Textual Programs: Syntax

We will now consider programs presented in textual form.

Besides declaration of types and variables, our simple language allows the following statements:

• **Assignment** – \( \vec{y} := \vec{E}(\vec{y}) \), where \( \vec{y} \) is a list of variables, and \( \vec{E} \) is a list of type-compatible expressions over the program variables. The statement **skip** can be introduced as an abbreviation for the assignment \( y := y \).

• **Concatenation** – If \( S_1 \) and \( S_2 \) are statements, then so is their **concatenation** \( S_1; S_2 \).

• **Conditional** – If \( S_1 \) and \( S_2 \) are statements and \( c \) is a boolean expression, then **if** \( c \) **then** \( S_1 \) **else** \( S_2 \) is a **conditional statement**. The one-branch conditional **if** \( c \) **then** \( S \) is an abbreviation for **if** \( c \) **then** \( S \) **else** **skip**.

• **While** – If \( S \) is a statement and \( c \) is a boolean expression, then **while** \( c \) **do** \( S \) is a **while statement**.
Example: INT-MULTIPLY

Following is an example of a textual program INT-MULTIPLY which multiplies two natural numbers \( x_1 \) and \( x_2 \) and returns their result in \( z \), using only multiplication and integer divisions by 2.

\[
\begin{align*}
(u_1, u_2, z) := (x_1, x_2, 0); \\
\textbf{while } u_1 \neq 0 \textbf{ do} \\
\quad \textbf{if } \text{odd}(u_1) \textbf{ then} \\
\quad \quad (u_1, z) := (u_1 - 1, z + u_2); \\
\quad \quad (u_1, u_2) := (u_1 \div 2, 2u_2)
\end{align*}
\]
Textual Programs: Semantics

Rather than directly defining the semantics of a textual program by describing the computations it can generate, we present a translation from a textual program into flow-graph programs. Each statement $S$ will be translated into a flow-graph

$$\ell_0 \quad S \quad \ell_t$$

The translation is defined inductively:

- The flow-graph corresponding to the assignment $\vec{y} := \vec{E}$ is given by:

  $$\ell_0 \quad \vec{y} := \vec{E} \quad \ell_t$$

- Assume that the flow-graphs corresponding to statements $S_1$ and $S_2$ are given by

  $$\ell_1 \quad S_1 \quad \ell_2 \quad \text{and} \quad \ell_3 \quad S_2 \quad \ell_4$$

  respectively. Then the flow-graph for $S_1; S_2$ will be given by

  $$\ell_1 \quad S_1 \quad \ell_2 \quad S_2 \quad \ell_4$$

  where we have identified nodes $\ell_2$ and $\ell_3$. 
Textual → Flow-graphs Translation Continued

- Assume that the flow-graph for statements $S_1$ and $S_2$ are given as before, by two corresponding flow-graphs. Then the flow-graph for if $c$ then $S_1$ else $S_2$ is given by:

- Assume that the flow-graph for statement $S$ is given by

Then the flow-graph for while $c$ do $S$ is given by:

where we have identified nodes $l_2$ and $l_4$. 

where we have identified nodes $l_2$ and $l$. 

Example: Program INT-MULTIPLY

Reconsider program INT-MULTIPLY:

\[
\begin{align*}
\ell_0: & \quad (u_1, u_2, z) := (x_1, x_2, 0); \\
\ell_1: & \quad \textbf{while} \ u_1 \neq 0 \ \textbf{do} \\
& \quad \begin{cases} \\
\ell_2: & \quad \textbf{if} \ \text{odd}(u_1) \ \textbf{then} \\
\ell_3: & \quad (u_1, z) := (u_1 - 1, z + u_2); \\
\ell_4: & \quad (u_1, u_2) := (u_1 \div 2, 2u_2) \\
\end{cases} \\
\ell_5: & \quad \\
\end{align*}
\]

Its flow-graph equivalent is given by:

[Diagram of flow graph showing the logic flow between \(\ell_0\) to \(\ell_5\).]

Sequential Program Analysis, NYU, Spring, 2003
Annotated Programs

To recreate the notion of an assertion network attached to locations, we introduce the notion of an annotated program. We begin by defining annotated statements as follows:

- The triple \( \{p\} y := e \{q\} \) is an annotated statement.

- If \( \{p\} S_1 \{q\} \) and \( \{q\} S_2 \{r\} \) are annotated statements, then so is \( \{p\} S_1 \{q\} S_2 \{r\} \).

- If \( \{p_1\} S_1 \{q\} \) and \( \{p_2\} S_2 \{q\} \) are annotated statements, then so is \( \{p\} \text{if } c \text{ then } \{p_1\} S_1 \text{ else } \{p_2\} S_2 \{q\} \).

- If \( \{p_1\} S \{p\} \) is an annotated statement, then so is \( \{p\} \text{while } c \text{ do } \{p_1\} S \{q\} \).

Finally, if \( P \) is a program whose body is the statement \( S \), and \( \{p\} S \{q\} \) is an annotated version of \( S \), then \( \{p\} S \{q\} \) is an annotated program.

For example, the following is an annotated program:

\[
\begin{align*}
\{p_0\} (u_1, u_2, z) & := (x_1, x_2, 0) \\
\{p_1\} \text{while } u_1 \neq 0 \text{ do} & \\
& \left[ \begin{array}{l}
\{p_2\} \text{if } \text{odd}(u_1) \text{ then} \\
& \{p_3\} (u_1, z) := (u_1 - 1, z + u_2) \\
& \{p_4\} (u_1, u_2) := (u_1 \div 2, 2u_2) \\
\{p_5\}
\end{array} \right]
\end{align*}
\]

where \( p_0, \ldots, p_5 \) are assertions.
Verification Conditions

Let $P$ be an annotated program. Each such program generates a set of verification conditions, as follows:

- Each annotated assignment $\{p\} \bar{y} := \bar{e} \{q\}$ generates the verification condition:

  $$Ver(\{p\} \bar{y} := \bar{e} \{q\}) = \{p \rightarrow q[\bar{e}/\bar{y}]\}$$

- Each annotated concatenation $\{p\}S_1\{q\}S_2\{r\}$ generates the verification conditions:

  $$Ver(\{p\}S_1\{q\}S_2\{r\}) = Ver(\{p\}S_1\{q\}) \cup Ver(\{q\}S_2\{r\})$$

- Each annotated conditional $\{p\}[\text{if } c \text{ then } \{p_1\}S_1 \text{ else } \{p_2\}S_2]\{q\}$ generates the verification conditions:

  $$Ver(\{p\}[\text{if } c \text{ then } \{p_1\}S_1 \text{ else } \{p_2\}S_2]\{q\}) =\$$

  $$\{p \wedge c \rightarrow p_1, \ p \wedge \neg c \rightarrow p_2\} \cup Ver(\{p_1\}S_1\{q\}) \cup Ver(\{p_2\}S_2\{q\})$$

- Each annotated while statement $\{p\}[\text{while } c \text{ do } \{p_1\}S]\{q\}$ generates the verification conditions:

  $$Ver(\{p\}[\text{while } c \text{ do } \{p_1\}S]\{q\}) =\$$

  $$\{p \wedge c \rightarrow p_1, \ p \wedge \neg c \rightarrow q\} \cup Ver(\{p_1\}S\{p\})$$
Example: Program INT-MULTIPLY

Reconsider the annotated program:

\begin{align*}
\{p_0\}(u_1, u_2, z) &:= (x_1, x_2, 0) \\
\{p_1\} &\textbf{while } u_1 \neq 0 \textbf{ do} \\
&\left[\begin{array}{l}
\{p_2\} \textbf{if } \text{odd}(u_1) \textbf{ then} \\
\{p_3\}(u_1, z) &:= (u_1 - 1, z + u_2) \\
\{p_4\}(u_1, u_2) &:= (u_1 \div 2, 2u_2)
\end{array}\right] \\
\{p_5\}
\end{align*}

This program gives rise to the following verification conditions:

\begin{align*}
\forall_0 : & \quad p_0 \quad \rightarrow \quad p_1[x_1,x_2,0/u_1,u_2,z] \\
\forall_1^{+} : & \quad p_1 \land u_1 \neq 0 \quad \rightarrow \quad p_2 \\
\forall_2^{+} : & \quad p_2 \land \text{odd}(u_1) \quad \rightarrow \quad p_3 \\
\forall_3 : & \quad p_3 \quad \rightarrow \quad p_4[u_1-1,z+u_2/u_1,z] \\
\forall_4 : & \quad p_4 \quad \rightarrow \quad p_1[u_1\div2,2u_2/u_1,u_2]
\end{align*}
The Method of Inductive Assertions for Annotated Textual Programs

We are now ready to formulate the method of inductive assertions applied to textual programs.

Claim 23. [Inductive Assertions for Textual Programs]
Let $\{p\}P\{q\}$ be an annotated program. If all the verification conditions for this program are valid, then $P$ is partially correct w.r.t $(p, q)$

The claim can be proven by reduction to flow-graphs and showing that we obtain an inductive full assertion network.

An annotated program all of whose verification conditions are valid is called a valid annotated program.
Apply to \textsc{INT-MULTIPLY}

Considering program \textsc{INT-MULTIPLY}, we wish to establish its partial correctness w.r.t \((\text{true}, z = x_1 \cdot x_2)\). Let \(\varphi = \varphi(u_1, u_2, z) : x_1 \cdot x_2 = z + u_1 \cdot u_2\). We propose the following annotated program:

\[
\begin{align*}
\{l_0: \text{true}\} (u_1, u_2, z) &:= (x_1, x_2, 0) \\
\{l_1: \varphi\} \text{while } u_1 \neq 0 \text{ do} &
\begin{cases}
\{l_2: \varphi\} \text{if odd}(u_1) \text{ then} &
\begin{cases}
\{l_3: \varphi \land \text{odd}(u_1)\} (u_1, z) &:= (u_1 - 1, z + u_2) \\
\{l_4: \varphi \land \text{even}(u_1)\} (u_1, u_2) &:= (u_1 \div 2, 2u_2)
\end{cases} \\
\{l_5: z = x_1 \cdot x_2\}
\end{cases}
\end{align*}
\]

The generated verification conditions are:

\[
\begin{align*}
V_0 : \text{true} & \quad \rightarrow \quad x_1 \cdot x_2 = 0 + x_1 \cdot x_2 \\
\varphi & \\
V_1^+ : \varphi \land u_1 \neq 0 & \quad \rightarrow \quad \varphi \\
V_1^- : x_1 \cdot x_2 = z + u_1 \cdot u_2 \land u_1 = 0 & \quad \rightarrow \quad z = x_1 \cdot x_2 \\
V_2^+ : \varphi \land \text{odd}(u_1) & \quad \rightarrow \quad \varphi \land \text{odd}(u_1) \\
V_2^- : \varphi \land \neg \text{odd}(u_1) & \quad \rightarrow \quad \varphi \land \text{even}(u_1) \\
V_3 : x_1 \cdot x_2 = z + u_1 \cdot u_2 \land \text{odd}(u_1) & \quad \rightarrow \quad x_1 \cdot x_2 = (z + u_2) + (u_1 - 1) \cdot u_2 \land \text{even}(u_1 - 1) \\
V_4 : x_1 \cdot x_2 = z + u_1 \cdot u_2 \land \text{even}(u_1) & \quad \rightarrow \quad x_1 \cdot x_2 = z + (u_1 \div 2) \cdot (2u_2)
\end{align*}
\]

which are all valid.
**Hoare Logic**

Following the work of C.A.R. Hoare from 1969, we introduce a **Hoare triplet** \( \{p\} S \{q\} \) whose intending meaning is that the statement \( S \) is partially correct w.r.t. \( \langle p, q \rangle \). We then introduce a list of inference rules as follows:

### Rule \text{ASSGN}

\[
p \rightarrow q[\vec{e}/\vec{y}]
\]

\[
\{p\} \bar{y} := \bar{e} \{q\}
\]

A rule dealing with concatenation:

### Rule \text{CONC}

\[
\{p\} S_1 \{q\}, \quad \{q\} S_2 \{r\}
\]

\[
\{p\} S_1; S_2 \{r\}
\]

A rule for conditional statements:

### Rule \text{COND}

\[
p \land c \rightarrow p_1, \quad p \land \neg c \rightarrow p_2, \quad \{p_1\} S_1 \{q\}, \quad \{p_2\} S_2 \{q\}
\]

\[
\{p\} \left[ \text{if } c \text{ then } S_1 \text{ else } S_2 \right] \{q\}
\]

A rule for a **while** statement:

### Rule \text{WHILE}

\[
p \land c \rightarrow q, \quad p \land \neg c \rightarrow r, \quad \{q\} S \{p\}
\]

\[
\{p\} \left[ \text{while } c \text{ do } S \right] \{r\}
\]

Finally, the **consequence** rule:

### Rule \text{CONS}

\[
p \rightarrow q, \quad r \rightarrow u, \quad \{q\} S \{r\}
\]

\[
\{p\} S \{u\}
\]
Example: Apply to \texttt{INT-MULTIPLY}

As illustration, we prove \{true\} \texttt{INT-MULTIPLY} \{z = x_1 \cdot x_2\} for the program:

\[
\begin{array}{l}
\{l_0\}: \text{true}\} (u_1, u_2, z) := (x_1, x_2, 0) \\
\{l_1\}: \psi \text{ while } u_1 \neq 0 \text{ do} \\
\quad \begin{cases} \\
\{l_2\}: \psi \land u_1 \neq 0 \text{ if } \text{odd}(u_1) \text{ then } \\
\quad \{l_3\}: \psi \land \text{odd}(u_1) \} (u_1, z) := (u_1 - 1, z + u_2) \\
\{l_4\}: \psi \land \text{even}(u_1) \} (u_1, u_2) := (u_1 \div 2, 2u_2) \\
\{l_5\}: z = x_1 \cdot x_2 \\
\end{cases}
\end{array}
\]

1. \[x_1 \cdot x_2 = z + u_1 \cdot u_2 \land \text{odd}(u_1) \rightarrow \\
   x_1 \cdot x_2 = (z + u_2) + (u_1 - 1) \cdot u_2 \land \text{even}(u_1 - 1) \] Logic
2. \( \{x_1 \cdot x_2 = z + u_1 \cdot u_2 \land \text{odd}(u_1)\} l_3 \{x_1 \cdot x_2 = z + u_1 \cdot u_2 \land \text{even}(u_1)\} \) ASSGN 1.
3. \( \psi \land u_1 \neq 0 \land \text{odd}(u_1) \rightarrow \psi \land \text{odd}(u_1) \) Logic
4. \( \psi \land u_1 \neq 0 \land \neg \text{odd}(u_1) \rightarrow \psi \land \text{even}(u_1) \) Logic
5. \( \{\psi \land u_1 \neq 0\} l_2 \{\psi \land \text{even}(u_1)\} \) COND 3,4,2.
6. \( x_1 \cdot x_2 = z + u_1 \cdot u_2 \land \text{even}(u_1) \rightarrow x_1 \cdot x_2 = z + (u_1 \div 2) \cdot (2u_2) \) Logic
7. \( \{\psi \land \text{even}(u_1)\} l_4 \{\psi\} \) ASSGN 6.
8. \( \{\psi \land u_1 \neq 0\} l_2; l_4 \{\psi\} \) CONC 5, 7.
9. \( \psi \land u_1 \neq 0 \rightarrow \psi \) Logic
10. \( x_1 \cdot x_2 = z + u_1 \cdot u_2 \land \neg (u_1 \neq 0) \rightarrow z = x_1 \cdot x_2 \) WHILE 9, 10, 8.
11. \( \{\psi\} l_1 \{z = x_1 \cdot x_2\} \) Logic
12. \( \text{true} \rightarrow x_1 \cdot x_2 = 0 + x_1 \cdot x_2 \) ASSGN 12.
13. \( \{\text{true}\} l_0 \{\psi\} \) CONC 13, 11.
Relation Between the Approaches: From Valid Annotation to Hoare Proof

Up to now we have identified two different approaches to proving partial correctness of textual programs: Valid annotation and Hoare logic. We will now explore the relation between these two approaches. We write $\vdash_{\mathcal{H}}\{p\} S \{q\}$ to denote the existence of a Hoare style proof of $\{p\} S \{q\}$.

Claim 24. [From Valid annotation to Hoare proof] 
If $\{p\} S \{q\}$ is a valid annotation, then $\vdash_{\mathcal{H}}\{p\} S \{q\}$

Proof:
The claim is proved by induction on the size of the annotated statement $\{p\} S \{q\}$, and by cases on the type of $S$

- For the case that $S = [y := e]$, validity of the annotation $\{p\} y := e \{q\}$ implies validity of the implication $p \rightarrow q[e/e]$. We can then construct the following proof:
  1. $p \rightarrow u[e/y]$ Logic
  2. $\{p\} y := e \{u\}$ Rule ASSGN 1.

- For the case that $S = S_1; S_2$, a valid annotation assumes the form $\{p\} S_1 \{q\} S_2 \{r\}$, which implies the validity of the annotations $\{p\} S_1 \{q\}$ and $\{q\} S_2 \{r\}$. We can then construct the following proof:
  1. $\vdash_{\mathcal{H}}\{p\} S_1 \{q\}$ Induction hypothesis applied to $\{p\} S_1 \{q\}$.
  2. $\vdash_{\mathcal{H}}\{q\} S_2 \{r\}$ Induction hypothesis applied to $\{q\} S_2 \{r\}$.
  3. $\{p\} S_1; S_2 \{u\}$ Rule CONC 1, 2.
Proof of Claim 24 Continued

- For the case that $S = \text{if } c \text{ then } S_1 \text{ else } S_2$, a valid annotation assumes the form $\{p\} \text{if } c \text{ then } \{p_1\} S_1 \text{ else } \{p_2\} S_2 \{q\}$, which implies the validity of the annotations $\{p_1\} S_1 \{q\}$, $\{p_2\} S_2 \{q\}$, and the implications $p \land c \rightarrow p_1$, $p \land \neg c \rightarrow p_2$. We can then construct the following proof:

1. $\vdash_{\mathcal{H}} \{p_1\} S_1 \{q\}$ Induction hypothesis applied to $\{p_1\} S_1 \{q\}$.
2. $\vdash_{\mathcal{H}} \{p_2\} S_2 \{q\}$ Induction hypothesis applied to $\{p_2\} S_2 \{q\}$.
3. $p \land c \rightarrow p_1$ Logic.
4. $p \land \neg c \rightarrow p_2$ Logic.
5. $\{p\} [\text{if } c \text{ then } S_1 \text{ else } S_2] \{q\}$ Rule COND 1, 2, 3, 4.

- For the case that $S = \text{while } c \text{ then } S_1$, a valid annotation assumes the form $\{p\} [\text{while } c \text{ do } \{p_1\} S_1] \{q\}$, which implies the validity of the annotation $\{p_1\} S_1 \{p\}$, and the implications $p \land c \rightarrow p_1$, $p \land \neg c \rightarrow q$. We can then construct the following proof:

1. $\vdash_{\mathcal{H}} \{p_1\} S_1 \{p\}$ Induction hypothesis applied to $\{p_1\} S_1 \{p\}$.
2. $p \land c \rightarrow p_1$ Logic.
3. $p \land \neg c \rightarrow q$ Logic.
4. $\{p\} [\text{while } c \text{ do } S_1] \{q\}$ Rule WHILE 1, 2, 3.

Note that the produced Hoare logic proof does not use the consequence rule.
From Hoare Logic to Annotated Programs

The other direction, transforming a Hoare logic proof into an annotated program is less straightforward. This is mainly due to the consequence rule which does not have a direct analogue in the annotated programs context. Note that the consequence rule contains two parts, a part allowing weakening of the post-condition and a part which allows strengthening the pre-condition. The first part can be accommodated within annotated programs as is claimed in the following:

Lemma 25. [Weakening a post-condition]
If $q \rightarrow r$ and $\{p\} S \{q\}$ is a valid annotation, then so is $\{p\} S \{r\}$.

Proof:
Inspecting all verification conditions which can be generated by the annotated statement $\{p\} S \{q\}$, we can show that all occurrences of $q$ are on the right-hand side of an implication. Therefore, if $q$ implies $r$, the verification conditions obtained by replacing $q$ by $r$ will also be valid.

Unfortunately, we do not have an analogue of Lemma 25 when we consider strengthening the pre-condition. It is possible to have a valid annotation $\{q\} S \{r\}$ and an assertion $p$, such that $p \rightarrow q$, yet the annotation $\{p\} S \{r\}$ is not valid. To illustrate this, consider the valid annotation $\{x \geq 0\} [\text{while } x \leq 10 \text{ do } \{x \geq 0\} x := x + 1] \{x > 10\}$ and the assertion $x = 0$ which, obviously, implies $x \geq 0$. Observe that the annotation $\{x = 0\} [\text{while } x \leq 10 \text{ do } \{x \geq 0\} x := x + 1] \{x > 10\}$ is not valid. This is because the verification condition $x \geq 0 \rightarrow x + 1 = 0$ is not valid.
The Transformation

In spite of the previously identified difficulties, there exists a transformation from a Hoare logic proof to a valid annotation.

**Claim 26. [From Hoare proof to valid annotation]**

If \( \vdash_H \{ p \} S \{ q \} \), then there exists a valid annotation \( \{ t \} \overline{S} \{ q \} \) such that \( p \rightarrow t \).

**Proof:**

The claim is proved by induction on the length of the proof \( \vdash_H \{ p \} S \{ q \} \) and by separately considering the various cases of the proof rule that was applied in the last step of the proof.

- For the case that the last rule applied was ASSGN, the proved statement must be \( \{ p \} \overline{y} := \overline{e} \{ q \} \), and the implication \( p \rightarrow q[\overline{e}/\overline{y}] \) must have been proven in a previous step of the proof. In that case, we take \( t = p \) and observe that \( \{ p \} \overline{y} := \overline{e} \{ q \} \) is a valid annotation.

- For the case that the last rule applied was CONC, the proved statement is of the form \( \{ p \} S_1 \{ q \} S_2 \{ r \} \), and the proof contains (shorter) subproofs of \( \{ p \} S_1 \{ q \} \) and \( \{ q \} S_2 \{ r \} \). Applying the induction hypothesis to these two proofs, we obtain the valid annotations \( \{ t_1 \} \overline{S_1} \{ q \} \) and \( \{ t_2 \} \overline{S_2} \{ r \} \), such that \( p \rightarrow t_1 \) and \( q \rightarrow t_2 \). According to Lemma 25, \( \{ t_1 \} \overline{S_1} \{ t_2 \} \) is also a valid annotation. It follows that \( \{ t_1 \} \overline{S_1} \{ t_2 \} \overline{S_2} \{ r \} \) is a valid annotation and \( p \rightarrow t_1 \), as required by the claim.
Proof of Claim 26 Continued

- For the case that the last rule applied was $\text{COND}$, the proved statement
  is of the form $\{p\} [\text{if } c \text{ then } S_1 \text{ else } S_2] \{q\}$, and the proof contains (shorter)
  subproofs of $\{p_1\} S_1 \{q\}$ and $\{p_2\} S_2 \{q\}$ and the implications
  $p \land c \rightarrow p_1$ and
  $p \land \neg c \rightarrow p_2$. Applying the induction hypothesis to these two subproofs, we
  obtain the valid annotations $\{t_1\} \overline{S_1} \{q\}$ and $\{t_2\} \overline{S_2} \{q\}$, such that
  $p_1 \rightarrow t_1$ and
  $p_2 \rightarrow t_2$. Combining the four implications, we obtain $p \land c \rightarrow t_1$ and $p \land \neg c \rightarrow t_2$,
  from which we can deduce that $\{p\} [\text{if } c \text{ then } \{t_1\} \overline{S_1} \text{ else } \{t_2\} \overline{S_2}] \{r\}$ is a valid
  annotation. Taking $t = p$, this satisfies the requirement of the claim.

- For the case that the last rule applied was $\text{WHILE}$, the proved statement is of
  the form $\{p\} [\text{while } c \text{ do } S_1] \{q\}$, and the proof contains a (shorter) subproof of
  $\{p_1\} S_1 \{p\}$ and the implications $p \land c \rightarrow p_1$ and $p \land \neg c \rightarrow q$. Applying the
  induction hypothesis to this subproof, we obtain the valid annotation $\{t_1\} \overline{S_1} \{p\}$,
  such that $p_1 \rightarrow t_1$. Combining these implications, we obtain $p \land c \rightarrow t_1$ and
  $p \land \neg c \rightarrow q$, from which we can deduce that $\{p\} [\text{while } c \text{ do } \{t_1\} \overline{S_1}] \{r\}$ is a valid
  annotation. Taking $t = p$, this satisfies the requirement of the claim.

- For the case that the last rule applied was $\text{CONC}$, the proved statement is of
  the form $\{p\} S \{u\}$, and the proof contains a (shorter) subproof of $\{q\} S \{r\}$ and
  the implications $p \rightarrow q$ and $r \rightarrow u$. Applying the induction hypothesis to this
  subproof, we obtain the valid annotation $\{t\} \overline{S} \{r\}$, such that $q \rightarrow t$. Combining
  $p \rightarrow q$ and $q \rightarrow t$, we obtain $p \rightarrow t$. By Lemma 25 and $r \rightarrow u$, $\{t\} \overline{S} \{u\}$ is a valid
  annotation. Since $p \rightarrow t$, this satisfies the requirement of the claim.  

Sequential Program Analysis, NYU, Spring, 2003
Initialized Programs

All the programs we have considered so far have a special structure which enables a more faithful transformation. A program $P$ is called an initialized program if its body has the form $S = y := e; S_1$.

**Claim 27.** [Proof transformation for initialized programs] If $S = y := e; S_1$ is the body of an initialized program and $\vdash_H \{p\} S \{q\}$, then there exists a valid annotation $\{p\} S \{q\}$.

**Proof:**
By Claim 26, there exists a valid annotation $\{t\} S \{q\}$ such that $p \rightarrow t$. Since $S = y := e; S_1$, we claim that also $\{p\} S \{q\}$ is a valid annotation. This is because, in all verification conditions generated by the annotation $\{t\} y := e; S_1 \{q\}$, all occurrences of $t$ are on the left-hand side of an implication. Since $p \rightarrow t$, replacing $t$ by $p$ yields another set of valid verification conditions.