Announcements

- QUIZ 1 today. You have 10 minutes
- Hand in quiz at the end of class

Announcements

- HW0 due today
  - Push to Git then Submit in Submitty
  - HW1 will be up after class
  - Check Homeworks for announcement

- Dafny binaries now available for all OS!
  [https://github.com/Microsoft/dafny](https://github.com/Microsoft/dafny)

- No office hours today. MYAD

Backward Reasoning: Rule for Assignment

```
{ wp("x=expression",Q) }
x = expression;
{ Q }
```

Rule: the weakest precondition

\( \text{wp}(\text{"x=expression","Q\}) = Q \text{ with all occurrences of } x \text{ in } Q \text{ replaced by } expression \)

Backward Reasoning: Rule for Sequence

```
// find weakest precondition for sequence S1; S2 and Q
{ wp(S1,wp(S2,Q)) }
S1; // statement
{ wp(S2,Q) }
S2; // another statement
{ Q }
```

Simple Example

```
{x + 1 + y > 1} equiv. to \{x + y > 0\}
x = x + 1;
{x + y > 1}
y = x + y;
{y > 1}
```
Rule for If-then-else

```java
if (b) {
    S1;
} else {
    S2;
}
{ Q }
```

Fall 17 CSCI 2600, A Milanova

Proving Correctness

Goal: Prove that \( \{ P \} \text{ code } \{ Q \} \) is a valid triple

- Backward reasoning: \( \{ P \} \)
  - derive \( \{ P' \} \)
  - code \( \{ Q' \} \)
  - Then show \( P \Rightarrow P' \)
- Forward reasoning:
  - code \( \{ Q \} \)
  - Then show \( Q' \Rightarrow Q \)

What Happens When There Is a Loop

Precondition: \( x \geq 0 \);
\( i = x; \)
\( z = 0; \)
while \( (i \neq 0) \)
  \( z = z+1; \)
  \( i = i-1; \)
\{ Q \}
Postcondition: \( x = z; \)

Outline

- Reasoning about loops
  - Total correctness = partial correctness + termination
  - Loop invariants
  - Computation induction (to prove partial correctness)
  - Decrementing function (to prove termination)

Reasoning About Loops

- Reasoning about loops is difficult
  - Unknown number of iterations and unknown number of paths. Recursion poses similar issues
  - We cannot enumerate all paths
  - Key is to guess a loop invariant
- Two things to prove about loops
  1. It computes correct values (partial correctness)
  2. It terminates (does not go into infinite loop)
  - total correctness = partial correctness + termination

Example: Partial Correctness + Termination

Precondition: \( x \geq 0 \);
\( i = x; \)
\( z = 0; \)
while \( (i \neq 0) \)
  \( i+z = x \)
  \{ z = z+1; i = i-1; \}
Postcondition: \( x = z; \)

1. \( i+z \) gives us that \( i+z = x \) holds at 0th iteration of loop // Base case
2. Assuming that \( i+z = x \) holds after \( k \)th iteration, we show it holds after \( (k+1) \)th iteration // Induction
   - \( z_{k+1} = z + 1 \) and \( i_{k+1} = i - 1 \) thus \( z_{k+1} + i_{k+1} = z + 1 + i - 1 = z+i = x \)
3. If loop terminated, we know \( i = 0 \). Since \( z+i = x \) holds, we have \( z = x \)

Example: Partial Correctness + Termination

Precondition: \( x \geq 0 \);
\( i = x; \)
\( z = 0; \)
while \( (i \neq 0) \)
  \( i+z = x \)
  \{ z = z+1; i = i-1; \}
Postcondition: \( x = z; \)

1. Loop terminates. Precondition \( x \geq 0 \) guarantees \( i \geq 0 \) before the loop.
   - \( i \geq 0 \) continues to hold as we iterate.
   - At each iteration \( i \) decreases by 1, thus it eventually reaches 0.
Reasoning About Loops by Induction

- **Loop invariant** (a fact that holds true before and after each loop iteration)
  - Even though `i` and `z` change, `i+z=x` stays true
  - We just made an inductive argument over the number of iterations of the loop
- **Computation induction**
  - Established that loop invariant holds at 0th iteration
  - Assuming that loop invariant holds after `k`th iteration, show that it holds after `(k+1)`st iteration

1. **Partial correctness**
   - Establish and prove loop invariant using computation induction
   - Loop exit condition and loop invariant must imply the desired postcondition
     - `i = 0` (loop exit condition) and `i+z = x` imply `z = x`

2. **Termination**
   - (Roughly) Establish “decrementing function” `D`
     - (1) `D ≥ 0`, (2) each iteration strictly decreases `D`, (3) loop invariant and `D` at 0 imply loop exit condition
   - We will discuss Partial Correctness first

Another Example: Partial Correctness + Termination

Let us prove that `sum` is correct:

- **Precondition:** `len ≥ 1` and `arr.length = len`
  - `int sum = arr[0];`
  - `int i = 1;`
  - `while (i < len) {
    sum = sum + arr[i];
    i = i+1;
  }
  `Postcondition: sum = arr[0] + ... + arr[arr.length-1]`

Another Example: Partial Correctness

- `Σ: 0 to i-1 arr[j]` stands for `arr[0]+arr[1]+...+arr[i-1]`
  - `Loop invariant: i <= len` and `sum = Σ: 0 to i-1 arr[j]`
  - `1) i <= len` and `sum = Σ: 0 to i-1 arr[j]` holds before loop
  - `2) Assume sum = Σ: 0 to i-1 arr[j]` holds after `k`th iteration
    - `sum_new = sum + arr[i];`
    - `i_new = i+1;`
    - Thus `sum = Σ: 0 to i_new arr[j]` holds after `(k+1)`th iteration
  - `i <= len` also holds (had we had `i = len` after `k`th iteration, there wouldn’t have been a `(k+1)`th iteration!)

Another Example: Partial Correctness

- `Σ: 0 to i-1 arr[j]` stands for `arr[0]+arr[1]+...+arr[i-1]`
  - `3) (i < len)` which is `i >= len` plus loop invariant
  - `i <= len ∧ sum = Σ: 0 to len-1 arr[j]`
  - `implies sum = Σ: 0 to len-1 arr[j]`, which with precondition `arr.length=len` gives us desired postcondition `sum = arr[0] + ... + arr[arr.length-1]`

We still must argue termination!

Another Example: Termination

- **Precondition:** `len ≥ 1` and `arr.length = len`
  - `int sum = arr[0];`
  - `int i = 1;`
  - `while (i < len) {
    sum = sum + arr[i];
    i = i+1;
  }
  `4) We now argue termination (informally).
  - At each iteration `i` increases by one, while `len` stays the same.
  - Thus, eventually `i` reaches `len`.
  - `i = len` implies loop exit condition.
  - More on termination later!
Partial Correctness, More Formally

\{ P \} \textbf{while} (b) \ S \ \{ Q \}

Must "guess" a loop invariant \texttt{Inv} such that

1) \( P \Rightarrow \texttt{Inv} \) // \texttt{Inv} holds before loop. Base case

2) \( \{ b \ \&\& \texttt{Inv} \} \ S \ \{ \texttt{Inv} \} \) // Assuming \texttt{Inv} held after \( k \)th iteration and execution took a \((k+1)\)st iteration, then \texttt{Inv} holds after \((k+1)\)th iteration

3) \((\neg b \ \&\& \texttt{Inv}) \Rightarrow Q\) // The exit condition and loop invariant imply desired postcondition

Choosing Loop Invariant

What is a suitable loop invariant?

**Precondition:** \( x \geq 0 \)

\begin{verbatim}
\texttt{i = x;}
\texttt{z = 0;}
\texttt{while (i != 0) \{}
\texttt{\ \ \ \ z = z+1;}
\texttt{\ \ \ \ i = i-1;}
\texttt{\}}
\end{verbatim}

**Postcondition:** \( x = z \)

Choosing Loop Invariant

What is a suitable loop invariant?

**Precondition:** \( x \geq 0 \) && \( y = 0 \)

\begin{verbatim}
\texttt{while (x != y) \{}
\texttt{\ \ \ \ y = y+1;}
\texttt{\}}
\end{verbatim}

**Postcondition:** \( x = y \)

Choosing Loop Invariant

What is a suitable loop invariant?

**Precondition:** \( \texttt{len} \geq 1 \) && \( \texttt{arr.length} = \texttt{len} \)

\begin{verbatim}
\texttt{int sum = arr[0];}
\texttt{int i = 1;}
\texttt{while (i < \texttt{len}) \{}
\texttt{\ \ \ \ sum = sum + arr[i];}
\texttt{\ \ \ \ i = i+1;}
\texttt{\}}
\end{verbatim}

**Postcondition:** \( \texttt{sum = arr[0]+...+arr[arr.length-1]} \)

Choosing Loop Invariant

What is a suitable loop invariant?

**Precondition:** \( n \geq 0 \)

\begin{verbatim}
\texttt{i = 0;}
\texttt{r = 1;}
\texttt{while (i < n) \{}
\texttt{\ \ \ \ i = i+1;}
\texttt{\ \ \ \ r = r*i;}
\texttt{\}}
\end{verbatim}

**Postcondition:** \( r = n! \)

Choosing Loop Invariant

**Precondition:** \( \texttt{len} \geq 1 \) && \( \texttt{a.length} = \texttt{len} \)

\begin{verbatim}
\texttt{int sum = arr[0];}
\texttt{int i = 1;}
\texttt{while (i < \texttt{len}) \{}
\texttt{\ \ \ \ invariant \texttt{sum = a[0]+...+a[i-1]} && i<=len}
\texttt{\ \ \ \ sum = sum + a[i];}
\texttt{\ \ \ \ i = i+1;}
\texttt{\}}
\end{verbatim}

**Postcondition:** \( \texttt{sum = a[0]+...+a[a.length-1]} \)

A “tedious” Invariant

\begin{figure}
\centering
\includegraphics[width=\textwidth]{tedious_invariant.png}
\caption{A “tedious” Invariant}
\end{figure}

**Precondition:** \( \texttt{len} \geq 1 \) && \( \texttt{a.length} = \texttt{len} \)

\begin{verbatim}
\texttt{int sum = arr[0];}
\texttt{int i = 1;}
\texttt{while (i < \texttt{len}) \{}
\texttt{\ \ \ \ \texttt{invariant \texttt{sum = a[0]+...+a[i-1]} && i<=len}
\texttt{\ \ \ \ sum = sum + a[i];}
\texttt{\ \ \ \ i = i+1;}
\texttt{\}}
\end{verbatim}

**Postcondition:** \( \texttt{sum = a[0]+...+a[a.length-1]} \)
A “tedious” Invariant

Inv: \( \text{sum} = a[0] + a[1] + \ldots + a[i-1] \)
\( i = 3 \)

Precondition: \( \text{len} \geq 1 \) && \( a.length = \text{len} \)
int \( \text{sum} = \text{arr}[0] \);
int \( i = 1; \)
while (\( i < \text{len} \)) {
    invariant \( \text{sum} = a[0] + \ldots + a[i-1] \) && \( i = \text{len} \)
    \( \text{sum} = \text{sum} + a[i]; \)
    \( i = i + 1; \)
}
Postcondition: \( \text{sum} = a[0] + \ldots + a[a.length-1] \)

A “tedious” Invariant

Inv: \( \text{sum} = a[0] + a[1] + \ldots + a[a.length-1] \)

As the induction variable \( i \) moves through the array, \( \text{sum} \) contains the correct partial result.
At the end of the array, \( \text{sum} \) contains the correct total.

Similar invariants for min, max, avg, sorting algorithms, etc. These invariants are easy to guess.

Fall 17 CSCI 2600, A Milanova

A “tedious” Invariant

Inv: \( \text{sum} = a[0] + a[1] + \ldots + a[a.length-1] \)

Precondition: \( x \geq 0 \);
i = x;
z = 0;
while (\( i != 0 \)) {
    \( z = z + 1; \)
    \( i = i - 1; \)
}
Postcondition: \( x = z \)

An “elegant” Invariant

What is a suitable loop invariant?
Precondition: \( x \geq 0 \);
i = x;
z = 0;
while (\( i != 0 \)) {
    z = z + 1;
    i = i - 1;
}
Postcondition: \( x = z \)

An “elegant” Invariant

Precondition: \( x != 0 \)
zeros = 0;
y = x;
while (\( y \% 10 == 0 \)) {
    y = y / 10;
    zeros = zeros + 1;
}
Postcondition: \( x = y * 10 \) && \( y \% 10 \neq 0 \)

Fall 17 CSCI 2600, A Milanova (example due to Mike Ernst)

Termination, More Formally

- We must “guess” a decrementing function \( D \)
- Maps state to natural numbers: \( 0, 1, 2, \ldots \)

\{ P \} while (\( b \)) S \{ Q \}

We need \( D \), with range in the natural numbers, such that
1) \{ Inv && b \} S \{ D_{after} < D_{before} \} // Execution of loop reduces the value of \( D \)
2) \{ Inv && D=0 \} => \sim b \ // Inv + D reaching 0 (D’s minimal value), must imply the loop exit condition

Choosing Decrementing Function

What is a suitable decrementing function?
Precondition: \( x \geq 0 \);
i = x;
z = 0;
while (\( i != 0 \)) {
    z = z + 1;
    i = i - 1;
}
Postcondition: \( x = z \)
Choosing Decrementing Function

What is a suitable decrementing function?

Precondition: $x \geq 0 \land y = 0$
while $(x \neq y)$ {
    $y = y+1$;
}  
Postcondition: $x = y$

Choosing Decrementing Function

What is a suitable decrementing function?

Precondition: $n \geq 0$
i = 0;
$r = 1$;
while $(i < n)$ {
i = i+1;
r = r*i;
}  
Postcondition: $r = n!$

Reasoning About Loops is Difficult

- For non-looping code, weakest precondition enables proof
- For loops, we have to guess
  - The loop invariant
  - The decrementing function
- Then use the proof techniques we discussed
- If the proof doesn’t work
  - Maybe you chose wrong invariant/function: Fix
  - Maybe the loop is incorrect!

Reasoning About Loops, Recap

total correctness = partial correctness + termination
1. Partial correctness
   - “Guess” and prove loop invariant
   - Loop exit condition and loop invariant must imply the postcondition
   - This gives us: “If the loop terminated then the postcondition holds”. But does the loop terminate?
2. Termination
   - (Roughly) “Guess” decrementing function $D \geq 0$. Each iteration decrements $D$, $D$ at its minimum value (usually 0) must imply loop exit condition
Example: Another Factorial

Precondition: \( t \geq 0 \)

\[ r = 1; \]
\[ n = t; \]
\[ \text{while} \ (n \neq 0) \ { \}
\[ \quad r = r*n; \]
\[ \quad n = n-1; \]
\[ \} \]
Postcondition: \( r = t! \)

Example: Integer Division \( x/y \)

Precondition: \( x \geq 0 \land y \geq 0 \)

\[ r = x; \]
\[ q = 0; \]
\[ \text{while} \ (y \leq r) \ { \}
\[ \quad r = r-y; \]
\[ \quad q = q+1; \]
\[ \} \]
Postcondition: \( x = y*q + r \land r < y \)

Example: Greatest Common Divisor

Precondition: \( x_1 > 0 \land x_2 > 0 \)

\[ y_1 = x_1; \]
\[ y_2 = x_2; \]
\[ \text{while} \ (y_1 \neq y_2) \ { \}
\[ \quad \text{if} \ (y_1 > y_2) \ { \}
\[ \quad \quad y_1 = y_1-y_2 \]
\[ \quad \} \text{else} \ { \}
\[ \quad \quad y_2 = y_2-y_1; \]
\[ \} \]
Postcondition: \( y_1 = \text{gcd}(x_1,x_2) \)

Why Study Formal Proofs

- Helps us write correct programs!
- Most of the time, our loops don’t need proofs
  
  \[ \text{for} \ i \ \text{in} \ \text{seq}: \ \text{print} \ i \]

- Sometimes we have to write more complex loops
  
  - Establish precondition and postcondition
  - Establish a loop invariant
  - Reason about correctness and termination