Today’s Lecture

▶ Go over the midterm
▶ Review path conditions & related verification conditions
▶ VCs for total correctness
The formula is valid

<table>
<thead>
<tr>
<th>P</th>
<th>Q</th>
<th>R</th>
<th>$F_1 : P \land Q$</th>
<th>$F_2 : \neg F_1$</th>
<th>$\neg R$</th>
<th>$F_3 : \neg R \rightarrow Q$</th>
<th>$F_4 : R \rightarrow F_3$</th>
<th>$F_5 : F_2 \rightarrow F_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

If you assumed different associativity of $\rightarrow$, we didn’t take points off
The formula can be expressed as $\neg(\neg F_1 \wedge \neg F_2)$

So, we want to disprove $I \not\models F_1 \lor F_2 \leftrightarrow \neg(\neg F_1 \wedge \neg F_2)$

Recalling the rule for $\leftrightarrow$, there are two cases to consider

$I \not\models F \leftrightarrow G$

\[
\begin{array}{c}
I \models F \wedge \neg G \\
I \models \neg F \wedge G
\end{array}
\]
## Problem 2, First Case

<table>
<thead>
<tr>
<th>Step</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. ( I \neq F_1 \vee F_2 \leftrightarrow \neg (\neg F_1 \land \neg F_2) )</td>
<td>assumption</td>
</tr>
<tr>
<td>2a. ( I \models (F_1 \vee F_2) \land \neg \neg (\neg F_1 \land \neg F_2) )</td>
<td>1 and ( \leftrightarrow ), case a</td>
</tr>
<tr>
<td>3a. ( I \models F_1 \lor F_2 )</td>
<td>2a and ( \land )</td>
</tr>
<tr>
<td>4a. ( I \models \neg \neg (\neg F_1 \land \neg F_2) )</td>
<td>2a and ( \land )</td>
</tr>
<tr>
<td>5a. ( I \neq \neg (\neg F_1 \land \neg F_2) )</td>
<td>4a and ( \neg )</td>
</tr>
<tr>
<td>6a. ( I \models \neg F_1 \land \neg F_2 )</td>
<td>5a and ( \neg )</td>
</tr>
<tr>
<td>7a. ( I \models \neg F_1 )</td>
<td>6a and ( \land )</td>
</tr>
<tr>
<td>8a. ( I \models \neg F_1 )</td>
<td>6a and ( \land )</td>
</tr>
<tr>
<td>9a. ( I \neq F_1 )</td>
<td>7a and ( \neg )</td>
</tr>
<tr>
<td>10a. ( I \neq F_2 )</td>
<td>8a and ( \neg )</td>
</tr>
<tr>
<td>11aa. ( I \models F_1 )</td>
<td>3a, case a</td>
</tr>
<tr>
<td>12aa. ( \bot )</td>
<td>11aa and 9a</td>
</tr>
<tr>
<td>11ab. ( I \models F_2 )</td>
<td>3a, case b</td>
</tr>
<tr>
<td>12ab. ( \bot )</td>
<td>11ab and 10a</td>
</tr>
<tr>
<td>Step</td>
<td>Reason</td>
</tr>
<tr>
<td>------</td>
<td>--------</td>
</tr>
<tr>
<td>1.  ( I \neq F_1 \lor F_2 \iff \neg(\neg F_1 \land \neg F_2) )</td>
<td>assumption</td>
</tr>
<tr>
<td>2b.  ( I \models \neg(F_1 \lor F_2) \land \neg(\neg F_1 \land \neg F_2) )</td>
<td>1 and \iff, case b</td>
</tr>
<tr>
<td>3b.  ( I \models \neg(F_1 \lor F_2) )</td>
<td>2b and \land</td>
</tr>
<tr>
<td>4b.  ( I \models \neg(\neg F_1 \land \neg F_2) )</td>
<td>2b and \land</td>
</tr>
<tr>
<td>5b.  ( I \neq F_1 \lor F_2 )</td>
<td>3b and \neg</td>
</tr>
<tr>
<td>6b.  ( I \neq \neg F_1 \land \neg F_2 )</td>
<td>4b and \neg</td>
</tr>
<tr>
<td>7b.  ( I \neq F_1 )</td>
<td>5b and \lor</td>
</tr>
<tr>
<td>8b.  ( I \neq F_2 )</td>
<td>5b and \lor</td>
</tr>
<tr>
<td>8ba. ( I \neq \neg F_1 )</td>
<td>6b and \land, case a</td>
</tr>
<tr>
<td>9ba. ( I \models F_1 )</td>
<td>8ba and \neg</td>
</tr>
<tr>
<td>10ba. ( \bot )</td>
<td>7b and 9ba</td>
</tr>
<tr>
<td>8bb. ( I \neq \neg F_2 )</td>
<td>6b and \land, case b</td>
</tr>
<tr>
<td>9bb. ( I \models F_2 )</td>
<td>8bb and \neg</td>
</tr>
<tr>
<td>10bb. ( \bot )</td>
<td>8b and 9bb</td>
</tr>
</tbody>
</table>
Problem 3

\[(\neg P \lor Q \lor R) \land (\neg Q \lor R) \land (\neg Q \lor \neg R) \land (P \lor \neg Q \lor \neg R)\]

<table>
<thead>
<tr>
<th>Step</th>
<th>State</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(\emptyset)</td>
<td>init.</td>
</tr>
<tr>
<td>2</td>
<td>(Q^\circ)</td>
<td>decide (Q)</td>
</tr>
<tr>
<td>3</td>
<td>(Q^\circ, R)</td>
<td>propagate (\neg Q \lor R)</td>
</tr>
<tr>
<td>4</td>
<td>(\overline{Q})</td>
<td>confl. (\neg Q \lor \neg R); backtrack</td>
</tr>
<tr>
<td>5</td>
<td>(\overline{Q}, P^\circ)</td>
<td>decide (P)</td>
</tr>
<tr>
<td>6</td>
<td>(\overline{Q}, P, R)</td>
<td>propagate (\neg P \lor Q \lor R)</td>
</tr>
<tr>
<td>7</td>
<td>sat</td>
<td></td>
</tr>
</tbody>
</table>
Problem 4

\[(\forall j. a[j] = b(i < v)[j]) \land a[i] \neq v\]

This formula is in the fragment, because it is equivalent to:

\[(\forall j. \text{true} \rightarrow a[j] = b(i < v)[j]) \land a[i] \neq v\]

Recall, we said that extensionality was in the fragment:

\[\forall i. a[i] = b[i]\]

Both formulas have a trivial index guard
Problem 4

We showed an algorithm for removing quantifiers in lecture 8

1. $F_1$: Put $F$ in negation normal form
2. $F_2$: Remove all write terms from $F_1$
3. $F_3$: Remove existential quantifiers from $F_2$
4. Select a sufficient set of index terms $\mathcal{I}$
5. $F_4$: Transform $\forall$ into conjunction using $\mathcal{I}$

This gives us a quantifier-free $T_{\text{EUF}}$ formula

No points off if you skipped step 2 on the exam
First, we remove the write operation by applying:

\[
F[a \langle i \triangle left v \rangle]
\]

\[
F[a'] \land a'[i] = v \land (\forall j. j \neq i \rightarrow a[j] = a'[j])
\]

For fresh \(a'\)

This leaves us with:

\[
(\forall j. a[j] = b'[j]) \land a[i] \neq v \land b'[i] = v \land (\forall j. i \neq j \rightarrow b'[j] = b[j])
\]

We then need to find the index set

\[
I = \{ \lambda \} \cup \{ t \mid [t] \in F \text{ and } t \text{ is not } \forall \text{-quantified} \} \\
\cup \{ t \mid t \text{ is an evar in an index guard of } F \}
\]

So, \(I = \{ \lambda, i \}\)
Problem 4

Now the important step: remove $\forall$

\[ H[\forall i. F[i] \rightarrow G[i]] \]
\[ \frac{H[\bigwedge_{i \in I} (F[i] \rightarrow G[i])]}{} \]

This gives us:

\[
\left( \bigwedge_{j \in \{\lambda, i\}} a[j] = b'[j] \right) \land a[i] \neq v \land b'[i] = v \land \left( \bigwedge_{j \in \{\lambda, i\}} i \neq j \rightarrow b'[j] = b[j] \right) \land \lambda \neq i
\]

We remove the “big” conjunction:

\[
a[i] = b'[i] \land a[\lambda] = b'[\lambda] \land a[i] \neq v \land b'[i] = v \land i \neq i \rightarrow b'[i] = b[i] \\
\land i \neq \lambda \rightarrow b'[\lambda] = b[\lambda] \land \lambda \neq i
\]

And finally, simplify to:

\[
a[i] = b'[i] \land a[\lambda] = b'[\lambda] \land a[i] \neq v \land b'[i] = v \land b'[\lambda] = b[\lambda] \land \lambda \neq i
\]
The following formula is valid if and only if the translation is correct:

\[(v_1 = f(x_1, y_1) \land v_2 = f(x_2, y_2) \land z = g(v_1, v_2)) \rightarrow z = g(f(x_1, y_1), f(x_2, y_2))\]

Our formula is of the form \(\phi_1 \rightarrow \phi_2\)

We know that if \(\phi_1 \land \phi_2\) is sat, then so is \(\phi_1 \rightarrow \phi_2\)

So we find a satisfying interpretation for:

\[(v_1 = f(x_1, y_1) \land v_2 = f(x_2, y_2) \land z = g(v_1, v_2)) \land z = g(f(x_1, y_1), f(x_2, y_2))\]
Problem 5

Computing the congruence closure:

\[
\{ v_1, v_2, \{ x_1, x_2 \}, \{ y_1, y_2 \}, \{ z \}, \{ f(x_1, y_1) \}, \{ f(x_2, y_2) \}, \{ g(v_1, v_2) \}, \{ g(f(x_1, y_1), f(x_1, y_1)) \} \}
\]

\[
\{ v_1, f(x_1, y_1) \}, \{ v_2 \}, \{ x_1 \}, \{ x_2 \}, \{ y_1 \}, \{ y_2 \}, \{ z \}, \{ f(x_2, y_2) \}, \{ g(v_1, v_2) \}, \{ g(f(x_1, y_1), f(x_1, y_1)) \} \}
\]

\[
\{ v_1, f(x_1, y_1) \}, \{ v_2, f(x_2, y_2) \}, \{ x_1 \}, \{ x_2 \}, \{ y_1 \}, \{ y_2 \}, \{ z \}, \{ g(v_1, v_2) \}, \{ g(f(x_1, y_1), f(x_1, y_1)) \} \}
\]

\[
\{ v_1, f(x_1, y_1) \}, \{ v_2, f(x_2, y_2) \}, \{ x_1 \}, \{ x_2 \}, \{ y_1 \}, \{ y_2 \}, \{ z, g(v_1, v_2) \}, \{ g(f(x_1, y_1), f(x_1, y_1)) \} \}
\]

\[
\{ v_1, f(x_1, y_1) \}, \{ v_2, f(x_2, y_2) \}, \{ x_1 \}, \{ x_2 \}, \{ y_1 \}, \{ y_2 \}, \{ z, g(v_1, v_2), g(f(x_1, y_1), f(x_1, y_1)) \} \}
\]
We discussed the following definition for program equivalence:

\[ \forall \sigma, \sigma'. \langle c_1, \sigma \rangle \downarrow \sigma' \iff \langle c_2, \sigma \rangle \downarrow \sigma' \]

For part 2, we need to prove both directions:

\[ \forall \sigma, \sigma'. \langle c_1, \sigma \rangle \downarrow \sigma' \Rightarrow \langle c_2, \sigma \rangle \downarrow \sigma' \]
\[ \forall \sigma, \sigma'. \langle c_1, \sigma \rangle \downarrow \sigma' \Leftarrow \langle c_2, \sigma \rangle \downarrow \sigma' \]
The last rule applied in the derivation was either WhileTrue or WhileFalse.

If it was WhileFalse, then there must have been $T_1$ that led to:

$$
\frac{T_1}{\langle b, \sigma \rangle \downarrow_b \text{false}}
\frac{\text{WhileFalse}}{\langle \text{while } b \text{ do } c, \sigma \rangle \downarrow \sigma}
$$

So in this case, $\sigma = \sigma'$. Then we also know that:

$$
\frac{T_1}{\langle b, \sigma \rangle \downarrow_b \text{false}}
\frac{\text{WhileFalse}}{\langle \text{while } b \text{ do (while } b \text{ do } c), \sigma \rangle \downarrow \sigma}
$$
If $b$ evaluates to true (so WhileTrue applies), we know that:

\[
\text{WhileTrue} \quad \frac{T_1}{\langle b, \sigma \rangle \downarrow_b \text{true}} \quad \frac{T_2}{\langle c, \sigma \rangle \downarrow \sigma''} \quad \frac{T_3}{\langle \text{while } b \text{ do } c, \sigma'' \rangle \downarrow \sigma'} \quad \langle \text{while } b \text{ do } c, \sigma \rangle \downarrow \sigma'
\]

We also know that $\langle b, \sigma' \rangle \downarrow_b \text{false}$. So:

\[
\text{WhileTrue} \quad \frac{T_1}{\langle b, \sigma \rangle \downarrow_b \text{true}} \quad \frac{T_1 \quad T_2 \quad T_3}{\langle \text{while } b \text{ do } c, \sigma \rangle \downarrow \sigma'} \quad \frac{T_3}{\langle \text{while } b \text{ do } (\cdots), \sigma' \rangle \downarrow \sigma'} \quad \langle \text{while } b \text{ do } (\text{while } b \text{ do } c), \sigma \rangle \downarrow \sigma'
\]
The other direction is similar.

In the case of WhileFalse,

\[
\begin{align*}
T_1 & \quad \frac{\langle b, \sigma \rangle \downarrow_b \text{false}}{\langle \text{while } b \text{ do } (\text{while } b \text{ do } c), \sigma \rangle \downarrow \sigma} \\
T_1 & \quad \frac{\langle b, \sigma \rangle \downarrow_b \text{false}}{\langle \text{while } b \text{ do } c, \sigma \rangle \downarrow \sigma}
\end{align*}
\]

In the case of WhileTrue,

\[
\begin{align*}
T_1 & \quad \frac{\langle b, \sigma \rangle \downarrow_b \text{true}}{\langle \text{while } b \text{ do } (\text{while } b \text{ do } c), \sigma \rangle \downarrow \sigma'} \\
T_2 & \quad \frac{\langle b, \sigma' \rangle \downarrow_b \text{false}}{\langle \text{while } b \text{ do } c, \sigma' \rangle \downarrow \sigma'}
\end{align*}
\]

Leaving us to conclude that,

\[
T_2 \quad \frac{\langle \text{while } b \text{ do } c, \sigma \rangle \downarrow \sigma'}
\]
The postcondition we’re after is:

\[
(\forall i.0 \leq i < n \rightarrow a[i] \leq m) \land (\exists i.0 \leq i < n \land m = a[i])
\]

The loop invariant needed to prove this is:

\[
(\forall i.0 \leq i < k \rightarrow a[i] \leq m) \land (\exists i.0 \leq i \leq k \land m = a[i])
\]

Technically, you also need \( k \leq n \) and \( i < n \) in the invariant if arrays have bounds.
The loop invariant needed for this problem is: \( I : x = y + a \)

This leaves us with two additional verification conditions:
\[
I \land \neg(a \neq 0) \rightarrow x = y
\]
\[
I \land a \neq 0 \rightarrow \text{wlp}(y := y + 1; a := a - 1, x = y + a)
\]

Clearly,
\[
(x = y + a \land a = 0) \rightarrow x = y
\]

Then:
\[
\text{wlp}(c_3; c_4, x = y + a) = \text{wlp}(y := y + 1, \text{wlp}(a := a - 1, x = y + 1))
\]
\[
= \text{wlp}(y := y + 1, x = y + a - 1)
\]
\[
= (x = y + 1 + a - 1)
\]
\[
= (x = y + a)
\]

And we know that \((x = y + a \land a \neq 0) \rightarrow x = y + a\)
Then:
\[
\text{wlp}(\text{while } a \neq 0 \text{ do } y := y + 1; a := a - 1, x = y) = (x = y + a)
\]

We need to show that
\[
0 \leq x \Rightarrow \text{wlp}(a := x; y := 0, x = y + a)
\]

Computing the weakest precondition:
\[
\begin{align*}
\text{wlp}(a := x; y := 0, x = y + a) &= \text{wlp}(a := x, \text{wlp}(y := 0, x = y + a)) \\
&= \text{wlp}(a := x, x = 0 + a) \\
&= (x = 0 + x) \\
&\iff (x = x)
\end{align*}
\]

So \(x = x\) is equivalent to \text{true}, and \(0 \leq x \rightarrow \text{true}.\)
Now, a program is partially correct if for each procedure $P$:

1. Whenever $P$’s preconditions are satisfied on entry
2. $P$’s postconditions are satisfied on exit

We’ll extend the approach we’ve talked about so far:

1. Reduce the annotated program to a set of verification conditions
2. If all VCs are valid, then the program is correct

Our approach will be different:

- Use annotations to decompose the program into simpler parts
- Generate VC for each part in isolation, assuming each annotation holds
- Make sure that correctness of the whole follows from correctness of each part
A **basic path** is a sequence of instructions that:
- Begins at procedure precondition or loop invariant
- Ends at a loop invariant, assertion, or procedure postcondition
- Doesn’t cross loops: invariants only at beginning or end of path

Basic paths correspond to straight-line segments of code

Think of a Hoare triple over a sequence command:

$$\{P\} \; c_1; \; c_2; \; \cdots; \; c_n \; \{Q\}$$

$P, Q$ are pre-/postconditions, loop invariants, or assertion guards
**assume** statement (Review)

\[ \{ l \leq i \land \forall j. l \leq j < i \rightarrow a[j] \neq e \} \]

**while** \((i \leq u)\);

**if** \((a[i] = e)\);

\(rv := 1;\)

\(\{ (rv = 1) \iff (\exists i. l \leq i \leq u \land a[i] = e) \} \)

\[ \{ l \leq i \land \forall j. l \leq j < i \rightarrow a[j] \neq e \} \]

**assume** \(i \leq u;\)

**assume** \(a[i] = e;\)

\(rv := 1; \{ (rv = 1) \iff (\exists i. l \leq i \leq u \land a[i] = e) \} \)

Guarded statements introduce **assumptions** about environment.

We write these in basic paths using an **assume** statement.

**assume** \(b\) means:

1. Rest of path executed only if \(b\) is true in current environment
2. When reasoning about rest of path, we can assume \(b\) holds
Basic Paths: Example

```plaintext
proc LinearSearch(a : array, l, u, e)
    pre 0 ≤ l ∧ u < |a|
    post (rv = 1) ↔ (∃i.l ≤ i ≤ u ∧ a[i] = e)
{
    i := l;
    while (i ≤ u)
        {l ≤ i ∧ ∀j.l ≤ j < i → a[j] ≠ e}
        {if(a[i] = e) return 1;
         i := i + 1;
        }
    return 0;
}
```
Given a procedure $f$ with prototype

```
proc f(x₁, ..., xₙ)
  pre P[x₁, ..., xₙ]
  post Q[x₁, ..., xₙ, rv]
```

When $f$ is called in context $w := f(e₁, ..., eₙ)$;

Augment the calling context with an assertion:

$$\{P[e₁, ..., eₙ]\};$$
$$w := f(e₁/x₁, ..., eₙ/xₙ);$$

In paths that pass through the call,

1. Create fresh variable $v$ to hold the return value
2. Replace call with an assumption of the postcondition:

```
assume G[e₁/x₁, ..., eₙ/xₙ, v/rv]
```
Recall the first basic path:
\[
\begin{align*}
\{&0 \leq l \land u < |a| \}\ \\
i &:= l \\
\{&l \leq i \land \forall j. l \leq j < i \rightarrow a[j] \neq e \}
\end{align*}
\]

The VC for this is:
\[
0 \leq l \land u < |a| \rightarrow \text{wlp}(i := l, l \leq i \land \forall j. l \leq j < i \rightarrow a[j] \neq e)
\]

We have that:
\[
\begin{align*}
\text{wlp}(i := l, l \leq i \land \forall j. l \leq j < i \rightarrow a[j] \neq e) &
\iff l \leq l \land \forall j. l \leq j < l \rightarrow a[j] \neq e \\
&\iff \text{true} \land \forall j. \text{false} \rightarrow a[j] \neq e \\
&\iff \text{true}
\end{align*}
\]

Our final condition is valid:
\[
0 \leq l \land u < |a| \Rightarrow \text{true}
\]
Example: Linear Search (2)

Recall the second basic path:

\[
\begin{align*}
\{ P : l \leq i \land \forall j. l \leq j < i \rightarrow a[j] \neq e \} \\
c_1 : \text{assume } i \leq u; \\
c_2 : \text{assume } a[i] = e; \\
c_3 : rv := 1; \\
\{ Q : (rv = 1) \leftrightarrow \exists j. l \leq j \leq u \land a[j] = e \}
\end{align*}
\]

The VC for this is:

\[
P \rightarrow \text{wlp}(c_1, c_2, c_3, Q) \iff P \rightarrow \text{wlp}(c_1, \text{wlp}(c_2, \text{wlp}(c_3, Q)))
\]

We have that:

\[
\text{wlp}(rv := 1, (rv = 1) \leftrightarrow \exists j. l \leq j \leq u \land a[j] = e) \\
\iff (1 = 1) \leftrightarrow \exists j. l \leq j \leq u \land a[j] = e \\
\iff \exists j. l \leq j \leq u \land a[j] = e
\]

Our final condition is:

\[
0 \leq l \land u < |a| \Rightarrow \text{true}
\]
Example: Linear Search (3)

Recall the second basic path:

\[
\begin{align*}
\{P : l \leq i \land \forall j. l \leq j < i \rightarrow a[j] \neq e\} \\
c_1 : \text{assume} \ i \leq u; \\
c_2 : \text{assume} \ a[i] = e; \\
c_3 : rv := 1; \\
\{Q : (rv = 1) \leftrightarrow \exists j. l \leq j \leq u \land a[j] = e\}
\end{align*}
\]

The VC for this is:

\[P \rightarrow \text{wlp}(c_1; c_2; c_3, Q) \iff P \rightarrow \text{wlp}(c_1, \text{wlp}(c_2, \text{wlp}(c_3, Q)))\]

Moving on,

\[
\begin{align*}
\text{wlp}(c_1, \text{wlp}(\text{assume} \ a[i] = e, \exists j. l \leq j \leq u \land a[j] = e)) \\
\iff \text{wlp}(c_1, a[i] = e \rightarrow \exists j. l \leq j \leq u \land a[j] = e) \\
\iff \text{wlp}(\text{assume} \ i \leq u, a[i] = e \rightarrow \exists j. l \leq j \leq u \land a[j] = e) \\
\iff i \leq u \rightarrow (a[i] = e \rightarrow \exists j. l \leq j \leq u \land a[j] = e)
\end{align*}
\]

Our final condition is:

\[0 \leq l \land u < |a| \Rightarrow \text{true}\]
Our final condition is:

\[
(l \leq i \leq u \land \forall j. l \leq j < i \rightarrow a[j] \neq e) \rightarrow (a[i] = e \rightarrow \exists j. l \leq j \leq u \land a[j] = e)
\]

\[\Leftrightarrow (l \leq i \leq u \land a[i] = e \land \forall j. l \leq j < i \rightarrow a[j] \neq e) \rightarrow \exists j. l \leq j \leq u \land a[j] = e\]

Notice that:

\[l \leq i \leq u \land a[i] = e \rightarrow \exists j. l \leq j \leq u \land a[j] = e\]

is valid

So, the condition is valid, and the corresponding triple is as well
Partial correctness doesn’t require termination

**Total correctness** is a stronger statement, written:

\[ [P] \ c \ [Q] \]

The meaning of \([P] \ c \ [Q]\) is:

- If we begin executing \(c\) in an **environment satisfying** \(P\),
- **then** \(c\) **terminates**,
- **and** its final environment will satisfy \(Q\)

Total correctness introduces another obligation for verification
Obviously, we’ll need to prove that the program eventually terminates.

Intuitively, we’ll do the following:

1. Find a set $S$ with some ordering and *smallest* element.
2. Find a function $\delta$ that maps program states to $S$.
3. Prove that $\delta$ always decreases as the program executes.

$\delta$ is called a **ranking function**.

**Idea:** If the program diverges, $\delta$ would decrease infinitely.
Let $\prec$ be a binary predicate over some set $S$, and $\succ$ be its “inverse” well-founded.

A well-founded relation $\prec$ is such that there is no infinite sequence $s_1, s_2, s_3, \ldots$ in $S$ where each element is less than its predecessor:

$$s_1 \succ s_2 \succ s_3 \succ \cdots$$

In other words, each decreasing sequence in $S$ is finite.
Aside: Well-Founded Induction

Well-founded induction generalizes “normal” induction over numbers

Can use any binary predicate \( \prec \) that’s well-founded in our theory

- **Inductive Hypothesis:** For every \( n, n' \) where \( n' \prec n \), assume \( F[n'] \) is \( T \)-valid

- Prove that \( F[n] \) is \( T \)-valid

This is summarized by the axiom schema:

\[
(\forall n. (\forall n'. n' \prec n \rightarrow F[n']) \rightarrow F[n]) \rightarrow \forall x. F[x]
\]

Structural induction is an instance: \( \prec \) is the “subprogram” relation
Example: Well-Founded Relation

$< \text{ defined over the natural numbers}$

$0$ is the least element; any decreasing sequence of $\mathbb{N}$ is finite:

Think of inductively-defined sets

Axioms are the least elements:

\[
\text{succ} (\text{succ}(\cdots 0 \cdots)) \succ \text{succ}(\cdots 0 \cdots) \succ \cdots \succ 0
\]

$< \text{ over the rationals is } \textbf{not} \text{ well-founded}$

No least element:

\[
1 > \frac{1}{2} > \frac{1}{3} > \frac{1}{4} > \cdots > \frac{1}{n} > \frac{1}{n+1} > \cdots
\]
Given sets \( S_1, \ldots, S_n \) and relations \(<_1, \ldots, <_m\), let \( S = S_1 \times \cdots \times S_n \)

Define the relation \(<\):

\[
(s_1, \ldots, s_n) < (t_1, \ldots, t_n) \iff \bigvee_{i=1}^{m} \left( s_i <_i t_i \land \bigwedge_{j=1}^{i-1} s_j = t_j \right)
\]

So, \((s_1, \ldots, s_n) < (t_1, \ldots, t_n)\) iff:

- At some position \(i\), \(s_i <_i t_i\)
- and for all preceding positions \(j\), \(s_j = t_j\)

If \(<_1, \ldots, <_i\) are well-founded, then so is \(<\)
We use well-foundedness to prove termination

Basic strategy:

1. Choose $S$ and well-founded $\prec$.
   Usually $f(some\ var)$ or a tuple with lexicographic relation $<_n$

2. Find a suitable ranking function
   $$\delta : program\ states \rightarrow S$$

3. Show that $\delta$ decreases according to $\prec$ on every basic path

Program must terminate, as no infinite-descending sequences in $\prec$
We annotate ranking functions in code with ↓.

Needed in function prototype, head of loops

For example:

- ↓ $i$, as long as $i$’s domain has a well-founded ordering
- ↓ $u - l + 1$, assuming $u, l$ are naturals
- ↓ $(i + 1, j - 1)$, same assumption as above

Basic paths now have ranking functions at beginning and end

- Usually accompanied by assertion at the beginning
- By themselves at the end, to prove the path decreases
We produce the verification condition:

\[ P \rightarrow \text{wlp}(c_1; \cdots; c_n, \delta(x) < \delta(x'))[x/x'] \]

- The value of \( \delta(x) \) at the end is less than at the beginning
- Don’t want wlp to modify value of \( x \) at the beginning
- Rename it to \( x' \), then subst. \( x \) back in after computing wlp
Example: Termination Verification Condition

\[
\begin{align*}
\{ i \geq 1 \} & \\
\downarrow & i \\
i & := i - 1; \\
\downarrow & i
\end{align*}
\]

Let’s compute the VC:

\[
i \geq 1 \rightarrow \text{wl}(i := i - 1, i < i')[i/i']
\]

becomes

\[
i \geq 1 \rightarrow (i - 1 < i')[i/i']
\]

becomes

\[
i \geq 1 \rightarrow i - 1 < i
\]

This is valid, as expected
Our verification condition is:

\[ P \rightarrow \text{wlp}(c_1; c_2, (i + 1, i + 1) <_2 (i' + 1, i' - j'))[i/i', j/j'] \]

Computing the wlp:

\[ \text{wlp}(c_1; c_2, (i + 1, i + 1) <_2 (i' + 1, i' - j')) \]
\[ \leftrightarrow \text{wlp}(c_1, \text{wlp}(i := i - 1, (i + 1, i + 1) <_2 (i' + 1, i' - j')))) \]
\[ \leftrightarrow \text{wlp}(\text{assume } j \geq i, (i - 1 + 1, i - 1 + 1) <_2 (i' + 1, i' - j')))) \]
\[ \leftrightarrow j \geq i \rightarrow (i, i) <_2 (i' + 1, i' - j')) \]
Slightly Larger Example

\[ \{ P : i + 1 \geq 0 \wedge i - j \geq 0 \} \]
\[ \downarrow (i + 1, i - j) \]

\( c_1 : \) **assume** \( j \geq i \)

\( c_2 : \)
\[ i := i - 1; \]
\[ \downarrow (i + 1, i + 1) \]

Our verification condition is now:

\[ P \rightarrow (j \geq i \rightarrow (i, i) <_2 (i' + 1, i' - j'))[[i'/i', j'/j']] \]
\[ \leftrightarrow P \rightarrow (j \geq i \rightarrow (i, i) <_2 (i + 1, i - j)) \]
\[ \leftrightarrow i + 1 \geq 0 \wedge i - j \geq 0 \rightarrow (j \geq i \rightarrow (i, i) <_2 (i + 1, i - j)) \]
\[ \leftrightarrow i + 1 \geq 0 \wedge i - j \geq 0 \wedge j \geq i \rightarrow (i, i) <_2 (i + 1, i - j) \]
\[ \leftrightarrow i > 0 \wedge i = j \rightarrow (i, i) <_2 (i + 1, i - j) \]

Again, valid as expected
Strategies for developing inductive annotations

How to effectively prove correctness for larger programs