Our Thesis

Formal proofs of program specifications (more precisely, proofs that specifications follow from their verification conditions) are best communicated by *annotated specifications* (sometimes called *proof outlines*), in which intermediate assertions and other notations are interspersed within the specification.

These annotated specifications can be defined by inference rules and mechanically translated into conventional formal proofs.
A Program for Fast Division

\{ x \geq 0 \land y > 0 \}\)

newvar n := 0 in newvar z := y in

\( \text{while } z \leq x \text{ do } (n := n + 1 ; z = z \times 2) ; \)

q := 0 ; r := x ;

while n \neq 0 do

\( (n := n - 1 ; z := z \div 2 ; q := q \times 2 ; \)

\text{if } z \leq r \text{ then } q := q + 1 ; r := r - z \text{ else skip} ) )

\{ x = q \times y + r \land 0 \leq r < y \}
A Formal Proof for Fast Division

The invariants:

\[ I_0 \overset{\text{def}}{=} z = y \times 2^n \land n \geq 0 \land x \geq 0 \land y > 0 \]

\[ I_1 \overset{\text{def}}{=} x = q \times z + r \land 0 \leq r < z \land z = y \times 2^n \land n \geq 0 \]

The proof:

1. \((x \geq 0 \land y > 0) \Rightarrow (y = y \times 2^0 \land 0 \geq 0 \land x \geq 0 \land y > 0)\)

2. \(\{x \geq 0 \land y > 0\}\)
\[ n := 0 ; z := y \]
\[ \{I_0\} \quad \text{(RAS,1)} \]

3. \((I_0 \land z \leq x) \Rightarrow (z \times 2 = y \times 2^{n+1} \land n + 1 \geq 0 \land x \geq 0 \land y > 0)\)

4. \(\{I_0 \land z \leq x\}\)
\[ n := n + 1 ; z = z \times 2 \]
\[ \{I_0\} \quad \text{(RAS,3)} \]

5. \(\{I_0\}\)
\[ \text{while } z \leq x \text{ do } (n := n + 1 ; z := z \times 2) \]
\[ \{I_0 \land \neg z \leq x\} \quad \text{(WH,4)} \]
6. \((I_0 \land \neg z \leq x) \Rightarrow (x = 0 \times z + x \land 0 \leq x < z \land z = y \times 2^n \land n \geq 0)\)

7. \{I_0 \land \neg z \leq x\}
   \quad q := 0 ; r := x
   \quad \{I_1\} \quad \text{(RAS,6)}

8. \((I_1 \land n \neq 0) \Rightarrow (x = (q \times 2) \times (z \div 2) + r \land 0 \leq r < (z \div 2) \times 2 \land z \div 2 = y \times 2^{n-1} \land n - 1 \geq 0)\)

9. \{I_1 \land n \neq 0\}
   \quad n := n - 1 ; z := z \div 2 ; q := q \times 2
   \quad \{x = q \times z + r \land 0 \leq r < z \times 2 \land z = y \times 2^n \land n \geq 0\} \quad \text{(RAS,8)}

10. \((x = q \times z + r \land 0 \leq r < z \times 2 \land z = y \times 2^n \land n \geq 0 \land z \leq r) \Rightarrow (x = (q + 1) \times z + (r - z) \land 0 \leq r - z < z \land z = y \times 2^n \land n \geq 0)\)

11. \{x = q \times z + r \land 0 \leq r < z \times 2 \land z = y \times 2^n \land n \geq 0 \land z \leq r\}
   \quad q := q + 1 ; r := r - z
   \quad \{I_1\} \quad \text{(RAS,10)}
12. \((x = q \times z + r \land 0 \leq r < z \times 2 \land z = y \times 2^n \land n \geq 0 \land -z \leq r) \Rightarrow I_1\)

13. \(\{x = q \times z + r \land 0 \leq r < z \times 2 \land z = y \times 2^n \land n \geq 0 \land -z \leq r\}\)

\(\text{skip}\)
\(\{I_1\}\) (ISK,12)

14. \(\{x = q \times z + r \land 0 \leq r < z \times 2 \land z = y \times 2^n \land n \geq 0\}\)

if \(z \leq r\) then \(q := q + 1 ; r := r - z\) else \(\text{skip}\)
\(\{I_1\}\) (CD,11,13)

15. \(\{I_1 \land n \neq 0\}\)

\(n := n - 1 ; z := z \div 2 ; q := q \times 2\);
if \(z \leq r\) then \(q := q + 1 ; r := r - z\) else \(\text{skip}\)
\(\{I_1\}\) (MSQ,9,14)

16. \(\{I_1\}\)

while \(n \neq 0\) do
\(\left(n := n - 1 ; z := z \div 2 ; q := q \times 2 ;
\quad \text{if } z \leq r \text{ then } q := q + 1 ; r := r - z \text{ else } \text{skip}\right)\)
\(\{I_1 \land \neg n \neq 0\}\) (WH,15)
17. \((I_1 \land \neg n \neq 0) \Rightarrow (x = q \times y + r \land 0 \leq r < y)\)

18. \(\{x \geq 0 \land y > 0\}\)
   \[n := 0 ; z := y ;\]
   \[\text{while } z \leq x \text{ do } (n := n + 1 ; z = z \times 2) ;\]
   \[q := 0 ; r := x ;\]
   \[\text{while } n \neq 0 \text{ do }\]
   \[(n := n - 1 ; z := z \div 2 ; q := q \times 2 ;\]
   \[\text{if } z \leq r \text{ then } q := q + 1 ; r := r - z \text{ else skip})\]
   \[\{x = q \times y + r \land 0 \leq r < y\}\]
   (MSQ,2,5,7,15,17)

19. \(\{x \geq 0 \land y > 0\}\)
   \[n := 0 ; \text{newvar } z := y \text{ in}\]
   \[(\text{while } z \leq x \text{ do } (n := n + 1 ; z = z \times 2) ;\]
   \[q := 0 ; r := x ;\]
   \[\text{while } n \neq 0 \text{ do }\]
   \[(n := n - 1 ; z := z \div 2 ; q := q \times 2 ;\]
   \[\text{if } z \leq r \text{ then } q := q + 1 ; r := r - z \text{ else skip})\]
   \[\{x = q \times y + r \land 0 \leq r < y\}\]
   (DC,18)
20. \( \{ x \geq 0 \land y > 0 \} \)

newvar \( n := 0 \) in newvar \( z := y \) in

( while \( z \leq x \) do ( \( n := n + 1 \); \( z := z \times 2 \) );
q := 0; r := x;
while \( n \neq 0 \) do

( \( n := n - 1 \); \( z := z \div 2 \); \( q := q \times 2 \);
if \( z \leq r \) then \( q := q + 1 \); \( r := r - z \) else skip)
)

\( \{ x = q \times y + r \land 0 \leq r < y \} \) \hfill (DC,19)
An Annotated Specification for Fast Division

\{x \geq 0 \land y > 0\}

newvar n := 0 in newvar z := y in
\((\{z = y \times 2^n \land n \geq 0 \land x \geq 0 \land y > 0\})\)

while z \leq x do (n := n + 1 ; z = z \times 2) ;
q := 0 ; r := x ;
\{x = q \times z + r \land 0 \leq r < z \land z = y \times 2^n \land n \geq 0\}

while n \neq 0 do
\((n := n - 1 ; z := z \div 2 ; q := q \times 2 ;
\{x = q \times z + r \land 0 \leq r < z \times 2 \land z = y \times 2^n \land n \geq 0\})
\text{if } z \leq r \text{ then } q := q + 1 ; r := r - z \text{ else skip}\))
\{x = q \times y + r \land 0 \leq r < y\}
An Example from Separation of Logic: Relative Pointers

\{\text{emp}\}
\begin{array}{l}
x := \text{cons}(a, a) ; \\
\{x \mapsto a, a\}
y := \text{cons}(b, b) ; \\
\{(x \mapsto a, a) \ast (y \mapsto b, b)\}
\{(x \mapsto a, \_ ) \ast (y \mapsto b, \_ )\}
\end{array}
\begin{array}{l}
[x + 1] := y - x ; \\
\{(x \mapsto a, y - x) \ast (y \mapsto b, \_ )\}
\end{array}
\begin{array}{l}
[y + 1] := x - y ; \\
\{(x \mapsto a, y - x) \ast (y \mapsto b, x - y)\}
\end{array}
\begin{array}{l}
\exists o. (x \mapsto a, o) \ast (x + o \mapsto b, - o)\}
\end{array}.
Another Example (O’Hearn): Concurrent Buffering

\{\text{emp}\}
\{\text{emp} \ast \text{emp}\}

\{\text{emp}\}
\text{cons}(\ldots, \ldots) ;
\{x \mapsto -, -\} \parallel
\text{put}(x) ;
\{\text{emp}\}

\{\text{emp}\}
\{\text{emp} \ast \text{emp}\}

\{\text{emp}\}
\text{get}(y) ;
\{y \mapsto -, -\}
\text{“Use } y\text{”} ;
\{y \mapsto -, -\}
\text{dispose } y ;
\{\text{emp}\}

\{\text{emp}\}
\{\text{emp}\}
\{\text{emp}\}
Annotation Descriptions

We will write the *annotation description*

\[ A \gg \{p\} c \{q\} \]

to indicate that \( A \) is an annotated specification proving the specification \( \{p\} c \{q\} \).

(The letter \( A \), with various decorations, will be a metavariable ranging over annotated specifications and their subphrases.)

We will define the valid annotation descriptions by means of inference rules.
A Surprise

Sometimes an annotated specification may contain fewer assertions than its unannotated version. For example, we will regard

\[
y := 2 \times y \{ y = 2^k \land k \leq n \}\]

as an annotated version of

\[
\{ 2 \times y = 2^k \land k \leq n \} y := 2 \times y \{ y = 2^k \land k \leq n \},
\]

and

\[
k := k + 1 ; y := 2 \times y \{ y = 2^k \land k \leq n \}\]

as an annotated version of

\[
\{ 2\times y = 2^{k+1} \land k+1 \leq n \} k := k+1 ; y := 2 \times y \{ y = 2^k \land k \leq n \}.\]

The main reason we can allow such incomplete specifications is that, in such cases, for the command \( c \) and postcondition \( q \), one can calculate a weakest (liberal) precondition \( p_w \), which is an assertion such that \( \{ p \} c \{ q \} \) holds just when \( p \Rightarrow p_w \). In many such cases, we will take \( p_w \) as an implicit precondition of the annotated specification.
Assignment (AS)

\[ v := e \{ q \} \implies \{ q/v \to e \} \quad v := e \{ q \}. \]

Instances

\[ y := 2 \times y \quad \{ y = 2^k \land k \leq n \} \implies \begin{cases} \{ 2 \times y = 2^k \land k \leq n \} \\ y := 2 \times y \\ \{ y = 2^k \land k \leq n \} \end{cases} \]

and

\[ k := k + 1 \quad \{ 2 \times y = 2^k \land k \leq n \} \implies \begin{cases} \{ 2 \times y = 2^{k+1} \land k + 1 \leq n \} \\ k := k + 1 \\ \{ 2 \times y = 2^k \land k \leq n \} \end{cases}. \]
Sequential Composition (SQ)

\[
A_1 \{q\} \gg \{p\} \ c_1 \{q\} \quad A_2 \gg \{q\} \ c_2 \{r\}
\]
\[
A_1 \ ; \ A_2 \gg \{p\} \ c_1 \ ; \ c_2 \{r\}.
\]

For instance,

\[
k := k + 1 \\
\{2 \times y = 2^k \land k \leq n\}
\]
\[
\gg
\begin{cases}
\{2 \times y = 2^{k+1} \land k + 1 \leq n\} \\
k := k + 1 \\
\{2 \times y = 2^k \land k \leq n\}
\end{cases}
\]
\[
y := 2 \times y \\
\{y = 2^k \land k \leq n\}
\]
\[
\gg
\begin{cases}
\{2 \times y = 2^k \land k \leq n\} \\
y := 2 \times y \\
\{y = 2^k \land k \leq n\}
\end{cases}
\]

\[
k := k + 1 \ ; \ y := 2 \times y \\
\{y = 2^k \land k \leq n\}
\]
\[
\gg
\begin{cases}
\{2 \times y = 2^{k+1} \land k + 1 \leq n\} \\
k := k + 1 \ ; \ y := 2 \times y \\
\{y = 2^k \land k \leq n\}
\end{cases}.
\]
Strengthening Precedent (SP)

\[ p \Rightarrow q \quad A \gg \{q\} \ c \ {r}\]

\[ \{p\} \ A \gg \{p\} \ c \ {r}. \]

For instance,

\[
(y = 2^k \land k \leq n \land k \neq n) \Rightarrow (2 \times y = 2^{k+1} \land k + 1 \leq n)
\]

\[
k := k + 1; y := 2 \times y
\]

\[
\{y = 2^k \land k \leq n\}
\]

\[
\gg
\]

\[
\{2 \times y = 2^{k+1} \land k + 1 \leq n\}
\]

\[
k := k + 1; y := 2 \times y
\]

\[
\{y = 2^k \land k \leq n\}
\]

\[
\}
\]

\[
\gg
\]

\[
\{y = 2^k \land k \leq n \land k \neq n\}
\]

\[
k := k + 1; y := 2 \times y
\]

\[
\{y = 2^k \land k \leq n\}
\]

\[
\gg
\]

\[
\{y = 2^k \land k \leq n \land k \neq n\}
\]

\[
k := k + 1; y := 2 \times y
\]

\[
\{y = 2^k \land k \leq n\}.
\]
Why Do We Ever Need Intermediate Assertions?

1. **while** commands and calls of recursive procedures do not always have weakest preconditions that can be expressed in our assertion language.

2. Certain structural inference rules, such as the existential quantifier rule (or the frame rule), do not fit well into the framework of weakest assertions.

3. Intermediate assertions are often needed to simplify verification conditions.
Partial Correctness of while (WH)

\[
\{i \land b\} A \{i\} \gg \{i \land b\} c \{i\}
\]
\[
\{i\} \text{ while } b \text{ do } A \gg \{i\} \text{ while } b \text{ do } c \{i \land \neg b\}.
\]

For instance,

\[
\begin{align*}
\{y = 2^k \land k \leq n \land k \neq n\} \\
k := k + 1; y := 2 \times y \\
\{y = 2^k \land k \leq n\}
\end{align*}
\]
\[
\gg
\begin{align*}
\{y = 2^k \land k \leq n \land k \neq n\} \\
k := k + 1; y := 2 \times y \\
\{y = 2^k \land k \leq n\}
\end{align*}
\]

\[
\begin{align*}
\{y = 2^k \land k \leq n\} \\
\text{while } k \neq n \text{ do} \\
(k := k + 1; y := 2 \times y)
\end{align*}
\]
\[
\gg
\begin{align*}
\{y = 2^k \land k \leq n\} \\
\text{while } k \neq n \text{ do} \\
(k := k + 1; y := 2 \times y) \\
\{y = 2^k \land k \leq n \land \neg k \neq n\}
\end{align*}
\]
Weakening Consequent (WC)

\[
\frac{A \Rightarrow \{p\} \quad c \quad \{q\} \quad q \Rightarrow r}{A \{r\} \Rightarrow \{p\} \quad c \quad \{r\}}.
\]

For instance,

\[
\begin{align*}
\{y = 2^k \land k \leq n\} \\
\text{while } k \neq n \text{ do} \\
(k := k + 1 \ ; \ y := 2 \times y) \end{align*} \quad \Rightarrow \quad \begin{align*}
\{y = 2^k \land k \leq n\} \\
\text{while } k \neq n \text{ do} \\
(k := k + 1 \ ; \ y := 2 \times y) \end{align*}
\]

\[
y = 2^k \land k \leq n \land k \neq n \Rightarrow y = 2^n
\]
Alternative Axiom Schema (Assignment)

• Backward Reasoning (Hoare)

\[ \mathcal{A}\{q\} \implies \{g(q)\} \ c \ \{q\} \]

e.g. \[ v := e \{q\} \implies \{q/v \to e\} \ v := e \{q\} \]

• Forward Reasoning (Floyd)

\[ \{p\} \mathcal{A} \implies \{p\} \ c \ \{g(p)\} \]

e.g. \[ \{p\} v := e \implies \{p\} \ v := e \ \{\exists v'. v = e' \land p'\}, \]

where \( v' \not\in \{v\} \cup \text{FV}(e) \cup \text{FV}(p) \), \( e' \) is \( e/v \to v' \), and \( p' \) is \( p/v \to v' \). The quantifier can be omitted when \( v \) does not occur in \( e \) or \( p \).
Completeness

We say that an annotated specification is *right-complete* if it ends with a postcondition, *left-complete* if it begins with a precondition, and *complete* if it is both right- and left-complete. Then the patterns

\[
\begin{align*}
\mathcal{A} \\
\mathcal{A}\{q\} \\
\{p\}\mathcal{A} \\
\{p\}\mathcal{A}\{q\}
\end{align*}
\]

will match any annotated specification. right-complete annotated specification. left-complete annotated specification. complete annotated specification.

Inclusions of Patterns

\[
\begin{array}{c}
\mathcal{A} \quad \text{(Outward)} \\
\mathcal{A}\{q\} \quad \text{(Backward)} \\
\{p\}\mathcal{A} \quad \text{(Forward)} \\
\{p\}\mathcal{A}\{q\} \quad \text{(Inward)}
\end{array}
\]

\[
\begin{align*}
\mathcal{A} &\subseteq \mathcal{A}\{q\} & \supseteq \{p\}\mathcal{A} \\
\mathcal{A}\{q\} &\supseteq \{p\}\mathcal{A}\{q\} & \subseteq \{p\}\mathcal{A}
\end{align*}
\]
Forcing Completeness

We can force any specification to be left-complete by strengthening its precedent with the vacuous verification condition \( p \Rightarrow p \):

\[
\begin{align*}
\frac{p \Rightarrow p}{\{p\} \mathcal{A} \gg \{p\} c \{r\}}
\end{align*}
\]

For example,

\[
2 \times y = 2^k \Rightarrow 2 \times y = 2^k
\]

\[
\begin{align*}
y &:= 2 \times y \{y = 2^k\} \gg \{2 \times y = 2^k\} \\
y &:= 2 \times y \{y = 2^k\}
\end{align*}
\]

\[
\{2 \times y = 2^k\} y := 2 \times y \{y = 2^k\} \gg
\]

\[
\{2 \times y = 2^k\} y := 2 \times y \{y = 2^k\}.
\]

Similarly, we can force any specification to be right-complete by weakening its consequent with \( q \Rightarrow q \).
Alternative Rules with Premisses (Conditionals)

- **Backward-Preserving Reasoning** (for weakest preconditions)

  \[
  \mathcal{A}_1\{q\} \gg \{p_1\} \ c_1 \ \{q\} \\
  \mathcal{A}_2\{q\} \gg \{p_2\} \ c_2 \ \{q\}
  \]

  if \(b\) then \(\mathcal{A}_1\) else \((\mathcal{A}_2)\) \(\{q\} \gg \{(b \Rightarrow p_1) \land (\neg b \Rightarrow p_2)\}\) if \(b\) then \(c_1\) else \(c_2\) \{q\}

- **Forward-Preserving Reasoning**

  \[
  \{p \land b\} \mathcal{A}_1 \gg \{p \land b\} \ c_1 \ \{q_1\} \\
  \{p \land \neg b\} \mathcal{A}_2 \gg \{p \land \neg b\} \ c_2 \ \{q_2\}
  \]

  \{p\} if \(b\) then \(\mathcal{A}_1\) else \((\mathcal{A}_2)\) \(\gg \{p\}\) if \(b\) then \(c_1\) else \(c_2\) \{\(q_1 \lor q_2\)\}

- **Inward-Preserving Reasoning** (my preference)

  \[
  \{p \land b\} \mathcal{A}_1 \{q\} \gg \{p \land b\} \ c_1 \ \{q\} \\
  \{p \land \neg b\} \mathcal{A}_2 \{q\} \gg \{p \land \neg b\} \ c_2 \ \{q\}
  \]

  \{p\} if \(b\) then \(\mathcal{A}_1\) else \((\mathcal{A}_2)\) \(\{q\} \gg \{p\}\) if \(b\) then \(c_1\) else \(c_2\) \{q\}

- **Outward-Preserving Reasoning**

  \[
  \mathcal{A}_1 \gg \{p_1\} \ c_1 \ \{q_1\} \\
  \mathcal{A}_2 \gg \{p_2\} \ c_2 \ \{q_2\}
  \]

  if \(b\) then \(\mathcal{A}_1\) else \((\mathcal{A}_2)\) \(\gg \{(b \Rightarrow p_1) \land (\neg b \Rightarrow p_2)\}\) if \(b\) then \(c_1\) else \(c_2\) \{\(q_1 \lor q_2\)\}
Variable Declaration (DC) (Inward-Preserving)

\[
\begin{align*}
\{p\} A \{q\} & \Rightarrow \{p\} c \{q\} \\
\{p\} \text{ newvar } v \text{ in } A \{q\} & \Rightarrow \{p\} \text{ newvar } v \text{ in } c \{q\},
\end{align*}
\]

when \(v\) does not occur free in \(p\) or \(q\).

Skip (SK) (Backward)

\[
\text{skip } \{q\} \Rightarrow \{q\} \text{ skip } \{q\}.
\]
More Structural Rules

In the following structural rules:

- In the unary rules a right brace is used to indicate the extent of the single operand.
- In the binary rules the two operands are placed symmetrically around the indicator DISJ, CONJ, or $\parallel$.

(We assume that the annotated specifications in the premisses will often be sequences of several lines.)
Vacuity (VAC) (Backward)

\[
\text{true} \{q\} \Rightarrow \{\text{false}\} \ c \ \{q\}.
\]

Here \( c \) contains no annotations, since no reasoning about its subcommands is used. For example, using (VAC), (SP), (WH), and (WC):

\[
\{s = 0 \land a - 1 \geq b \land k \geq b\}
\]

\textbf{while} \( k < b \) \textbf{do}

\[
(k := k + 1; \quad s := s + k) \quad \} \ VAC
\]

\[
\{s = 0 \land a - 1 \geq b\}.
\]
Disjunction (DISJ) (Backward-Preserving)

\[
\frac{A_1 \{q\} \gg \{p_1\} c \{q\} \quad A_2 \{q\} \gg \{p_2\} c \{q\}}{(A_1 \text{ DISJ } A_2) \{q\} \gg \{p_1 \lor p_2\} c \{q\}}.
\]

For example,

\{
\text{true}
\}

\{a - 1 \leq b\}

\begin{align*}
s & := 0 ; k := a - 1 ; \\
\{s = \sum_{i=a}^{k} i \land k \leq b\}
\end{align*}

while \(k < b\) do

\begin{align*}
(k & := k + 1 ; s := s + k) \\
\{s = \sum_{i=a}^{b} i\}.
\end{align*}
Conjunction (CONJ)

- Forward-Preserving

\[
\begin{align*}
\{p\} A_1 \gg \{p\} c \{q_1\} & \quad \{p\} A_2 \gg \{p\} c \{q_2\} \\
\{p\} (A_1 \text{ CONJ } A_2) \gg \{p\} c \{q_1 \land q_2\},
\end{align*}
\]

- Outward-Preserving (Better?)

\[
\begin{align*}
A_1 \gg \{p_1\} c \{q_1\} & \quad A_2 \gg \{p_2\} c \{q_2\} \\
(A_1 \text{ CONJ } A_2) \gg \{p_1 \land p_2\} c \{q_1 \land q_2\}.
\end{align*}
\]
Concurrency (CONC) (Outward-Preserving)

- In Hoare Logic:
  \[
  \begin{align*}
  A_1 & \gg \{p_1\} \ c_1 \ \{q_1\} \quad A_2 & \gg \{p_2\} \ c_2 \ \{q_2\} \\
  (A_1 \parallel A_2) & \gg \{p_1 \land p_2\} \ c_1 \parallel c_2 \ \{q_1 \land q_2\},
  \end{align*}
  \]
  where no variable occurring free in \(p_1, c_1,\) or \(q_1\) is modified by \(c_2,\) and vice-versa.

The above rule is unsound for separation logic; instead we have:

- In Separation Logic (O’Hearn)
  \[
  \begin{align*}
  A_1 & \gg \{p_1\} \ c_1 \ \{q_1\} \quad A_2 & \gg \{p_2\} \ c_2 \ \{q_2\} \\
  (A_1 \parallel A_2) & \gg \{p_1 \ast p_2\} \ c_1 \parallel c_2 \ \{q_1 \ast q_2\},
  \end{align*}
  \]
  where no variable occurring free in \(p_1, c_1,\) or \(q_1\) is modified by \(c_2,\) and vice-versa.
Existential Quantification (EQ) (Outward-Preserving)

\[ \mathcal{A} \gg \{p\} \ c \ \{q\} \]
\[ \mathcal{A} \} \ \exists v \gg \{\exists v. \ p\} \ c \ \{\exists v. \ q\} , \]
where \( v \) is not free in \( c \).

Universal Quantification (UQ) (Outward-Preserving)

\[ \mathcal{A} \gg \{p\} \ c \ \{q\} \]
\[ \mathcal{A} \} \ \forall v \gg \{\forall v. \ p\} \ c \ \{\forall v. \ q\} , \]
where \( v \) is not free in \( c \).
Constancy (CONST) (Outward-Preserving)

\[ \mathcal{A} \gg \{p\} \ c \ \{q\} \quad \mathcal{A} \}
\wedge r \gg \{p \wedge r\} \ c \ \{q \wedge r\}, \]

where no variable occurring free in \( r \) is modified by \( c \).

(In separation logic, \( r \) must be pure.)

An Example

\[
\begin{align*}
\{f = \text{fib}(k) \wedge g = \text{fib}(k-1) \wedge k < n\} \\
\{f = \text{fib}(k) \wedge g = \text{fib}(k-1)\} \\
t := g ; \\
g := f ; \\
f := g + t ; \\
\{f = \text{fib}(k + 1) \wedge g = \text{fib}(k)\} \\
k := k + 1 \\
\{f = \text{fib}(k) \wedge g = \text{fib}(k - 1) \wedge k \leq n\}
\end{align*}
\]
Frame (FR) (O’Hearn) (Outward-Preserving)

\[
\begin{align*}
\mathcal{A} & \mathrel{\Rightarrow} \{p\} c \{q\} \\
\mathcal{A} \ast r & \mathrel{\Rightarrow} \{p \ast r\} c \{q \ast r\},
\end{align*}
\]

where no variable occurring free in \( r \) is modified by \( c \).

An Example

\[
\begin{align*}
\{\exists j. \ x \mapsto -, j \ast \text{list } \alpha \ j\} \\
\{x \mapsto -\} \\
[x] \assign a & \ast x + 1 \mapsto j \ast \text{list } \alpha \ j \} \\
\{x \mapsto a\} \\
\{\exists j. \ x \mapsto a, j \ast \text{list } \alpha \ j\}
\end{align*}
\]
Substitution (SUB) (Outward-Preserving)

\[ \mathcal{A} \gg \{p\} \ c \ \{q\} \]

\[ \mathcal{A}/\delta \gg \{p/\delta\} \ (c/\delta) \ \{q/\delta\}, \]

where \( \delta \) is the substitution \( v_1 \rightarrow e_1, \ldots, v_n \rightarrow e_n; \) \( v_1, \ldots, v_n \) are the variables occurring free in \( p, c, \) or \( q, \) and, if \( v_i \) is modified by \( c, \) then \( e_i \) is a variable that does not occur free in any other \( e_j. \)

In the conclusion of this rule, \( \mathcal{A}/\delta \) denotes an annotated specification in which “/” and the substitution denoted by \( \delta \) occur literally, i.e., the substitution is not carried out on \( \mathcal{A}. \)
An Example

In

\[ \{ x = y \} x := x + y \{ x = 2 \times y \}, \]

one can substitute \( x \to z, y \to 2 \times w - 1 \) to infer

\[ \{ z = 2 \times w - 1 \} z := z + 2 \times w - 1 \{ z = 2 \times (2 \times w - 1) \}. \]

But one cannot substitute \( x \to z, y \to 2 \times z - 1 \) to infer the invalid

\[ \{ z = 2 \times z - 1 \} z := z + 2 \times z - 1 \{ z = 2 \times (2 \times z - 1) \}. \]
Simple Procedures

By “simple” procedures, we mean that the following restrictions are imposed:

- Parameters are variables and expressions, not commands or procedure names.
- There are no “global” variables: All free variables of the procedure body must be formal parameters of the procedure.
- Procedures are proper, i.e., their calls are commands.
- Calls are restricted to prevent aliasing.

An additional peculiarity, which substantially simplifies reasoning about simple procedures, is that we syntactically distinguish parameters that may be modified from those that may not be.
Procedure Definitions

A simple nonrecursive (or recursive) procedure definition is a command of the form

\[
\text{let } h(v_1, \ldots, v_m; v'_1, \ldots, v'_n) = c \text{ in } c',
\]

\[
\text{letrec } h(v_1, \ldots, v_m; v'_1, \ldots, v'_n) = c \text{ in } c',
\]

where

- \( h \) is a binding occurrence of a procedure name, whose scope is \( c' \) (or \( c \) and \( c' \) in the recursive case).
- \( c \) and \( c' \) are commands.
- \( v_1, \ldots, v_m; v'_1, \ldots, v'_n \) is a list of distinct variables, called formal parameters, that includes all of the free variables of \( c \). The formal parameters are binding occurrences whose scope is \( c \).
- \( v_1, \ldots, v_m \) includes all of the variables modified by \( c \).
Procedure Calls

A *procedure call* is a command of the form

\[ h(w_1, \ldots, w_m; e'_1, \ldots, e'_n), \]

where

- \( h \) is a procedure name.
- \( w_1, \ldots, w_m \) and \( e'_1, \ldots, e'_n \) are called *actual parameters*.
- \( w_1, \ldots, w_m \) are distinct variables.
- \( e'_1, \ldots, e'_n \) are expressions that do not contain occurrences of the variables \( w_1, \ldots, w_m \).
- The free variables of the procedure call are

\[
\text{FV}(h(w_1, \ldots, w_m; e'_1, \ldots, e'_n)) = \{w_1, \ldots, w_m\} \cup \text{FV}(e'_1) \cup \cdots \cup \text{FV}(e'_n)
\]

and the variables modified by the call are \( w_1, \ldots, w_m \).
Hypothetical Specifications

The truth of a specification $\{p\} c \{q\}$ will depend upon an environment, which maps the procedure names occurring free in $c$ into their meanings.

We define a hypothetical specification to have the form

$$\Gamma \vdash \{p\} c \{q\},$$

where the context $\Gamma$ is a sequence of specifications of the form

$$\{p_0\} c_0 \{q_0\}, \ldots, \{p_{n-1}\} c_{n-1} \{q_{n-1}\}.$$

We say that such a hypothetical specification is true iff $\{p\} c \{q\}$ holds for every environment in which all of the specifications in $\Gamma$ hold.
Generalizing Old Inference Rules

For example,

- Strengthening Precedent (SP)

\[
\begin{array}{c}
p \Rightarrow q \\
\Gamma \vdash \{q\} \ c \ \{r\}
\end{array}
\]
\[
\Gamma \vdash \{p\} \ c \ \{r\},
\]

- Substitution (SUB)

\[
\begin{array}{c}
\Gamma \vdash \{p\} \ c \ \{q\}
\end{array}
\]
\[
\Gamma \vdash \{p/\delta\} \ (c/\delta) \ \{q/\delta\},
\]

where \(\delta\) is the substitution \(v_1 \rightarrow e_1, \ldots, v_n \rightarrow e_n; v_1, \ldots, v_n\) are the variables occurring free in \(p, c,\) or \(q,\) and, if \(v_i\) is modified by \(c,\) then \(e_i\) is a variable that does not occur free in any other \(e_j.\)

Note that substitutions do not affect procedure names.
Rules for Procedures

- Hypothesis (HYPO)

  \[
  \Gamma, \{p\} \ c \ \{q\}, \Gamma' \vdash \{p\} \ c \ \{q\}.
  \]

- Simple Procedures (SPROC)

  \[
  \Gamma \vdash \{p\} \ c \ \{q\}
  \Gamma, \{p\} \ h(v_1, \ldots, v_m; v'_1, \ldots, v'_n) \ \{q\} \vdash \{p'\} \ c' \ \{q'\}
  \]

  \[
  \Gamma \vdash \{p'\} \ \text{let} \ h(v_1, \ldots, v_m; v'_1, \ldots, v'_n) = c \ \text{in} \ c' \ \{q'\},
  \]

  where \( h \) does not occur free in any triple of \( \Gamma \).

- Simple Recursive Procedures (SRPROC)

  (partial correctness only)

  \[
  \Gamma, \{p\} \ h(v_1, \ldots, v_m; v'_1, \ldots, v'_n) \ \{q\} \vdash \{p\} \ c \ \{q\}
  \Gamma, \{p\} \ h(v_1, \ldots, v_m; v'_1, \ldots, v'_n) \ \{q\} \vdash \{p'\} \ c' \ \{q'\}
  \]

  \[
  \Gamma \vdash \{p'\} \ \text{letrec} \ h(v_1, \ldots, v_m; v'_1, \ldots, v'_n) = c \ \text{in} \ c' \ \{q'\},
  \]

  where \( h \) does not occur free in any triple of \( \Gamma \).
Some Limitations

To keep our exposition straightforward, we have ignored:

- Simultaneous recursion,
- Multiple hypotheses for the same procedure.
Two Derived Rules

From (HYPO):

- Call (CALL)

\[
\Gamma, \{p\} \ h(v_1, \ldots, v_m; v'_1, \ldots, v'_n) \ \{q\}, \ \Gamma' \vdash \\
\{p\} \ h(v_1, \ldots, v_m; v'_1, \ldots, v'_n) \ \{q\}.
\]

and from (CALL) and (SUB):

- General Call (GCALL)

\[
\Gamma, \{p\} \ h(v_1, \ldots, v_m; v'_1, \ldots, v'_n) \ \{q\}, \ \Gamma' \vdash \\
\{p/\delta\} \ h(w_1, \ldots, w_m; e'_1, \ldots, e'_n) \ \{q/\delta\},
\]

where \(\delta\) is a substitution

\[
\delta = v_1 \rightarrow w_1, \ldots, v_m \rightarrow w_m,
\]

\[
v'_1 \rightarrow e'_1, \ldots, v'_n \rightarrow e'_n,
\]

\[
v''_1 \rightarrow e''_1, \ldots, v''_k \rightarrow e''_k,
\]

which acts on all the free variables in

\[
\{p\} \ h(v_1, \ldots, v_m; v'_1, \ldots, v'_n) \ \{q\},
\]

and none of the variables \(w_1, \ldots, w_m\) occur free in the expressions \(e'_1, \ldots, e'_n\) or \(e''_1, \ldots, e''_k\).
Annotated Specifications: Ghosts

In (GCALL):

\[
\Gamma, \{p\} \vdash h(v_1, \ldots, v_m; v'_1, \ldots, v'_n) \{q\}, \Gamma' \vdash \{p/\delta\} h(w_1, \ldots, w_m; e'_1, \ldots, e'_n) \{q/\delta\},
\]

where \(\delta\) is a substitution

\[
\delta = v_1 \rightarrow w_1, \ldots, v_m \rightarrow w_m,
\]

\[
v'_1 \rightarrow e'_1, \ldots, v'_n \rightarrow e'_n,
\]

\[
v''_1 \rightarrow e''_1, \ldots, v''_k \rightarrow e''_k,
\]

which acts on . . .

there may be ghost variables \(v''_1, \ldots, v''_k\) that appear in \(\delta\) but are not formal parameters.

We will treat \(v''_1, \ldots, v''_k\) as formal ghost parameters, and \(e''_1, \ldots, e''_k\) as actual ghost parameters.
For example,

\[
\begin{align*}
\{n \geq 0 \land r = r_0\} & \quad \vdash \quad \{n - 1 \geq 0 \land r = n \times r_0\} \\
\text{multfact}(r; n) \quad \{r = n! \times r_0\} & \quad \vdash \quad \text{multfact}(r; n - 1) \quad \{r = (n - 1)! \times n \times r_0\}
\end{align*}
\]

is an instance of (GCALL) using the substitution

\[r \rightarrow r, \ n \rightarrow n - 1, \ r_0 \rightarrow n \times r_0.\]

The corresponding annotated specification will be

\[
\begin{align*}
\{n \geq 0 \land r = r_0\} & \quad \vdash \quad \{n - 1 \geq 0 \land r = n \times r_0\} \\
\text{multfact}(r; n) \quad \{r = n! \times r_0\} & \quad \vdash \quad \text{multfact}(r; n - 1) \quad \{r = (n - 1)! \times n \times r_0\}
\end{align*}
\]
Generalizing Annotation Descriptions

An *annotated context* is a sequence of *annotated hypotheses*, which have the form

\[
\{p\} \ h(v_1, \ldots, v_m; v'_1, \ldots, v'_n) \{v''_1, \ldots, v''_k\} \ {q},
\]

where \(v''_1, \ldots, v''_k\) is a list of formal ghost parameters (and all of the formal parameters, including the ghosts, are distinct).

We write \(\hat{\Gamma}\) to denote an annotated context, and \(\Gamma\) to denote the corresponding ordinary context that is obtained by erasing the lists of ghost formal parameters. Then an annotation description has the form:

\[
\hat{\Gamma} \vdash A \gg \{p\} \ c \ {q},
\]

meaning that \(\hat{\Gamma} \vdash A\) is an annotated hypothetical specification proving the hypothetical specification \(\Gamma \vdash \{p\} \ c \ {q}\).
Rules for Procedural Annotated Specifications

- General Call (GCALL) (Outward)

\[
\Gamma', \{p\} \ h(v_1, \ldots, v_m; v'_1, \ldots, v'_n)\{v''_1, \ldots, v''_k\} \ \{q\}, \Gamma' \vdash \\
h(w_1, \ldots, w_m; e'_1, \ldots, e'_n)\{e''_1, \ldots, e''_k\} \gg \\
\{p/\delta\} \ h(w_1, \ldots, w_m; e'_1, \ldots, e'_n) \ \{q/\delta\},
\]

where \(\delta\) is the substitution

\[
\delta = v_1 \rightarrow w_1, \ldots, v_m \rightarrow w_m, \\
v'_1 \rightarrow e'_1, \ldots, v'_n \rightarrow e'_n, \\
v''_1 \rightarrow e''_1, \ldots, v''_k \rightarrow e''_k,
\]

which acts on all the free variables in

\[
\{p\} \ h(v_1, \ldots, v_m; v'_1, \ldots, v'_n) \ \{q\},
\]

and \(w_1, \ldots, w_m\) are distinct variables that do not occur free in the expressions \(e'_1, \ldots, e'_n\) or \(e''_1, \ldots, e''_k\).
Simple Procedures (SPROC)

\[ \Gamma \vdash \{p\} \ A \ \{q\} \gg \{p\} \ c \ \{q\} \]

\[ \Gamma, \{p\} \ h(v_1, \ldots, v_m; v'_1, \ldots, v'_n)\{v''_1, \ldots, v''_k\} \ \{q\} \vdash \]

\[ A' \gg \{p'\} \ c' \ \{q'\} \]

\[ \Gamma \vdash \text{let } h(v_1, \ldots, v_m; v'_1, \ldots, v'_n)\{v''_1, \ldots, v''_k\} = \]

\[ \{p\} \ A \ \{q\} \ \text{in} \ A' \]

\[ \gg \{p'\} \ \text{let } h(v_1, \ldots, v_m; v'_1, \ldots, v'_n) = c \ \text{in} \ c' \ \{q'\}, \]

where \( h \) does not occur free in any triple of \( \Gamma \).
Simple Recursive Procedures (SRPROC)

\[
\begin{align*}
\hat{\Gamma}, \{p\} \ h(v_1, \ldots, v_m; v'_1, \ldots, v'_n) \{v''_1, \ldots, v''_k\} \ {q} \vdash \\
\{p\} \ A \ {q} \gg \{p\} \ c \ {q} \\
\hat{\Gamma}, \{p\} \ h(v_1, \ldots, v_m; v'_1, \ldots, v'_n) \{v''_1, \ldots, v''_k\} \ {q} \vdash \\
A' \gg \{p'\} \ c' \ {q'}
\end{align*}
\]

\[
\hat{\Gamma} \vdash \text{letrec} \ h(v_1, \ldots, v_m; v'_1, \ldots, v'_n) \{v''_1, \ldots, v''_k\} = \\
A \text{ in } \{p'\} \ A' \ {q'} \\
\gg \{p'\} \text{ letrec} \ h(v_1, \ldots, v_m; v'_1, \ldots, v'_n) = c \text{ in } c' \ {q'},
\]

where \( h \) does not occur free in any triple of \( \hat{\Gamma} \).
An Example

\{z = 10\}
\letrec \text{multfact}(r; n)\{r_0\} =
\{n \geq 0 \land r = r_0\}
\text{if } n = 0 \text{ then}
\{n = 0 \land r = r_0\} \text{ skip } \{r = n! \times r_0\}
\text{else}
\{n - 1 \geq 0 \land n \times r = n \times r_0\}
\quad r := n \times r ;
\quad \{n - 1 \geq 0 \land r = n \times r_0\}
\quad \{n - 1 \geq 0 \land r = n \times r_0\}
\quad \{r = (n - 1)! \times n \times r_0\}
\quad \{n - 1 \geq 0 \land r = (n - 1)! \times n \times r_0\}
\quad \{r = n! \times r_0\}
\text{in}
\{5 \geq 0 \land z = 10\}
\text{multfact}(z; 5)\{10\}
\{z = 5! \times 10\}
How the Annotations Determine a Formal Proof

The application of (SRPROC) to the letrec definition gives rise to the hypothesis

$$\{ n \geq 0 \land r = r_0 \} \ \text{multfact}(r; n) \{ r_0 \} \ \{ r = n! \times r_0 \}.$$ 

By (GCALL), the hypothesis entails

$$\{ n - 1 \geq 0 \land r = n \times r_0 \} \ \text{multfact}(r; n - 1) \{ n \times r_0 \} \ \{ r = (n - 1)! \times n \times r_0 \}.$$ 

Next, since $n$ is not modified by the call $\text{multfact}(r; n - 1)$, the rule of constancy gives

$$\{ n - 1 \geq 0 \land r = n \times r_0 \land n - 1 \geq 0 \} \ \text{multfact}(r; n - 1) \{ n \times r_0 \} \ \{ r = (n - 1)! \times n \times r_0 \land n - 1 \geq 0 \}.$$ 

Then, we can strengthen the precondition to obtain

$$\{ n - 1 \geq 0 \land r = n \times r_0 \} \ \text{multfact}(r; n - 1) \{ n \times r_0 \} \ \{ r = (n - 1)! \times n \times r_0 \land n - 1 \geq 0 \}.$$ 

Also, by (GCALL), the hypothesis entails

$$\{ 5 \geq 0 \land z = 10 \} \ \text{multfact}(z; 5) \{ 10 \} \ \{ z = 5! \times 10 \}.$$
Why Annotated Specifications Work

Proofs of Specifications

\[ \Phi \]

Proofs of Annotation Descriptions

\[ \text{concl} \]

Annotation Descriptions

\[ \text{erase-spec} \]

Annotated Specifications

\[ \Psi \]

Proofs of Specifications

• \( \sim \) relates proofs that contain the same verification conditions.

• \( \sim \) relates proofs that contain the same verification conditions, except for vacuous conditions of the form \( p \Rightarrow p \).

This proof depends upon the fact that we have not permitted more than one rule for any particular command type or structural situation.
Annotated Specifications for the Heap Commands

- **Mutation (MUBR) (Backward)**
  \[
  [e] := e' \{p\} \implies ((e \mapsto -) * ((e \mapsto e') \mapsto p)) \quad [e] := e' \{p\}.
  \]

- **Disposal (DISBR) (Backward)**
  \[
  \text{dispose } e \{r\} \implies \{(e \mapsto -) * r\} \quad \text{dispose } e \{r\}.
  \]

- **Allocation (CONSBR) (Backward)**
  \[
  v := \text{cons}(e) \{p\} \implies \{\forall v''. (v'' \mapsto e) \mapsto p''\} \quad v := \text{cons}(e) \{p\},
  \]
  where \(v''\) is distinct from \(v\), \(v'' \notin \text{FV}(e, p)\), and \(p''\) denotes \(p/v \mapsto v''\).

- **Lookup (LKBR1) (Backward)**
  \[
  v := [e] \{p\} \implies \{\exists v''. (e \mapsto v'') * ((e \mapsto v'') \mapsto p'')\} \quad v := [e] \{p\},
  \]
  where \(v'' \notin \text{FV}(e) \cup (\text{FV}(p) - \{v\})\), and \(p''\) denotes \(p/v \mapsto v''\).
Deriving Local and Global Rules

By taking $p$ in (MUBR) to be $e \mapsto e'$, and using the valid verification condition

$$VC = (e \mapsto -) \Rightarrow (e \mapsto -) \ast ((e \mapsto e') \ast (e \mapsto e'))$$

we may use (SP) to obtain a proof:

\[
\begin{align*}
[e] & := e' \{e \mapsto e'\} \gg \\
VC & \{ (e \mapsto -) \ast ((e \mapsto e') \ast (e \mapsto e')) \} [e] := e' \{e \mapsto e'\} \\
& \{e \mapsto -\} [e] := e' \{e \mapsto e'\} \gg \{e \mapsto -\} [e] := e' \{e \mapsto e'\}
\end{align*}
\]

of an annotation description corresponding to the local form (MUL). In such a manner, one may derive local and global rules of the form

$$\{p\} c \{q\} \gg \{p\} c \{q\}.$$