A. Why

• Diverging programs aren’t useful, so it’s useful to know how to show that loops terminate.

B. Objectives

At the end of this class you should understand

• The loop bound method of ensuring termination.
• How to extend proofs of partial correctness to total correctness.

C. Loop Divergence

• Aside from runtime errors, the other way that programs don’t terminate is that they diverge (run forever). For our programs, that means infinite loops.
  • (For programs with recursion, we also have to worry about infinite recursion, but the discussion here is adaptable, especially if you remember that a loop is simply an optimized tail-recursive function.)
  • For some loops, we can ensure termination by calculating the number of iterations left. E.g., at each loop test, \( k := 0; \text{while } k < n \text{ do } \ldots; k := k + 1 \text{ od} \) has \( n - k \) iterations left.
  • But in general, we can’t calculate the number of iterations for all loops (see theory of computation course for uncomputable functions).
  • But we don’t need the exact number of iterations — it’s sufficient to find a decreasing upper bound expression \( t \) for the number of iterations. This \( t \) is a logical expression (we’re not planning to actually evaluate it at runtime). It can contain variables from the program or proof, which is why \( t \) is also often called the bound function.

• Syntax: We’ll attach the upper bound expression \( t \) to a loop using the syntax \{bd t\}.

• To show convergence of the loop \{inv p\} \{bd t\} while B do S od \{p ∧ ¬B\}, it’s sufficient for the bound expression \( t \) to meets the two following properties:
  • \( p \rightarrow t \geq 0 \)
    • The invariant guarantees that there is a nonnegative number of iterations left to do.
  • \( \{p ∧ B ∧ t = t_0\} S \{p ∧ t < t_0\} \) where \( t_0 \) is a fresh logical variable.
    • If you compare the value of the bound expression at the beginning and end of the loop body, you find that the value has decreased. I.e., if you were to print out the value \( t \) at each while test, you would find a strictly decreasing sequence of nonnegative integers.
- The variable \( t_0 \) is a logical variable (we don't actually calculate it at runtime). We're using it in the correctness proof to name the value of \( t \) before running the loop body. It should be a fresh variable (one we're not already using) to avoid clashing with existing variables.
- (Note: To get full total correctness, we also have to avoid runtime errors, which we saw in an earlier class.)

**Example 1:** For the \( \text{sum}(0, n) \) program, we can use \( n - k \) for the bound:

\[
\begin{align*}
\{n \geq 0\} & \quad k := 0; \quad s := 0; \\
\{\text{inv} \} & \quad 0 \leq k \leq n \land s = \text{sum}(0, k) \\
\{\text{bd} n - k\} & \quad \text{while } k < n \text{ do } k := k + 1; \quad s := s + k \text{ od} \\
\{s = \text{sum}(0, n)\}
\end{align*}
\]

- At the loop test, we always have \( \geq 0 \) iterations left: \( (p \rightarrow n - k \geq 0) \) because \( p \) implies \( 0 \leq k \leq n \).
- Execution of the loop body lowers the bound. Let \( t_0 \) be our fresh logical variable, then we need \( \{p \land k < n \land n - k = t_0\} \text{ loop body } \{n - k < t_0\} \). Since the loop body includes \( k := k + 1 \), we know this is true: \( \{n - k = t_0\} \{n - (k + 1) < t_0\} k := k + 1 \{n - k < t_0\} \) by the assignment's \( \text{wp} \), with precondition strengthening.

**Hidden Requirements for a Bound Expression**

- The two properties we need a bound expression to have (being nonnegative and decreasing with each iteration) imply that bound expressions have some hidden requirements to meet.

- **The bound expression can't be a constant**, since constants don't change values.
  - **Example 2:** For the loop \( k := 0; \text{while } k < n \text{ do } ; \text{ } k := k + 1 \text{ od} \), people often make an initial guess of "n" for the bound expression instead of \( n - k \). When \( k = 0 \), the upper bound is indeed \( n - k = n \), but as \( k \) increases, the number of iterations left decreases.

- **A nonnegative bound can't imply that the loop test holds:** If \( B \) is the while loop test, then \( t \geq 0 \rightarrow B \) causes divergence: Since \( p \rightarrow t \geq 0 \), if \( t \geq 0 \rightarrow B \), then \( p \rightarrow B \), so \( B \) is true at every loop test.
  - There's no requirement that when \( B \) is false, \( t \) must be zero. It's allowed but not required.
  - Similarly, \( t > 0 \rightarrow B \) is allowed, but it's not required.

- **\( p \land B \rightarrow t > 0 \) is required:** When \( p \) and \( B \) hold, we run the loop body, which should decrease \( t \) but leave it \( \geq 0 \). Equivalently, \( p \land t = 0 \rightarrow \neg B \) because there's no room for the loop body to decrease \( t \), therefore we'd better not be able to do that iteration.

**Non-Requirements for a Bound Expression**

- It's often the case that people think the bound expression has to have certain properties that, though nice to have, are in fact not required.
- First, we're only trying to prove termination; we're not figuring out the asymptotic running time, so the upper bound doesn't have to be tight.
• **Not required:** We don’t require \( t – 1 \) to not be an upper bound. More generally, using big-O notation, we don’t need the running time to be \( \in \Theta(t) \), just \( \in O(t) \).
  • The bound expression doesn’t have to ever become zero.

• **Not required:** \( p \land \neg B \rightarrow t = 0 \).
  • The bound expression doesn’t have to decrease by exactly one:

• **Not required:** \( \{ p \land B \land t = t_0 \} \text{ loop body } \{ t = t_0 - 1 \} \)

  • **Example 3:** For binary search, if \( L \) and \( R \) are the left and right endpoints of the search and \( p \rightarrow L < R \), then \( R - L \) is a perfectly fine upper bound even though ceiling(\( \log_2(R-L) \)) is tighter.

D. Heuristics For Finding A Bound Expression

• To find a bound expression \( t \), there’s no algorithm but there are some guidelines.
  • First, start with \( t = 0 \).
  • For each (? some?) variable \( x \) that the loop body decreases, add \( x \) to \( t \).
  • For each (? some?) variable \( y \) that the loop body increases, subtract \( y \) from \( t \).
  • If \( t < 0 \) is possible, try to find some large expression \( e \) such that \( e + t \geq 0 \).
    • (A large constant might be helpful here.)

  • **Example 4:** For a loop that sets \( k := k - 1 \), try \( k \) (i.e., \( 0 + k \)) for \( t \).
    • If the invariant allows \( k < 0 \), then we need add something to \( t \) to make it larger. E.g., if the invariant implied \( k \geq -10 \), then it would imply \( k + 10 \geq 0 \), we could add \( 10 \) to our candidate bound and get \( t + 10 \).

  • **Example 5:** For a loop that sets \( k := k + 1 \), try \((-k)\) (i.e., \( 0 - k \)) for \( t \).
    • If \((-k)\) can be \(< 0\), we should add something to \((-k)\). E.g., if the invariant implies \( k \leq e \), then it implies \( e - k \geq 0 \), so maybe adding \( e \) to \( t \) will help.

E. Increasing and Decreasing Loop Variables

• We’ve looked at the simple summation loop
  \[
  \{ n \geq 0 \} \; k := 0; \; s := 0; \\
  \{ \text{inv} \; p = 0 \leq k \leq n \land s = \text{sum}(0,k) \} \{ \text{bd} \; n - k \} \\
  \text{while} \; k < n \; \text{do} \\
  \quad k := k + 1; \\
  \quad s := s + k \\
  \text{od} \\
  \{ s = \text{sum}(0, n) \}
  \]

  • **A first bound function:**
    • Using our heuristic, since \( k \) and \( s \) are increasing, \(-k-s\) is a candidate bound function that fails because it’s negative.
    • For terms to add to \(-k-s\) to make it nonnegative, we know \( n-k \geq 0 \) because the invariant includes \( k \leq n \), so let’s add \( n \) and get \( n-k-s \).
But $n - k - s$ can be negative, so we want to add some expression $e$ such that $e + n - k - s \geq 0$.

The invariant doesn't give an explicit bound for $s$, but from algebra we know that $0 + 1 + 2 + \ldots + n$ grows quadratically, and it's easy to verify that $n^2 - s \geq 0$ for all $n \in \mathbb{N}$.

This gives $n^2 + n - k - s$ as a bound function.

- **A second and third bound function**
  - Since $n - k \geq 0$ and decreases as $k$ increases, it by itself works as a bound function.
  - Similarly, $n^2 - s \geq 0$ and decreases as $s$ increases, so it works by itself as a bound function too.

- **Modifications to bound functions**
  - Bound functions are not unique: If $t$ is a bound expression, then so is $at^n + b$ for any positive $a$, $b$, and $n$. Similarly, if $t_1$ and $t_2$ are bound functions separately, then $t_1 + t_2$ and $t_1 \times t_2$ are also bound functions. So it's possible to encounter $n^2 + n - k - s$ by finding $n - k$ and $n^2 - s$ individually and then adding them.

F. **Another Loop Example: Iterative GCD**

- Not all loops modify only one loop variable with each iteration: Some modify multiple variables, with some being modified sometimes and others being modified another time.

- **Definition:** For $x, y \in \mathbb{N}$, $x, y > 0$, the greatest common divisor of $x$ and $y$, written $gcd(x, y)$, is the largest value that divides both $x$ and $y$ evenly (i.e., without remainder).
  - E.g., $gcd(300, 180) = gcd(2^2 \times 3 \times 5^2, 2^2 \times 3^2 \times 5) = 2^2 \times 3 \times 5 = 60$.

- Some useful $gcd$ properties:
  - If $x = y$, then $gcd(x, y) = x = y$
  - If $x > y$, then $gcd(x, y) = gcd(x - y, y)$
  - If $y > x$, then $gcd(x, y) = gcd(x, y - x)$
  - E.g., $gcd(300, 180) = gcd(120, 180)$, $gcd(120, 60) = gcd(60, 60) = 60$.

- Here's a minimal proof outline for an iterative $gcd$-calculating loop:

  ```
  \{ x > 0 \land y > 0 \land x = x_0 \land y = y_0 \}
  \{ inv \ p = x > 0 \land y > 0 \land gcd(x_0, y_0) = gcd(x, y) \} \quad \{ bd \ ??? \} / / to be filled\-in
  while x \neq y do
      if x > y then x := x - y else y := y - x fi
  od
  \{ x = gcd(x_0, y_0) \}
  ```

- Here's a full proof outline for partial correctness.

  ```
  \{ x > 0 \land y > 0 \land x = x_0 \land y = y_0 \}
  \{ inv \ p = x > 0 \land y > 0 \land gcd(x_0, y_0) = gcd(x, y) \} \quad \{ bd \ ??? \} / / to be filled\-in
  while x \neq y do
      \{ p \land x \neq y \}
  ```

© James Sasaki, 2020
if \( x > y \) then
\[ \{ p \land x \neq y \land x > y \} \{ p[x-x/y] \} \ x := x - y \ \{ p \} \]
else
\[ \{ p \land x \neq y \land x \leq y \} \{ p[y-x/y] \} \ y := y - x \ \{ p \} \]
fi
\[ \{ p \land x = y \} \{ x = gcd(x_0, y_0) \} \]
• We have a number of predicate logic obligations
  • \((x > 0 \land y > 0 \land x = x_0 \land y = y_0) \rightarrow p\)
  • \( p \land x \neq y \land x > y \rightarrow p[x-x/y] \)
  • \( p \land x \neq y \land x \leq y \rightarrow p[y-x/y] \)
  • \( p \land x = y \rightarrow x = gcd(x_0, y_0) \)
• With \( p = x > 0 \land y > 0 \land gcd(x_0, y_0) = gcd(x, y) \), the substitutions are
  • \( p[x-x/y] = x-y \land \land y > 0 \land gcd(x_0, y_0) = gcd(x-y, y) \)
  • \( p[y-x/y] = x > 0 \land y-x > 0 \land gcd(x_0, y_0) = gcd(x, y-x) \)
• (There are other full outline expansions, for example, one using the wp of the entire if–fi, which is
  • \((p \land x \neq y) \rightarrow ((x > y \rightarrow p[x-x/y]) \land (x \leq y \rightarrow p[y-x/y]))\)
  • But these other outlines produce logic obligations of roughly the same proof difficulty.
• What about convergence?
  • The loop body contains code that makes both \( x \) and \( y \) smaller, so our heuristic gives us \( x+y \) as a candidate bound function. Non–negativity is easy to show: the invariant implies \( x, y > 0 \), so \( x+y \geq 0 \).
  • Reduction of \( x+y \) is slightly subtle: Though the loop body doesn’t always reduce \( x \) or always reduce \( y \), it always reduces one of them, so \( x+y \) is always reduced.
• So our final minimally–annotated program is
  \[ \{ x > 0 \land y > 0 \land x = x_0 \land y = y_0 \} \quad // \quad x_0 \text{ and } y_0 \text{ are the initial values of } x \text{ and } y \]
  \[ \{ \text{inv} \ p = x > 0 \land y > 0 \land gcd(x_0, y_0) = gcd(x, y) \} \]
  \[ \{ \text{bd} \ x+y \} \]
while \( x \neq y \) do
  if \( x > y \) then \( x := x - y \) else \( y := y - x \) fi
od
\[ \{ x = gcd(x_0, y_0) \} \]
• We can add the material in blue below to fill in the full outline for total correctness.
  \[ \{ x > 0 \land y > 0 \land x = x_0 \land y = y_0 \} \]
  \[ \{ \text{inv} \ p = x > 0 \land y > 0 \land gcd(x_0, y_0) = gcd(x, y) \land x+y \geq 0 \} \]
  \[ \{ \text{bd} \ x+y \} \]
while \( x \neq y \) do
  \[ \{ p \land x \neq y \land x+y = t_0 \} \]
if \( x > y \) then
\[
\{ p \land x \neq y \land x > y \land x + y = t_0 \} \{ p[x-y/x] \land (x-y)+y < t_0 \} \ x := x-y \ \{ p \land x+y < t_0 \}
\]
else
\[
\{ p \land x \neq y \land x \leq y \land x + y = t_0 \} \{ p[y-x/y] \land x + (y-x) < t_0 \} \ y := y-x \ \{ p \land x+y < t_0 \}
\]
fi
\[
\{ p \land x+y < t_0 \}
\]
end
\[
\{ p \land x = y \}
\]
•

G. Semantics of Convergence
•

Here’s a semantic assertion about bound functions and loop termination.

• Lemma (Loop Convergence): Let \( W = \{ \text{inv } p \} \{ \text{bnd } t \} \text{ while } B \text{ do } S \text{ od} \) be an loop annotated with an invariant and bound function. Assume we can prove partial correctness of \( W \) and total correctness of \( \{ p \land B \land t = t_0 \} S \{ p \land t < t_0 \} \). Then if \( \sigma \models p \land B \land t = t_0 \), then \( \bot \notin M(W, \sigma) \). (Proof omitted.)

H. Total Correctness of a Loop
•

To show total correctness of \( \{ p_0 \} S_0; \{ \text{inv } p \} \{ \text{bd } t \} \text{ while } B \text{ do } S \text{ od } \{ q \} \) we need
  • Partial correctness: \( \models \{ p_0 \} S_0; W \{ q \} \), where \( W \) is the loop proper.
    1. Initialization establishes the invariant: \( \{ p_0 \} S_0 \{ p \} \) (typically, because \( p_0 \rightarrow \text{wp}(S_0, p) \) or \( \text{sp}(p_0, S_0) \rightarrow p \)).
    2. The loop body maintains the invariant: \( \{ p \land B \} S \{ p \} \)
    3. The loop establishes the final postcondition: \( p \land \neg B \rightarrow q \)
  • Termination: \( \models_{\text{tot}} \{ p_0 \} S_0; W \{ T \} \)
    4. No runtime errors during initialization (\( p_0 \rightarrow D(S_0) \)) or during loop evaluation: (\( p \rightarrow D(B) \)) and (\( p \land B \rightarrow D(S) \)).
    5. No divergence: The bound function is nonnegative (\( p \rightarrow t \geq 0 \)) and evaluation of the loop body decreases the bound: \( \{ p \land B \land t = t_0 \} S \{ t < t_0 \} \).
•

For a formal proof rule, let’s concentrate on the loop itself and not worry about initialization or finalization. This leaves partial correctness (line 2 above) and termination (lines 4 and 5 above).

While Loop (Total Correctness)

1. \( p \rightarrow D(B) \land (B \rightarrow D(S)) \)
2. \( p \rightarrow t \geq 0 \)
3. \( \models \{ p \land B \land t = t_0 \} S \{ p \land t < t_0 \} \)
4. \( \models_{\text{tot}} \{ \text{inv } p \} \{ \text{bnd } t \} \text{ while } B \text{ do } S \text{ od } \{ p \land \neg B \} \) while 1, 2, 3
Loop Convergence & Total Correctness

CS 536: Science of Programming version 2

3/25 numerous updates

A. Why

- Runtime errors make our programs not work, so we want to avoid them.
- Diverging programs aren't useful, so it's useful to know how to show that loops terminate.

B. Objectives

At the end of this activity you should be able to

- Calculate the domain predicate of an expression.
- Show what domain predicates need to hold within a program.
- Generate possible loop bounds for a given loop.
- State the extra obligations required to prove that a partially correct program is totally correct.

C. Questions

1. Consider the triple \{inv p\} \{bd e\} while k < n do ... k := k+1 od \{p \land k \geq n\}. Assume p \rightarrow n \geq k.

   To show that this loop terminates, we need

   (1) p \rightarrow n - k \geq 0 (which holds by assumption) and

   (2) \{p \land k < n \land e = t_0\} ... ; k := k+1 \{e < t_0\}

   a. Does condition (2) hold if e = n - k?  (If so, we can use n-k as a bound expression.)

   b. Can we use n-k+1 as a bound expression?

   c. Can we use 2n-k as a bound expression?

2. Use the same program as in Question 3 but assume p \rightarrow n \geq k-3, not n \geq k.

   a. Why does n-k now fail as a bound expression?

   b. Give an example of a bound expression that does work.

3. Consider the loop below. (Assume n is a constant and the omitted code does not change k.)

   a. Why does using just k as the bound function fail?

   b. Find an expression that involves k and prove that it's a loop bound.  (You'll need to augment p.)

   \{n \geq -1\}

   k := n;

   \{inv p \land _____ \} \{bd _____ \}

   while k \geq -1

   do ... k := k-1 ... od
4. What is the minimum expression (i.e., closest to zero) that can be used as a loop bound for
   \{inv \ n \leq x+y\ \{bd \ldots\} \ while \ x+y > n \ do \ldots \ y := y-1 \ od?\}
   (Assume \(x\) and \(n\) are constant.)

5. Consider the loop \(\{n > 0\} \ k := n; \{inv \ ???\} \ while \ k > 1 \ do \ldots \ k := k/2 \ od \{\ldots\}\)
a. Argue that \(\text{ceiling}(\log_2 k)\) is a loop bound. (Augment the invariant as necessary.)
b. Argue that \(k\) is a loop bound.
c. Argue that \(\text{ceiling}(\log_2 n)\) is \textbf{not} a loop bound. (Trick question.)

6. Let's look at the general problem of convergence of \(\{inv \ p\} \ while \ B \ do \ S \ od \{q\}\). For each property below, briefly discuss whether it is (1) required, (2) allowable but not required, or (3) incompatible with the requirements.
a. \(p \rightarrow t \geq 0\)
b. \(t < 0 \rightarrow \neg p\)
c. \(\{p \land B \land t = t_0\} \ S \{t = t_0 - 1\}\)
d. \(p \land t \geq 0 \rightarrow B\)
e. \(\neg B \rightarrow t = 0\)
f. \(\{p \land B \land t = t_0\} \ S \{t < t_0\}\)
Solution to Practice 18 (Loop Termination)

1. (Termination of \{inv p\} \{bd n-k\} while \(k < n\) do \(k := k + 1\) od)
   a. Yes: \(p \land k < n \land n - k = t_0\) \(\Rightarrow\) \(\{n - (k+1) < t_0\} k := k + 1 \{n - k < t_0\}\) requires \(n - (k+1) < n - k\) which is true.
   b. Yes: Decrementing \(k\) certainly decreases \(n - k + 1\), and \(n - k + 1 > n - k \geq 0\), which is the other requirement.
   c. Yes, but only if \(n \geq 0\): We know \(n-k \geq 0\), so \(2n - k \geq 0\), which is \(\geq 0\) if \(n \geq 0\). (If \(n < 0\) then \(2n - k\) might be negative.)

2. If \(n \geq k - 3\), then we only know \(n - k \geq -3\). (Note \(n - k + 3\) works as a bound, however.)

3. (Decreasing loop variable)
   a. We can't just \(k\) as the bound expression because we don't know \(k \geq 0\). In fact, the loop terminates with \(k = -2\).
   b. Since \(k\) is initialized to \(n\), we can add \(-2 \leq k \leq n\) to the invariant and use \(k+2\) as the bound expression.
   c. We need to know that the invariant implies \(k+2 \geq 0\) and that the loop body decreases \(k+2\).

4. The smallest loop bound is \(x + y - n\). We know it's \(\geq 0\) because \(n \leq x + y\), and we know it decreases by \(1\) each iteration, so at loop termination, \(x + y - n = 0\), which implies that nothing less than \(x + y - n\) can work as a bound.

5. (\(\Theta(\log n)\) loop)
   a. Add \(0 \leq k \leq n\) to the invariant. Since \(k > 1\), we know \(\text{ceiling}(\log_2 k) > 0\), and halving \(k\) decreases \(\text{ceiling}(\log_2 k)\) by one and \(\text{ceiling}(\log_2 k) - 1 \geq 0\). Thus \(\text{ceiling}(\log_2 k)\) works as a loop bound.
   b. Since \(k > 1\), halving \(k\) decreases it but leaves it \(\geq 0\).
   c. \(\text{ceiling}(\log_2 n)\) doesn't decrease because \(n\) is a constant. (Constants make terrible bounds :-)

6. (Loop convergence) Required are (a) \(p \rightarrow t \geq 0\), (b) \(t < 0 \rightarrow \neg p\) [i.e., the contrapositive of (a)], and (f) \(\{p \land B \land t = t_0\} S \{t < t_0\}\). Property (c) \(\{p \land B \land t = t_0\} S \{t = t_0 - 1\}\) is allowable but not required: It implies (f) but is stronger than we need. Property (e) \(\neg B \rightarrow t = 0\) is allowable but not required. Property (d) \(p \land t \geq 0 \rightarrow B\) is incompatible with the requirements (it would cause an infinite loop).