c$ starts in a state, in which property $P$ holds, then it does not abort and eventually terminates in a state in which $Q$ holds.
- Sources of abortion.
  - Division by zero.
  - Index out of bounds exception.

$D(e)$ makes sure that every subexpression of $e$ is well defined.



## Definedness of Expressions

$D(0) :\Leftrightarrow \text{true}$.
$D(1) :\Leftrightarrow \text{true}$.
$D(x) :\Leftrightarrow \text{true}$.
$D(a[i]) :\Leftrightarrow D(i) \land 0 \leq i < \text{length}(a)$.
$D(e_1 + e_2) :\Leftrightarrow D(e_1) \land D(e_2)$.
$D(e_1 * e_2) :\Leftrightarrow D(e_1) \land D(e_2)$.
$D(e_1/e_2) :\Leftrightarrow D(e_1) \land D(e_2) \land e_2 \neq 0$.
$D(\text{true}) :\Leftrightarrow \text{true}$.
$D(\text{false}) :\Leftrightarrow \text{true}$.
$D(\neg b) :\Leftrightarrow D(b)$.
$D(b_1 \land b_2) :\Leftrightarrow D(b_1) \land D(b_2)$.
$D(b_1 \lor b_2) :\Leftrightarrow D(b_1) \land D(b_2)$.
$D(e_1 < e_2) :\Leftrightarrow D(e_1) \land D(e_2)$.
$D(e_1 \leq e_2) :\Leftrightarrow D(e_1) \land D(e_2)$.
$D(e_1 > e_2) :\Leftrightarrow D(e_1) \land D(e_2)$.
$D(e_1 \geq e_2) :\Leftrightarrow D(e_1) \land D(e_2)$.

Assumes that expressions have already been type-checked.

## Abortion

Slight modification of existing rules.

$$\frac{\{P \wedge b \wedge D(b)\}\ c_1\ \{Q\} \quad \{P \wedge \neg b \wedge D(b)\}\ c_2\ \{Q\}}{\{P\}\ \textbf{if}\ b\ \textbf{then}\ c_1\ \textbf{else}\ c_2\ \{Q\}}$$

$$\frac{\{P \wedge b \wedge D(b)\}\ c\ \{Q\} \quad (P \wedge \neg b \wedge D(b)) \Rightarrow Q}{\{P\}\ \textbf{if}\ b\ \textbf{then}\ c\ \{Q\}}$$

$$\frac{\begin{array}{c} P \Rightarrow I \quad I \Rightarrow (T \in \mathbb{N} \wedge D(b)) \\ \{I \wedge b \wedge T = t\}\ c\ \{I \wedge T < t\} \quad (I \wedge \neg b) \Rightarrow Q \end{array}}{\{P\}\ \textbf{while}\ b\ \textbf{do}\ c\ \{Q\}}$$

Expressions must be defined in any context.



## Abortion

Similar modifications of weakest preconditions.

$\mathrm{wp}(\textbf{abort}, Q) \Leftrightarrow \text{false}$
$\mathrm{wp}(x := e, Q) \Leftrightarrow Q[e/x] \wedge D(e)$
$\mathrm{wp}(\textbf{if}\ b\ \textbf{then}\ c_1\ \textbf{else}\ c_2, Q) \Leftrightarrow$
$\quad D(b) \wedge (b \Rightarrow \mathrm{wp}(c_1, Q)) \wedge (\neg b \Rightarrow \mathrm{wp}(c_2, Q))$
$\mathrm{wp}(\textbf{if}\ b\ \textbf{then}\ c, Q) \Leftrightarrow D(b) \wedge (b \Rightarrow \mathrm{wp}(c, Q)) \wedge (\neg b \Rightarrow Q)$
$\mathrm{wp}(\textbf{while}\ b\ \textbf{do}\ c, Q) \Leftrightarrow \exists i \in \mathbb{N} : L_i(Q)$

$L_0(Q) :\Leftrightarrow \text{false}$
$L_{i+1}(Q) :\Leftrightarrow D(b) \wedge (\neg b \Rightarrow Q) \wedge (b \Rightarrow \mathrm{wp}(c, L_i(Q)))$

$\mathrm{wp}(c, Q)$ now makes sure that the execution of $c$ does not abort but eventually terminates in a state in which $Q$ holds.



1. The Hoare Calculus for Non-Loop Programs
2. Predicate Transformers
3. Partial Correctness of Loop Programs
4. Total Correctness of Loop Programs
5. Abortion
6. **Procedures**



## Procedure Specifications

global $F$;
requires $Pre$;
ensures $Post$;
$p(i, t, o)$ { $c$ }

- Specification of procedure $p(i, t, o)$.
  - Input parameter $i$, transient parameter $t$, output parameter $o$.
    - A call has form $p(e, x, y)$ for expression $e$ and variables $x$ and $y$.
  - Set of global variables ("frame") $F$.
    - Those global variables that $p$ may read/write (in addition to $i, t, o$).
    - Let $f$ denote all variables in $F$; let $g$ denote all variables not in $F$.
  - Precondition $Pre$ (may refer to $i, t, f$).
  - Postcondition $Post$ (may refer to $i, t, t_0, f, f_0, o$).
- Proof obligation

$$\{Pre \wedge i_0 = i \wedge t_0 = t \wedge f_0 = f\}\ c\ \{Post[i_0/i]\}$$

## Procedure Calls

First let us give an alternative (equivalent) version of the assignment rule.

- Original:

$$\{D(e) \wedge Q[e/x]\}$$
$$x := e$$
$$\{Q\}$$

- Alternative:

$$\{D(e) \wedge \forall x' : x' = e \Rightarrow Q[x'/x]\}$$
$$x := e$$
$$\{Q\}$$

The new value of $x$ is given name $x'$ in the precondition.



## Procedure Calls

From this, we can derive a rule for the correctness of procedure calls.

$$\{D(e) \wedge Pre[e/i, x/t] \wedge$$
$$\forall x', y', f' : Post[e/i, x/t_0, x'/t, y'/o, f/f_0, f'/f] \Rightarrow Q[x'/x, y'/y, f'/f]\}$$
$$p(e, x, y)$$
$$\{Q\}$$

- $Pre[e/i, x/t]$ refers to the values of the actual arguments $e$ and $x$ (rather than to the formal parameters $i$ and $t$).
- $x', y', f'$ denote the values of the vars $x$, $y$, and $f$ after the call.
- $Post[\ldots]$ refers to the argument values before and after the call.
- $Q[x'/x, y'/y, f'/f]$ refers to the argument values after the call.

Modular reasoning: rule only relies on the *specification* of $p$, not on its implementation.



## Corresponding Predicate Transformers

$$\begin{array}{l}
\mathrm{wp}(p(e,x,y), Q) \Leftrightarrow \\
\quad D(e) \wedge Pre[e/i, x/t] \wedge \\
\quad \forall x', y', f' : \\
\quad\quad Post[e/i, x/t_0, x'/t, y'/o, f/f_0, f'/f] \Rightarrow Q[x'/x, y'/y, f'/f]
\end{array}$$

$$\begin{array}{l}
\mathrm{sp}(P, p(e,x,y)) \Leftrightarrow \\
\quad \exists x_0, y_0, f_0 : \\
\quad\quad P[x_0/x, y_0/y, f_0/f] \wedge \\
\quad\quad Post[e[x_0/x, y_0/y, f_0/f]/i, x_0/t_0, x/t, y/o]
\end{array}$$

Explicit naming of old/new values required.



## Procedure Calls Example

- Procedure specification:

  global $f$
  requires $f \geq 0 \wedge i > 0$
  ensures $f_0 = f \cdot i + o \wedge 0 \leq o < i$
  $dividesF(i, o)$

- Procedure call:

  $\{f \geq 0 \wedge f = N \wedge b \geq 0\}$
  $dividesF(b+1, y)$
  $\{f \cdot (b+1) \leq N < (f+1) \cdot (b+1)\}$

- To be ultimately proved:

$$\begin{array}{l}
f \geq 0 \wedge f = N \wedge b \geq 0 \Rightarrow \\
\quad D(b+1) \wedge f \geq 0 \wedge b+1 > 0 \wedge \\
\quad \forall y', f' : \\
\quad\quad f = f' \cdot (b+1) + y' \wedge 0 \leq y' < b+1 \Rightarrow \\
\quad\quad\quad f' \cdot (b+1) \leq N < (f'+1) \cdot (b+1)
\end{array}$$

# Not Yet Covered

- Primitive data types.
  - `int` values are actually finite precision integers.
- More data and control structures.
  - switch, do-while (easy); continue, break, return (more complicated).
  - Records can be handled similar to arrays.
- Recursion.
  - Procedures may not terminate due to recursive calls.
- Exceptions and Exception Handling.
  - Short discussion in the context of ESC/Java2 later.
- Pointers and Objects.
  - Here reasoning gets complicated.
- ...

The more features are covered, the more complicated reasoning becomes.