Loop Invariants
Annotating Programs

• General intuition behind annotations: label points in program with assertions that should hold when control is at that point!
  – You can do this using your intuition
  – Strong postconditions / weakest preconditions give you a systematic way to generate these assertions
  – In many cases (e.g. assignment, statement blocks, if-then-else) strongest postconditions / weakest preconditions can computed automatically!

• When is an annotation of a piece of code complete and correct?
  – An annotation is complete if every statement in the code has both a precondition and a postcondition (these will be shared: the postcondition of one statement will be a precondition of the following statement)
  – An annotation is correct if every embedded Hoare triple is valid

• If an annotation is complete and correct, then the Hoare triple consisting of the precondition of the code, the code itself, and the postcondition is valid!
\[
\{ Q \ [X \mapsto a] \} \quad X := a \quad \{ Q \}
\]

(by hoare_asgn)

\[
\begin{align*}
\{ P \} & \quad c_1 & \{ Q \} \\
\{ Q \} & \quad c_2 & \{ R \}
\end{align*}
\]

(by hoare_seq)

\[
\{ P \} \quad c_1; c_2 \quad \{ R \}
\]

(by hoare_seq)

\[
\begin{align*}
\{ P \land b \} & \quad c_1 \quad \{ Q \} \\
\{ P \land \neg b \} & \quad c_2 \quad \{ Q \}
\end{align*}
\]

(by hoare_if)

\[
\{ P \} \quad \text{if } b \text{ then } c_1 \text{ else } c_2 \text{ end} \quad \{ Q \}
\]
Recall: Three Key Concepts in Systematic Annotation Construction

• Strongest postconditions
• Weakest preconditions
• *Loop invariants*
Annotations and Loops

- Strongest postconditions / weakest preconditions still exist for loops!
- However, they cannot generally be computed automatically
- *Loop invariants* fill this gap
  - They are propositions
  - They must be added manually in Dafny
  - Once added, Dafny can check that they really are invariants!
Defining “Loop Invariant”

• Let code $S$ be while $B \{ S' \} \ (\{ S' \}$ is the loop body)
• Then a proposition $I$ is a **loop invariant** for $S$ if and only if $\{ I \land B \} S' \ \{ I \}$ is valid
  – If you start $S'$ in a state satisfying $I$ and loop condition $B$ ...
  – ... then whenever $S'$ terminates the result state satisfies $I$!
• This means that as the loop “loops”, $I$ is being kept true
• Also: if $I$ is a loop invariant for $S$ then $\{ I \} S \ \{ I \land \neg B \}$ is valid
  – If loop terminates then $B$ must be false (so $\neg B$ must be true)
  – Since loop body keeps $I$ true, when loop exists $I \land \neg B$ must hold!
\[
\{Q \ [X \mapsto a]\} \ X:=a \ {Q}\]

\[
\{P\} \ c_1 \ {Q}\]
\{Q\} \ c_2 \ {R}\]
\[
\{P\} \ c_1; c_2 \ {R}\]

\[
\{P /\ b\} \ c_1 \ {Q}\]
\{P /\ \sim b\} \ c_2 \ {Q}\]
\[
\{P\} \ if \ b \ then \ c_1 \ else \ c_2 \ end \ {Q}\]

\[
\{P /\ b\} \ c \ {P}\]
\[
\{P\} \ while \ b \ do \ c \ end \ {P /\ \sim b}\]
Loop Invariants in Dafny

```dachn
method FindMinVal (a : array<int>.) returns (min : int)
    requires a.Length > 0 // Precondition
    ensures forall i : int :: 0 <= i < a.Length ===> min <= a[i] // Postcondition
{
    min := a[0];
    var i := 1;
    while (i < a.Length)
    {
        invariant
        {
            if a[i] < min {
                min := a[i];
            }
            i := i+1;
        }
    }
}
```

- Declared with keyword “invariant” after loop invocation, before body
- You can have as many invariant declarations as you like; multiple invariants are interpreted as being conjoined
Loop Invariants in Dafny

```dsharp
method FindMinVal (a : array<int>) returns (min : int)
    requires a.Length > 0  // Precondition
    ensures forall i : int :: 0 <= i < a.Length ==> min <= a[i]  // Postcondition
{
    min := a[0];
    var i := 1;
    while (i < a.Length)
        invariant forall j : int :: 0 <= j < i ==> min <= a[j]
        {
            if a[i] < min {
                min := a[i];
            }
            i := i+1;
        }
}
```

- Declared with keyword “invariant” after loop invocation, before body
- You can have as many invariant declarations as you like; multiple invariants are interpreted as being conjoined
Strengthening Invariants

- Sometimes Dafny complains that it cannot complete the verification of a given invariant
- Often you can add extra invariants to give facts to Dafny that it needs
Adding Invariants

method FindMinVal (a : array<int>) returns (min : int)
    requires a.Length > 0 // Precondition
    ensures forall i : int :: 0 <= i < a.Length ==> min <= a[i] // Postcondition
{
    min := a[0];
    var i := 1;
    while (i < a.Length)
        invariant 0 <= i <= a.Length // Extra invariant to constrain i
        invariant forall j : int :: 0 <= j < i ==> min <= a[j]
        {
            if a[i] < min {
                min := a[i];
            }
            i := i+1;
        }
}

• Dafny could not complete the previous proof because it did not know that $i \leq a.Length$
is preserved by the loop
• Adding this enables completion of verification
Another Example

method Search (key : int, a : array<int>) returns (found : bool)
  ensures found <= exists i : int :: 0 <= i < a.Length && key == a[i]
  {
    var i : int := 0;
    found := false;
    while (i < a.Length)
    
      invariant i <= a.Length;
      invariant found <= exists j : int :: 0 <= j < i && key == a[j]
      {
        if (key == a[i])
        {
          found := true;
        }
        i := i+1;
      }
  }
Yet Another Example

method Locate (key : int, a : array<int>) returns (found : bool, index : int)
  ensures -1 <= index < a.Length
  ensures found ==> index >= 0 && key == a[index]
  ensures !found ==> index == -1
{
  var i : int := 0;
  found := false;
  index := -1;
  while (i < a.Length)
    invariant i <= a.Length
    invariant found ==> key == a[index]
    invariant (!found) ==> index == -1
    {
      if (key == a[i])
      {
        return true, i;
      }
      i := i+1;
    }
}
Verifying Methods in Dafny

• Add requires, ensures clauses
• Add invariants to all loops
• If it verifies, you are done!
• Otherwise
  – Strengthen / weaken invariants
  – Strengthen requires, ensures
  – Start constructing the annotation on your own to see if that helps Dafny