More on Program Logic

Robert Keller
21 February 2005

Making the "Triples" Method More Readable

- We could represent derivation of a program triple line-by-line, as in natural deduction proofs.
- Each line would be either:
  - A logic formula
  - A triple
- and each line would be appropriately justified as a premise, or from previously-derived lines.
- The problem is that lines containing large fragments of program code can become unwieldy.

Full Annotation of Programs

- As an alternative, we can put the logic formulas **in-line** with program statements.
- The understanding is that each line of the program is preceded and followed by a logic formula that can be justified by the derivation rules.
- The derivations themselves can be traced by a series of logic formulas.

Note on Notation

- In the following, we shall use indentation rather than braces to avoid confusion with the braces that demark logical formulas.

Example

- Previously we derived the triple

  $\{x \leq n\}$ while($x < n$) **x := x+1**$\{x = n\}$

- Here is the corresponding program fully annotated:

$\{x \leq n\}$
while(x < n)
    $\{x \leq n \wedge x < n\}$
    $\{x < n\}$
    $\{x+1 \leq n\}$
    x := x+1
    $\{x \leq n\}$
$\{x \leq n \wedge \neg(x < n)\}$
$\{x = n\}$

Example

- Previously we derived the triple
  $\{x = x_0 \wedge y = y_0\}$
  if(x > y) {z := x; x := y; y := z;}
  $\{x \leq y \wedge ((x = x_0 \wedge y = y_0) \vee (y = x_0 \wedge x = y_0))\}$

- Here is the corresponding program fully annotated:

  $\{x = x_0 \wedge y = y_0\}$
  if(x > y)
  $\{x > y \wedge (x = x_0 \wedge y = y_0)\}$
  $\{y \leq x \wedge ((y = x_0 \wedge x = y_0) \vee (x = x_0 \wedge y = y_0))\}$
  z := x;
  $\{y \leq z \wedge ((y = x_0 \wedge z = y_0) \vee (z = x_0 \wedge y = y_0))\}$
  x := y;
  $\{x \leq z \wedge ((x = x_0 \wedge z = y_0) \vee (z = x_0 \wedge x = y_0))\}$
  y := z;
  $\{x \leq y \wedge ((x = x_0 \wedge y = y_0) \vee (y = x_0 \wedge x = y_0))\}$
  $\{x \leq y \wedge ((x = x_0 \wedge y = y_0) \vee (y = x_0 \wedge x = y_0))\}$

Requirements for Full Annotation: Pure logic

- Each pair of successive logic formulas at the same level must be a derivable implication.

Requirements for Full Annotation: **Assignment**

- Each **assignment** statement must be preceded by a formula derived from the formula after it, according to the assignment rule.

Requirements for Full Annotation: **while** statements

- The while statement must be immediately preceded by its loop invariant.
- The while test must be immediately followed by the loop invariant conjoined with the test condition.
- The last line of the body (including annotations) of the while, must be the loop invariant.
- The line following the last line of the body (which is out-dented) must be the conjunction of the loop invariant and the **negation** of the test condition.

Check for Conformance

$\{x \leq n\}$
while(x < n)
  $\{x \leq n \wedge x < n\}$
  $\{x < n\}$
  $\{x+1 \leq n\}$
  x := x+1
  $\{x \leq n\}$
$\{x \leq n \wedge \neg(x < n)\}$
$\{x = n\}$

Requirements for Full Annotation: **if** statements

- The first line of the “true” branch of an **if** test must be the formula before the **if** conjoined with the test condition.

- The first line of the “false” branch of an **if** test must be the formula before the **if** conjoined with the negation of the test condition.

- The last line of each branch and the line following the overall **if** (which is out-dented) must be the same.

Check for Conformance

- $\{x = x_0 \wedge y = y_0\}$
  if(x > y)
  $\{x > y \wedge (x = x_0 \wedge y = y_0)\}$
  $\{y \leq x \wedge ((y = x_0 \wedge x = y_0) \vee (x = x_0 \wedge y = y_0))\}$
  z := x;
  $\{y \leq z \wedge ((y = x_0 \wedge z = y_0) \vee (z = x_0 \wedge y = y_0))\}$
  x := y;
  $\{x \leq z \wedge ((x = x_0 \wedge z = y_0) \vee (z = x_0 \wedge x = y_0))\}$
  y := z;
  $\{x \leq y \wedge ((x = x_0 \wedge y = y_0) \vee (y = x_0 \wedge x = y_0))\}$
  $\{x \leq y \wedge ((x = x_0 \wedge y = y_0) \vee (y = x_0 \wedge x = y_0))\}$

Example: Euclid's Algorithm for GCD

- All quantities are natural numbers.

- $\{(x = x_0) \wedge (y = y_0)\}$
  while(x ≠ y)
      if(x > y)
          x := x - y
      else
          y := y - x
  $\{ x = \gcd(x_0, y_0) \}$

Step 1: Figure out the Loop Invariant

- This is a creative process that generally can’t be automated.
- However, we know some **boundary conditions**:
  - The loop invariant must be **implied by** the initial assumption, modified by any initialization before the loop.
  - The loop invariant, together with the negation of the test condition, must **imply** the final expectation.
  - The loop invariant must be **preserved by** the loop body, and we can use the added assumption that the test condition is true to establish this.

Step 1: Figure out the Loop Invariant

- For the GCD program, we can scratch around and figure out that:

  $\gcd(x - y, x) = \gcd(x, y)$ if $x > y$

  $\gcd(x, y - x) = \gcd(x, y)$ if $y > x$

- We might even be able to prove these (by showing that the pairs $\{x-y, x\}$ and $\{x, y\}$ have the same sets of divisors).

- This suggests that the body of the loop **preserves** the value $\gcd(x, y)$.

- Moreover, we can see that the **initial** value of this quantity is that of $\gcd(x_0, y_0)$, although we don’t have this quantity in a variable **explicitly**.

Euclid's Algorithm with potential loop invariant

- All quantities are integer.

$\{(x = x_0) \wedge (y = y_0)\}$
$\{gcd(x, y) = gcd(x_0, y_0)\}$
while(x ≠ y)
    $\{gcd(x, y) = gcd(x_0, y_0) \wedge x \neq y\}$
    if(x > y)
        x := x - y
    else
        y := y - x
    $\{gcd(x, y) = gcd(x_0, y_0)\}$
$\{gcd(x, y) = gcd(x_0, y_0) \wedge \neg(x \neq y)\}$
$\{ x = gcd(x_0, y_0) \}$

Checking Step 1

- Now do a sanity check to see if the formulas obey the proper relationships.
  - $\{x = x_0 \wedge y = y_0\} \rightarrow \{gcd(x, y) = gcd(x_0, y_0)\}$?
  - $\{gcd(x, y) = gcd(x_0, y_0) \wedge \neg(x \neq y)\} \rightarrow \{ x = gcd(x_0, y_0) \}$?
- If not, interpose additional formulas, or try to fix the invariant.
- Above, we can use the additional fact:
  $gcd(x, x) = x$.

Step 2: Fill in the body formulas

- $\{(x = x_0) \wedge (y = y_0)\}$
  $\{gcd(x, y) = gcd(x_0, y_0)\}$
  while(x ≠y)
  $\quad\{gcd(x, y) = gcd(x_0, y_0) \wedge x \neq y\}$
  $\quad$**if(x > y)**
  $\qquad\{gcd(x, y) = gcd(x_0, y_0) \wedge x \neq y \wedge x > y\}$
  $\qquad$**x := x - y**
  $\qquad\{gcd(x, y) = gcd(x_0, y_0)\}$
  $\quad$**else**
  $\qquad\{gcd(x, y) = gcd(x_0, y_0) \wedge x \neq y \wedge \neg(x > y)\}$
  $\qquad$**y := y - x**
  $\qquad\{gcd(x, y) = gcd(x_0, y_0)\}$
  $\quad\{gcd(x, y) = gcd(x_0, y_0)\}$
  $\{gcd(x, y) = gcd(x_0, y_0) \wedge \neg(x \neq y)\}$
  $\{ x = gcd(x_0, y_0) \}$

Step 3: Fill in any needed connecting formulas

- $\{(x = x_0) \wedge (y = y_0)\}$
  $\{gcd(x, y) = gcd(x_0, y_0)\}$
  while(x ≠ y)
  $\{gcd(x, y) = gcd(x_0, y_0) \wedge x \neq y\}$
  if(x > y)
  $\{gcd(x, y) = gcd(x_0, y_0) \wedge x \neq y \wedge x > y\}$
  $\{gcd(x, y) = gcd(x_0, y_0) \wedge x > y\}$
  $\{gcd(x-y, x) = gcd(x_0, y_0)\}$
  x := x - y
  $\{gcd(x, y) = gcd(x_0, y_0)\}$
  else
  $\{gcd(x, y) = gcd(x_0, y_0) \wedge x \neq y \wedge \neg(x > y)\}$
  $\{gcd(x, y) = gcd(x_0, y_0) \wedge x < y\}$
  $\{gcd(x, y-x) = gcd(x_0, y_0)\}$
  y := y - x
  $\{gcd(x, y) = gcd(x_0, y_0)\}$
  $\{gcd(x, y) = gcd(x_0, y_0)\}$
  $\{gcd(x, y) = gcd(x_0, y_0) \wedge \neg(x \neq y)\}$
  $\{gcd(x, y) = gcd(x_0, y_0) \wedge (x = y)\}$
  $\{ x = gcd(x_0, y_0) \}$

Example: Factorial Program
(one of many)

- $\{n = n_0 \wedge n_0 \geq 0\}$

```
f := 1;
while( n > 0 )
    f := f * n;
    n := n - 1;
```

$\{f = n_0!\}$

Arrays

- An array is, in effect, a **function**.
- Array x[0..n-1] is a function from the set of indices {0, 1, …, n-1} to the domain from which array values are drawn.
- So x[i] can be thought of x(i) (x **applied to** i).

Sub-Arrays and Index Sets

- Suppose x[0..n-1] is an array.
- For integers r and s, where r, s $\in$ {0, …, n-1}

  [r..s]

  denotes the set of indices {r, r+1, …, s}.

- If r > s then [r..s] is the empty set.

- x[r..s] denotes the sub-array of x with indices [r..s].

Example

- If x[0..5] is the array [2, 3, 5, 7, 11, 13] with indices [0..5]

  then x[1..4] is the sub-array [3, 5, 7, 11] with indices [1..4].

Proofs of Programs That Use Arrays

- Dichotomy:
  - Arrays are read-only
  - Arrays are modified by assignment

Quantifying over Array Indices

- As before, for any formula φ, $\varphi[j/i]$ means the formula obtained by substituting j for all free occurrences of i in φ.

- $\forall j \in [m..n]\ \varphi[j/i]$ means that

  $\varphi[m/i] \wedge \varphi[(m+1)/i] \wedge \ldots \wedge \varphi[n/i]$

- $\exists j \in [m..n]\ \varphi[j/i]$ means that

  $\varphi[m/i] \vee \varphi[(m+1)/i] \vee \ldots \vee \varphi[n/i]$

- If m > n, then the first case is equivalent to true, while the second case is equivalent to false.

Some Properties of Array Reference

- $((\forall j \in [m..n]\ \varphi[j/i]) \wedge \varphi[(n+1)/i])$
  $\to (\forall j \in [m..(n+1)]\ \varphi[j/i])$
- $((\forall j \in [m..n]\ \varphi[j/i]) \wedge \varphi[(m-1)/i])$
  $\to (\forall j \in [(m-1)..n]\ \varphi[j/i])$
- $((\exists j \in [m..n]\ \varphi[j/i]) \wedge \varphi[(n+1)/i])$
  $\to (\exists j \in [m..(n+1)]\ \varphi[j/i])$
- $((\exists j \in [m..n]\ \varphi[j/i]) \wedge \varphi[(m-1)/i])$
  $\to (\exists j \in [(m-1)..n]\ \varphi[j/i])$

Example Read-Only Array Program

- Determine whether a specific value y occurs in the sub-array x[m..n], using = comparison.

- ```
  i := m;
  occurs := false;
  while( i ≤ n )
      if( x[i] = y )
          occurs := true;
      i := i+1;
  ```
  ```

Specification

- $\{m \leq n\}$ Assumption

```
i := m;
occurs := false;
while( i ≤ n )
     if( x[i] = y )
          occurs := true;
     i := i+1;
```

$\{occurs \leftrightarrow \exists j \in [m..n]\ x[j] = y\}$ Expectation

Note that we don't need to anchor the initial array values here, since the array never changes.

Step 1: Determine the Loop Invariant

- $\{m \leq n\}$ Assumption
  i := m;
  occurs := false;
  $\{i \leq n+1 \wedge occurs \leftrightarrow \exists j \in [m..i-1]\ x[j] = y\}$ Invariant
  while(i ≤ n)
      if(x[i] = y)
          occurs := true;
      i := i+1;
      $\{i \leq n+1 \wedge occurs \leftrightarrow \exists j \in [m..i-1]\ x[j] = y\}$ Invariant
  $\{occurs \leftrightarrow \exists j \in [m..n]\ x[j] = y\}$ Expectation

Step 2: Begin Filling In

- $\{m \leq n\}$
  i := m;
  occurs := false;
  $\{i \leq n+1 \wedge occurs \leftrightarrow \exists j \in [m..i-1]\ x[j] = y\}$
  while(i $\leq$ n)
  $\{i \leq n \wedge i \leq n+1 \wedge occurs \leftrightarrow \exists j \in [m..i-1]\ x[j] = y\}$ ←
  if(x[i] = y)
  occurs := true;
  i := i+1;
  $\{i \leq n+1 \wedge occurs \leftrightarrow \exists j \in [m..i-1]\ x[j] = y\}$
  $\{\neg(i \leq n) \wedge i \leq n+1 \wedge occurs \leftrightarrow \exists j \in [m..i-1]\ x[j] = y\}$ ←
  $\{occurs \leftrightarrow \exists j \in [m..n]\ x[j] = y\}$

Step 2: Continue Filling In

- $\{m \leq n\}$
  i := m;
  occurs := false;
  $\{i \leq n+1 \wedge occurs \leftrightarrow \exists j \in [m..i-1]\ x[j] = y\}$
  while(i $\leq$ n)
  $\quad\{i \leq n \wedge i \leq n+1 \wedge occurs \leftrightarrow \exists j \in [m..i-1]\ x[j] = y\}$
  $\quad\{i \leq n \wedge occurs \leftrightarrow \exists j \in [m..i-1]\ x[j] = y\}$ ←
  $\quad$**if(x[i] = y)**
  $\qquad\{x[i] = y \wedge i \leq n \wedge occurs \leftrightarrow \exists j \in [m..i-1]\ x[j] = y\}$ ←
  $\qquad$**occurs := true;**
  $\quad$**i := i+1;**
  $\quad\{i \leq n+1 \wedge occurs \leftrightarrow \exists j \in [m..i-1]\ x[j] = y\}$
  $\{\neg(i \leq n) \wedge i \leq n+1 \wedge occurs \leftrightarrow \exists j \in [m..i-1]\ x[j] = y\}$
  $\{i = n+1 \wedge occurs \leftrightarrow \exists j \in [m..i-1]\ x[j] = y\}$ ←
  $\{occurs \leftrightarrow \exists j \in [m..n]\ x[j] = y\}$

Step 2: Still More Filling In

- $\{m \leq n\}$
  i := m;
  $\{i \leq n+1 \wedge \text{false} \leftrightarrow \exists j \in [m..i-1]\ x[j] = y\}$ ←
  occurs := false;
  $\{i \leq n+1 \wedge \text{occurs} \leftrightarrow \exists j \in [m..i-1]\ x[j] = y\}$
  while(i ≤ n)
  $\{i \leq n \wedge i \leq n+1 \wedge \text{occurs} \leftrightarrow \exists j \in [m..i-1]\ x[j] = y\}$
  $\{i \leq n \wedge \text{occurs} \leftrightarrow \exists j \in [m..i-1]\ x[j] = y\}$
  if(x[i] = y)
  $\{x[i] = y \wedge i \leq n \wedge \text{occurs} \leftrightarrow \exists j \in [m..i-1]\ x[j] = y\}$
  $\{i \leq n \wedge \text{true} \leftrightarrow \exists j \in [m..i]\ x[j] = y\}$ ←
  occurs := true;
  $\{i+1 \leq n+1 \wedge \text{occurs} \leftrightarrow \exists j \in [m..i+1-1]\ x[j] = y\}$ ←
  i := i+1;
  $\{i \leq n+1 \wedge \text{occurs} \leftrightarrow \exists j \in [m..i-1]\ x[j] = y\}$
  $\{\neg(i \leq n) \wedge i \leq n+1 \wedge \text{occurs} \leftrightarrow \exists j \in [m..i-1]\ x[j] = y\}$
  $\{i = n+1 \wedge \text{occurs} \leftrightarrow \exists j \in [m..i-1]\ x[j] = y\}$
  $\{\text{occurs} \leftrightarrow \exists j \in [m..n]\ x[j] = y\}$

Step 2: Still^2 More Filling In

- $\{m \leq n\}$
  $\{m \leq n+1 \land \text{false} \leftrightarrow \exists j \in [m..m-1]\ x[j] = y\}$
  i := m;
  $\{i \leq n+1 \land \text{false} \leftrightarrow \exists j \in [m..i-1]\ x[j] = y\}$
  occurs := false;
  $\{i \leq n+1 \land \text{occurs} \leftrightarrow \exists j \in [m..i-1]\ x[j] = y\}$
  while(i $\leq$ n)
  $\{i \leq n \land i \leq n+1 \land \text{occurs} \leftrightarrow \exists j \in [m..i-1]\ x[j] = y\}$
  $\{i \leq n \land \text{occurs} \leftrightarrow \exists j \in [m..i-1]\ x[j] = y\}$
  if(x[i] = y)
  $\{x[i] = y \land i \leq n \land \text{occurs} \leftrightarrow \exists j \in [m..i-1]\ x[j] = y\}$
  $\{i \leq n \land \text{true} \leftrightarrow \exists j \in [m..i]\ x[j] = y\}$
  occurs := true;
  $\{i \leq n \land \text{occurs} \leftrightarrow \exists j \in [m..i]\ x[j] = y\}$
  $\{i+1 \leq n+1 \land \text{occurs} \leftrightarrow \exists j \in [m..i+1-1]\ x[j] = y\}$
  i := i+1;
  $\{i \leq n+1 \land \text{occurs} \leftrightarrow \exists j \in [m..i-1]\ x[j] = y\}$
  $\{\neg(i \leq n) \land i \leq n+1 \land \text{occurs} \leftrightarrow \exists j \in [m..i-1]\ x[j] = y\}$
  $\{i = n+1 \land \text{occurs} \leftrightarrow \exists j \in [m..i-1]\ x[j] = y\}$
  $\{\text{occurs} \leftrightarrow \exists j \in [m..n]\ x[j] = y\}$

Conclusion

- $\{m \le n\}$
  $\{m \le n+1 \wedge \text{false} \leftrightarrow \exists j \in [m..m-1]\ x[j] = y\}$
  i := m;
  $\{i \le n+1 \wedge \text{false} \leftrightarrow \exists j \in [m..i-1]\ x[j] = y\}$
  occurs := false;
  $\{i \le n+1 \wedge \text{occurs} \leftrightarrow \exists j \in [m..i-1]\ x[j] = y\}$
  while(i $\le$ n)
  $\{i \le n \wedge i \le n+1 \wedge \text{occurs} \leftrightarrow \exists j \in [m..i-1]\ x[j] = y\}$
  $\{i \le n \wedge \text{occurs} \leftrightarrow \exists j \in [m..i-1]\ x[j] = y\}$
  if(x[i] = y)
  $\{x[i] = y \wedge i \le n \wedge \text{occurs} \leftrightarrow \exists j \in [m..i-1]\ x[j] = y\}$
  $\{i \le n \wedge \text{true} \leftrightarrow \exists j \in [m..i]\ x[j] = y\}$
  occurs := true;
  $\{i \le n \wedge \text{occurs} \leftrightarrow \exists j \in [m..i]\ x[j] = y\}$ $\leftarrow$
  $\{i \le n \wedge \text{occurs} \leftrightarrow \exists j \in [m..i]\ x[j] = y\}$
  $\{i+1 \le n+1 \wedge \text{occurs} \leftrightarrow \exists j \in [m..i+1-1]\ x[j] = y\}$
  i := i+1;
  $\{i \le n+1 \wedge \text{occurs} \leftrightarrow \exists j \in [m..i-1]\ x[j] = y\}$
  $\{\neg(i \le n) \wedge i \le n+1 \wedge \text{occurs} \leftrightarrow \exists j \in [m..i-1]\ x[j] = y\}$
  $\{i = n+1 \wedge \text{occurs} \leftrightarrow \exists j \in [m..i-1]\ x[j] = y\}$
  $\{\text{occurs} \leftrightarrow \exists j \in [m..n]\ x[j] = y\}$

Exercise: Optimized Version of the Previous Program

- Determine whether a specific value y occurs in the sub-array x[m..n], using = comparison.

- ```
  i := m;
  occurs := false;
  while( ¬occurs ∧ i ≤ n )
      if( x[i] = y )
          occurs := true;
      i := i+1;
  ```
  ```

Example Read-Only Array Program

- Find the index of a **maximal** value in the sub-array x[m..n], using > comparison.

- {m $\leq$ n} Assumption

```
k := m;
i := m+1;
while( i ≤ n )
      if( x[i] > x[k] )
              k = i;
      i := i+1;
```

{$\forall$j $\in$ [m..n] x[j] $\leq$ x[k]} Expectation

Note that we don't need to anchor the initial array values here, since the array never changes.

Step 1: Find the loop invariant

- $\{m \leq n\}$
  k := m;
  i := m+1;
  $\{\forall j \in [m..i-1]\ \ x[j] \leq x[k] \wedge (i \leq n+1)\}$
  while(i $\leq$ n)
      if(x[i] > x[k])
          k := i;
      i := i+1;
      $\{\forall j \in [m..i-1]\ \ x[j] \leq x[k] \wedge (i \leq n+1)\}$
  $\{\forall j \in [m..n]\ \ x[j] \leq x[k]\}$

Step 2: Begin Filling In

- $\{m \leq n\}$
  k := m;
  i := m+1;
  $\{\forall j \in [m..i-1]\ \ x[j] \leq x[k] \wedge (i \leq n+1)\}$
  while(i ≤ n)
      $\{\forall j \in [m..i-1]\ \ x[j] \leq x[k] \wedge (i \leq n+1) \wedge (i \leq n)\}$
      if(x[i] > x[k])
          k := i;
      i := i+1;
      $\{\forall j \in [m..i-1]\ \ x[j] \leq x[k] \wedge (i \leq n+1)\}$
  $\{\forall j \in [m..i-1]\ \ x[j] \leq x[k] \wedge (i \leq n+1) \wedge \neg(i \leq n)\}$
  $\{\forall j \in [m..n]\ \ x[j] \leq x[k]\}$

Step 2: More Filling In

- $\{m \leq n\}$
  k := m;
  $\{\forall j \in [m..m+1-1]\ \ x[j] \leq x[k] \wedge (m+1 \leq n+1)\}$
  i := m+1;
  $\{\forall j \in [m..i-1]\ \ x[j] \leq x[k] \wedge (i \leq n+1)\}$
  while(i $\leq$ n)
      $\{\forall j \in [m..i-1]\ \ x[j] \leq x[k] \wedge (i \leq n+1) \wedge (i \leq n)\}$
      if(x[i] > x[k])
          k := i;
      $\{\forall j \in [m..i+1-1]\ \ x[j] \leq x[k] \wedge (i+1 \leq n+1)\}$
      i := i+1;
      $\{\forall j \in [m..i-1]\ \ x[j] \leq x[k] \wedge (i \leq n+1)\}$
  $\{\forall j \in [m..i-1]\ \ x[j] \leq x[k] \wedge (i \leq n+1) \wedge \neg(i \leq n)\}$
  $\{\forall j \in [m..n]\ \ x[j] \leq x[k]\}$

Step 2: More^2 Filling In

- $\{m \le n\}$
  k := m;
  $\{\forall j \in [m..m]\ \ x[j] \le x[k] \wedge (m \le n)\}$
  $\{\forall j \in [m..m+1-1]\ \ x[j] \le x[k] \wedge (m+1 \le n+1)\}$
  i := m+1;
  $\{\forall j \in [m..i-1]\ \ x[j] \le x[k] \wedge (i \le n+1)\}$
  while(i $\le$ n)
  $\{\forall j \in [m..i-1]\ \ x[j] \le x[k] \wedge (i \le n+1) \wedge (i \le n)\}$
  if(x[i] > x[k])
  k := i;
  $\{\forall j \in [m..i]\ \ x[j] \le x[k] \wedge (i \le n)\}$
  $\{\forall j \in [m..i]\ \ x[j] \le x[k] \wedge (i \le n)\}$
  $\{\forall j \in [m..i+1-1]\ \ x[j] \le x[k] \wedge (i+1 \le n+1)\}$
  i := i+1;
  $\{\forall j \in [m..i-1]\ \ x[j] \le x[k] \wedge (i \le n+1)\}$
  $\{\forall j \in [m..i-1]\ \ x[j] \le x[k] \wedge (i \le n+1) \wedge \neg(i \le n)\}$
  $\{\forall j \in [m..n]\ \ x[j] \le x[k]\}$

Step 2: More^3 Filling In

- $\{m \leq n\}$
  k := m;
  $\{\forall j \in [m..m]\ \ x[j] \leq x[k] \wedge (m \leq n)\}$
  $\{\forall j \in [m..m+1-1]\ \ x[j] \leq x[k] \wedge (m+1 \leq n+1)\}$
  i := m+1;
  $\{\forall j \in [m..i-1]\ \ x[j] \leq x[k] \wedge (i \leq n+1)\}$
  while(i ≤ n)
      $\{\forall j \in [m..i-1]\ \ x[j] \leq x[k] \wedge (i \leq n+1) \wedge (i \leq n)\}$
      if(x[i] > x[k])
          $\{\forall j \in [m..i]\ \ x[j] \leq x[i] \wedge (i \leq n)\}$
          k := i;
          $\{\forall j \in [m..i]\ \ x[j] \leq x[k] \wedge (i \leq n)\}$
      $\{\forall j \in [m..i]\ \ x[j] \leq x[k] \wedge (i \leq n)\}$
      $\{\forall j \in [m..i+1-1]\ \ x[j] \leq x[k] \wedge (i+1 \leq n+1)\}$
      i := i+1;
      $\{\forall j \in [m..i-1]\ \ x[j] \leq x[k] \wedge (i \leq n+1)\}$
  $\{\forall j \in [m..i-1]\ \ x[j] \leq x[k] \wedge (i \leq n+1) \wedge \neg(i \leq n)\}$
  $\{\forall j \in [m..n]\ \ x[j] \leq x[k]\}$

Are We Done Yet?

- $\{m \leq n\}$
  $\{\forall j \in [m..m]\ \ x[j] \leq x[m] \wedge (m \leq n)\}$
  k := m;
  $\{\forall j \in [m..m]\ \ x[j] \leq x[k] \wedge (m \leq n)\}$
  $\{\forall j \in [m..m+1-1]\ \ x[j] \leq x[k] \wedge (m+1 \leq n+1)\}$
  i := m+1;
  $\{\forall j \in [m..i-1]\ \ x[j] \leq x[k] \wedge (i \leq n+1)\}$
  while(i $\leq$ n)
  $\{\forall j \in [m..i-1]\ \ x[j] \leq x[k] \wedge (i \leq n+1) \wedge (i \leq n)\}$
  if(x[i] > x[k])
  $\{\forall j \in [m..i]\ \ x[j] \leq x[k] \wedge (i \leq n) \wedge x[k] < x[i]\}$
  $\{\forall j \in [m..i]\ \ x[j] \leq x[i] \wedge (i \leq n)\}$
  k := i;
  $\{\forall j \in [m..i]\ \ x[j] \leq x[k] \wedge (i \leq n)\}$
  $\{\forall j \in [m..i]\ \ x[j] \leq x[k] \wedge (i \leq n)\}$
  $\{\forall j \in [m..i+1-1]\ \ x[j] \leq x[k] \wedge (i+1 \leq n+1)\}$
  i := i+1;
  $\{\forall j \in [m..i-1]\ \ x[j] \leq x[k] \wedge (i \leq n+1)\}$
  $\{\forall j \in [m..i-1]\ \ x[j] \leq x[k] \wedge (i \leq n+1) \wedge \neg(i \leq n)\}$
  $\{\forall j \in [m..n]\ \ x[j] \leq x[k]\}$

Assignment to Arrays

- A way to think about arrays in terms of assignment, such as

  x[i] := 5*x[i-1];

  is **as if**:

  - The assignment installs a **new** function as the value of the variable x.

What is this new function?

- The function x before the assignment had values:

  $x[j] = x_0[j]$ for each j

- The function x after the **assignment**

  $x[i] := E;$

  has values:

  $x[j] = (j = i)\ ?\ E : x_0[j]$ for each j

Array-Update Shorthand Notation

- Let x be an array. By

  $$x(i{:}V)$$

  we mean the array that is like x, except that x[i] has been replaced with value V.

- In other words

$$x(i{:}V)[j] = \begin{cases} V & \text{if } j = i \\ x[j] & \text{if } j \neq i \end{cases}$$

Some Axioms for Array Update Notation

- $x(i{:}V)[i] = V$
- $j \neq i \rightarrow x(i{:}V)[j] = x[j]$
- $x(i{:}V)(i{:}V') = x(i{:}V')$
- $j \neq i \rightarrow x(i{:}V)(j{:}W) = x(j{:}W)(i{:}V)$

Example Using in a Proof Rule

- $\{??\}\ x[5] := 99;\ \{\forall j\ (j > 5 \rightarrow x[j] \leq 100)\}$
- According to the assignment rule,

  $??$ is $\forall j\ (j > 5 \rightarrow x(5{:}99)[j] \leq 100)$

  which is seen to be implied by

  $\forall j\ (j > 4 \rightarrow x[j] \leq 100))$ since $99 \leq 100$

Assignment Rule for Arrays in Hein

- $\{\varphi'\}$ x[i] = E; $\{\varphi\}$

where

φ' is like φ **except**

Any occurrence of expression **x[i]** (the LHS) in φ is replaced with expression E.

Any occurrence of **x[j]**
where j is not the variable i is replaced with
(j = i) ? E : x[j].

Example (illustrates both cases)

- {??} x[5] := 99; {x[5] > x[6]}

  Identify i with 5, x[i] with x[5], E with 99

- ?? is 99 > (6 = 5 ? 99 : x[j])

  (false ? 99 : x[6])

  99 > x[6] when simplified

Example

- $\{??\}\ x[5] := 99;\ \{\forall j\ (j > 5 \rightarrow x[j] \leq 100)\}$

  ?? is

  $\forall j\ (j > 5 \rightarrow (j = 5\ ?\ 99 : x[j]) \leq 100)$

  which doesn't seem as helpful as the one derived earlier, since it is equivalent to:

  $$\forall j\ (j > 5 \rightarrow (\text{false}\ ?\ 99 : x[j]) \leq 100)$$

  i.e.

  $$\forall j\ (j > 5 \rightarrow x[j] \leq 100)$$

Dis-Allowed Case

- The rule doesn't allow the array index to be a function of the array itself, as in:

  x[x[1]] := 2;

- Consider {??} x[x[1]] = 2; {x[x[1]] = 2}
- ?? would be (j = x[1] ? 2 : x[j])[1] = 2
- But if x[1] were 1 before assignment, then x[1] would be 2 after,
  whereas ?? says that x[1] must be 2 before the assignment.