One More Inference Rules for Loop-free Statement under Total Correctness

There is actually one more inference rule that needs to be updated for total correctness:

- **Nondeterministic Conditional Rule** (total correctness version):
  1. $\{p \wedge B_1\}\, S_1\, \{q_1\}$
  2. $\{p \wedge B_2\}\, S_2\, \{q_2\}$
  3. $\{p \wedge (B_1 \vee B_2)\}\, \textbf{if}\ B_1 \rightarrow S_1 \,\square\, B_2 \rightarrow S_2\ \textbf{fi}\, \{q_1 \vee q_2\}$ $\qquad$ $\text{if} - \text{fi } 1{,}2$

At least one of the guards needs to be true or else we will have a runtime error.

Proof under Total Correctness 2: Proof of Convergence

- Remember that we only create proofs for provable triples. If a statement contains a loop that can be proved to be convergent, then this loop must iterate for a finite number of iterations.
  - For example, let $S \equiv k \coloneqq 0; \textbf{while}\ k < 5\ \textbf{do}\ k \coloneqq k + 1\ \textbf{od}$, then while loop iterates 5 iterations. When k gets increased to 2, including the current iteration, the loop still has 3 iterations.

- In general, we cannot always find the exact number of (remaining) iterations a loop has, but if we can upper bound the number of (remaining) iterations of a loop by a finite natural number and the number decreases after each iteration, then we can prove the loop converges.
  - For example, let $W \equiv \textbf{while}\ k < n\ \textbf{do}\ k \coloneqq k + 1\ \textbf{od}$, we don't have an exact number of iterations; but we can use $n - k$ to express the number of (remaining) iterations before W terminates, and $n - k$ is decreasing after each iteration.

- A **bound expression** or **bound function** t for a loop is a finite ***natural number*** expression that, at each loop test, gives a ***strictly decreasing*** upper bound on the number of iterations remaining before termination. A bound expression can use program variables (variables in statement) and logical variables (variables in precondition or postcondition).
- We'll attach the bound expression t to a loop using the syntax $\{\textbf{bd}\ t\}$, thus in a proof outline for total correctness, a loop usually looks like: $\{\textbf{inv}\ p\}\{\textbf{bd}\ t\}\ \textbf{while}\ B\ \textbf{do}\ S\ \textbf{od}$.

(Properties of Bound Expressions)
What properties do we need for t so that it can be bound for loop $W \equiv \{\textbf{inv}\ p\}\ \{\textbf{bd}\ t\}\ \textbf{while}\ B\ \textbf{do}\ S\ \textbf{od}$? Here we list two requirements.

- $p \Rightarrow t \geq 0$.
  - Loop invariant p is always true after each iteration of a while loop, it needs to logically imply that there are non-negative iterations left to do.

- $\vdash_{tot} \{p \wedge B \wedge t = t_0\}\, S\, \{p \wedge t < t_0\}$
  - t_0 is the value of t before the execution of one iteration; after the execution of this iteration, the value of t needs to be strictly decreased to guarantee convergence.

1. Show a partial proof outline under total correctness to show that the loop in the following program converges and p is a correct loop invariant.

$\{n \geq 0\}$
$k \coloneqq 0;\ s \coloneqq 0;$
$\{\mathbf{inv}\ p \equiv 0 \leq k \leq n \wedge s = sum(0\,,k)\}$
$\{\mathbf{bd}\ ?\}$ $\mathbf{while}\ k < n\ \mathbf{do}$
$\quad s \coloneqq s + k + 1; k \coloneqq k + 1$
$\mathbf{od}$
$\{s = sum\ (0, n)\}$

- First, let's look for a bound function, it needs to be non-negative, and it needs to be reduced after each iteration. $n - k$ is an easy choice: loop invariant implies that $n \geq k$ and k is increased by 1 after each iteration.

- We can get the following partial proof outline including only conditions about the loop.

$\{n \geq 0\}$
$k \coloneqq 0;\ s \coloneqq 0;$
$\{\mathbf{inv}\ p \equiv 0 \leq k \leq n \wedge s = sum(0\,,k)\}$
$\{\mathbf{bd}\ n - k\}$ $\mathbf{while}\ k < n\ \mathbf{do}$
$\quad \{p \wedge k < n \wedge n - k = t_0\}$
$\quad \{p\ [k + 1\ /\ k][s + k + 1\ /\ s] \wedge n - (k + 1) < t_0\}$
$\quad s \coloneqq s + k + 1; \{p[k + 1\ /\ k] \wedge n - (k + 1) < t_0\}$
$\quad k \coloneqq k + 1$
$\quad \{p \wedge n - k < t_0\}$
$\mathbf{od}$
$\{p \wedge k \geq n\}$
$\{s = sum\ (0, n)\}$

2. Let $W \equiv \{\mathbf{inv}\ p\}\ \{\mathbf{bd}\ t\}\ \mathbf{while}\ B\ \mathbf{do}\ S\ \mathbf{od}.$ Decide true or false for each of the following statements about loop bound expressions.
   a. t can be a constant.
   False. A constant cannot be reduced after each iteration. So, in the example $\mathbf{while}\ k < 5\ \mathbf{do}\ k \coloneqq k + 1\ \mathbf{od}$, we cannot use 5 as a bound expression.

   b. $t \geq 0 \Rightarrow B$.
   False. Since $p \Rightarrow t \geq 0$, so if $t \geq 0 \Rightarrow B$ then $p \Rightarrow B$ after each iteration, which means the loop diverges and it is a contradiction to the existence of t.

   c. $p \wedge B \Rightarrow t > 0$.
   True. If $p \wedge B$, then we will enter the loop body, which means there is at least another iteration; thus t is strictly greater than 0 since its value needs to be reduced by at least 1 after this iteration.

   d. $p \wedge t = 0 \Rightarrow \neg B$.
   True. This is the contrapositive of c.

   e. $\vDash_{tot} \{p \wedge B \wedge t = t_0\}\ S\ \{t = t_0 - 1\}$
   False. We don't require t reduce by exactly 1 after each iteration. For example, in a binary search t is reduced by half after each iteration.

   f. Let N be an expression showing the exact number of remaining iterations of W, then N is $\Theta(t)$.

False. We don't require t is a tight upper bound of N. For example, N can be $n - k$, but t can be 2^{n-k}.

g. $p \land \neg B \Rightarrow t = 0$.
False. We don't require t to go down to 0 after the last iteration.

- There is no algorithm to find a bound expression for a loop. But here we have some heuristics based on the requirements for bound expression:
  a) We start with $t \equiv 0$.
  b) In the loop body, if the value of a variable x gets decreased then we CAN concatenate $+x$ to t; if the value of a variable y gets increased then we CAN concatenate $-y$ to t.
  c) If p does not logically imply $t \geq 0$, adjust the constant for each term or add constant terms to the expression.

3. Find a bound expression for the following loops:
   a. $\{\mathbf{inv}\ n \leq x + y\}\{\mathbf{bd}\ ?\}\ \mathbf{while}\ x + y > n\ \mathbf{do}\ y \coloneqq y - 1\ \mathbf{od}$
   - Following the first two steps of the above guidelines, we can try to use $0 + y$ as a bound expression. It is strictly decreased after each iteration but there is no evidence that $y \geq 0$. From loop invariant we know that $n \leq x + y$, so $x + y - n \geq 0$; x and n doesn't change after each iteration, so $x + y - n$ strictly decrease after each iteration. So, $x + y - n$ is a bound expression.

   b. $\{\mathbf{inv}\ n \leq 2y - x\}\{\mathbf{bd}\ ?\}\ \mathbf{while}\ y > n + x\ \mathbf{do}\ y \coloneqq y - 1\ \mathbf{od}$
   - Similarly, we can try to use y as a bound expression but there is no evidence that $y \geq 0$. From loop invariant we know that $n \leq 2y - x$, so $2y - x - n \geq 0$; x and n doesn't change after each iteration, so and $2y - x - n$ is a bound expression.

   c. $\{\mathbf{inv}\ x + y < n\}\{\mathbf{bd}\ ?\}\ \mathbf{while}\ x + y < n\ \mathbf{do}\ y \coloneqq y - 1; x \coloneqq x + 2\ \mathbf{od}$
   - In each iteration, x is increased and y is decreased; we can try to use $-x + y$ as a bound expression but there is no evidence showing that $-x + y \geq 0$. Even if we adjust the constant, we still cannot find an expression with $-x$ and $+y$ that's both non-negative and decreasing. Then we noticed that $(x + y)$ as whole increases after each iteration, and the loop invariant tell us $n - (x + y) > 0$; so $n - (x + y)$ is a bound expression.

4. For $x, y \in \mathbb{Z}^+$, the greatest common divisor of x and y, written as $\gcd(x, y)$ is largest positive integer that divides both x and y. For example, $\gcd(300, 180) = 60$. We usually use the Euclidean algorithm to find $\gcd(x, y)$:

$$\gcd(x, y) = \begin{cases} x \text{ or } y, & \text{if } x = y \\ \gcd(x - y, y), & \text{if } x > y \\ \gcd(x, y - x), & \text{if } x < y \end{cases}$$

   For example, $\gcd(300, 180) = \gcd(120, 180) = \gcd(120, 60) = \gcd(60, 60) = 60$.
   Create a program for the above algorithm, then give its full proof outline under total correctness.

   - Let's start with the statement itself; the value of $\gcd(x, y)$ stored in variables x and y after the program

$$\mathbf{while}\ x \neq y\ \mathbf{do}\ \mathbf{if}\ x > y\ \mathbf{then}\ x \coloneqq x - y\ \mathbf{else}\ y \coloneqq y - x\ \mathbf{fi}\ \mathbf{od}$$

   - Then let's add precondition and postcondition:

$$\{x > 0 \land y > 0 \land x = x_0 \land y = y_0\}$$
$$\mathbf{while}\ x \neq y\ \mathbf{do}\ \mathbf{if}\ x > y\ \mathbf{then}\ x \coloneqq x - y\ \mathbf{else}\ y \coloneqq y - x\ \mathbf{fi}\ \mathbf{od}$$

$\{x = y = \text{gcd}(x_0, y_0)\}$

- What is true before and after each iteration? First, $x > 0$ and $y > 0$ are always true. Also, we want show that $\text{gcd}(x, y) = \text{gcd}(x_0, y_0)$ is always true. Thus, we can get the following minimal proof out line:

  $\{x > 0 \land y > 0 \land x = x_0 \land y = y_0\}$
  $\{\textbf{inv}\ x > 0 \land y > 0 \land \text{gcd}(x, y) = \text{gcd}(x_0, y_0)\}$
  $\textbf{while}\ x \neq y\ \textbf{do}$
  $\quad \textbf{if}\ x > y\ \textbf{then}\ x \coloneqq x - y\ \textbf{else}\ y \coloneqq y - x\ \textbf{fi}$
  $\textbf{od}$
  $\{x = y = \text{gcd}(x_0, y_0)\}$

- Then we expand this proof outline to a full proof outline under partial correctness.

  $\{x > 0 \land y > 0 \land x = x_0 \land y = y_0\}$
  $\{\textbf{inv}\ p \equiv x > 0 \land y > 0 \land \text{gcd}(x, y) = \text{gcd}(x_0, y_0)\}$
  $\textbf{while}\ x \neq y\ \textbf{do}$
  $\quad \{p \land x \neq y\}$
  $\quad \textbf{if}\ x > y\ \textbf{then}$
  $\quad\quad \{p \land x \neq y \land x > y\}\ \{p[x - y\ /\ x]\}\ x \coloneqq x - y\ \{p\}$
  $\quad \textbf{else}$
  $\quad\quad \{p \land x \neq y \land x \leq y\}\ \{p[y - x\ /\ y]\}\ y \coloneqq y - x\ \{p\}$
  $\quad \textbf{fi}$
  $\quad \{p\}$
  $\textbf{od}$
  $\{p \land x = y\}$
  $\{x = y = \text{gcd}(x_0, y_0)\}$

  In the above proof outline, red is for Loop Rule and blue is for Conditional Rule 1 and Backward Assignment.

- We can see that all the conditions in the proof outline are safe, and the statement itself cannot create errors, so we don't need to add any domain predicates.

- Now we need to find a bound expression. Both x and y in the loop condition can be decrease after each iteration, so we can try $x + y$ as the bound: $x + y > 0$ and no matter we go to true or false branch in each iteration, its value is always decreased. Then we have the following full proof outline under total correctness.

  $\{x > 0 \land y > 0 \land x = x_0 \land y = y_0\}$
  $\{\textbf{inv}\ p \equiv x > 0 \land y > 0 \land \text{gcd}(x, y) = \text{gcd}(x_0, y_0)\}$
  $\{\textbf{bd}\ x + y\}$
  $\textbf{while}\ x \neq y\ \textbf{do}$
  $\quad \{p \land x \neq y \land x + y = t_0\}$
  $\quad \textbf{if}\ x > y\ \textbf{then}$
  $\quad\quad \{p \land x \neq y \land x > y\ \land x + y = t_0\}$
  $\quad\quad \{p[x - y\ /\ x]\ \land (x - y) + y < t_0\}\ x \coloneqq x - y\ \{p \land x + y < t_0\}$
  $\quad \textbf{else}$
  $\quad\quad \{p \land x \neq y \land x \leq y \land x + y = t_0\}$
  $\quad\quad \{p[y - x\ /\ y]\ \land x + (y - x) < t_0\}\ y \coloneqq y - x\ \{p \land x + y < t_0\}$
  $\quad \textbf{fi}$
  $\quad \{p \land x + y < t_0\}$
  $\textbf{od}$

$\{p \wedge x = y\}$
$\{x = y = \gcd(x_0, y_0)\}$

In the above proof outline, green is for bound expression.

- To finish the above proof outline, we still need to show the following logical implications:
  - $x > 0 \wedge y > 0 \wedge x = x_0 \wedge y = y_0 \Rightarrow p$
  - $p \wedge x = y \Rightarrow\ x = y = \gcd(x_0, y_0)$
  - $p \wedge x \neq y \wedge x > y\ \wedge x + y = t_0 \wedge x > y \Rightarrow\ p[x - y\ /\ x]\ \wedge (x - y) + y < t_0$
  - $p \wedge x \neq y \wedge x \leq y \wedge x + y = t_0 \Rightarrow\ p[y - x\ /\ y]\ \wedge x + (y - x) < t_0$

  Here we omitted these arguments, but they are all true. Please try to prove them after class.

Loop Rule under Total Correctness*

- In our assignments and exams, there won't be questions that need the total correctness version of the While Loop Rule: possible runtime errors and possible divergence won't appear in the same proof under total correctness.

- To show that a loop $W \equiv \{\textbf{inv}\ p\}\{\textbf{bd}\ t\}\ \textbf{while}\ B\ \textbf{do}\ S\ \textbf{od}$ is totally correct, we need:
  a) p is a loop invariant and p is safe.
  b) No runtime error while evaluating B or executing S: $p \Rightarrow D(B)$ and $p \wedge B \Rightarrow D(S)$
  c) p can logically imply the bound expression t being at least 0 and safe.
  d) Loop bound t is decreased after each iteration: $\vdash_{tot} \{p \wedge B \wedge t = t_0\}\ S\ \{p \wedge t < t_0\}$.

  Thus, we have
  While Loop Rule (total correctness version):
  1. $p \Rightarrow D(B)$ # Here p is a safe predicate
  2. $p \wedge B \Rightarrow D(S)$
  3. $p \Rightarrow \downarrow (t \geq 0)$
  4. $\{p \wedge B \wedge t = t_0\}\ S\ \{p \wedge t < t_0\}$
  5. $\{\textbf{inv}\ p\}\{\textbf{bd}\ t\}\ \textbf{while}\ B\ \textbf{do}\ S\ \textbf{od}\ \{p \wedge \neg B\}$ loop 1,2,3,4

As an aside, since "possible runtime errors and possible divergence won't appear in the same proof under total correctness" in our assignments and exams, so we assume that $D(B) \equiv T$, $D(S) \equiv T$ and $D(t) \equiv T$, so the above rule will the simplified to the following version:

While Loop Rule (total correctness version for no possible runtime errors):
1. $p \Rightarrow t \geq 0$
2. $\{p \wedge B \wedge t = t_0\}\ S\ \{p \wedge t < t_0\}$
3. $\{\textbf{inv}\ p\}\{\textbf{bd}\ t\}\ \textbf{while}\ B\ \textbf{do}\ S\ \textbf{od}\ \{p \wedge \neg B\}$ loop 1,2

You will find line 1 and 2 are exactly the two requirements we need for bound expression.