Inference Rules for Loop-free Statement under Total Correctness

- Previously, we have learned how to get $wp$ from $wlp$: we include domain predicates for expressions $D(e)$, predicates $D(p)$, and statements $D(S)$ that guarantee the evaluation of $e, p$ and $S$ won't cause a runtime error.
- We use a similar idea to get $\vdash_{tot} \{p'\}\ S\ \{q'\}$ from $\vdash \{p\}\ S\ \{q\}$ by including domain predicates:
  - In general, for a loop-free $S$:
    - if we get $\vdash \{wlp(S, q)\}\ S\ \{q\}$, then for total correctness we have $\vdash_{tot} \{wp(S, q \wedge D(q))\}\ S\{q \wedge D(q)\}$ .
    - if we get $\vdash \{p\}\ S\ \{sp(p, S)\}$, then for total correctness we have $\vdash_{tot} \{p \wedge D(p) \wedge D(S)\}\ S\ \{sp(p \wedge D(p) \wedge D(S), S)\}$.

- For a predicate $p$, we define the **safe version** of $p$ as $\downarrow p \equiv p \wedge D(p)$.
  - With this definition we can rewrite $wp(S, q) \Leftrightarrow D(S) \wedge \downarrow wlp(S, q)$

- If a predicate $p \Leftrightarrow \downarrow p$, then we say $p$ **is safe**.
- In a triple $\{p\}\ S\ \{q\}$, if $p$ and $q$ are both safe, then $\{p\}\ S\ \{q\}$ is **safe**. **In a proof of total correctness, we only prove safe triples.**
- If we are asked to prove a triple with unsafe precondition or postcondition under total correctness, it's not doable. All we can do is to add domain predicates to make the triple safe first. In general, for triple $\{p\}\ S\ \{q\}$, its **safe version** is $\{\downarrow p \wedge D(S)\}\ S\ \{\downarrow q\}$.

- We have the following non-loop axioms / inference rules for total correctness. Note that the antecedent triples and consequent triples in each rule are valid under total correctness.
  - **Skip Axiom** (total correctness version):

    $\{\downarrow p\}\ \mathbf{skip}\ \{\downarrow p\}$ skip

  - **Backward Assignment Axiom** (total correctness version):

    $\{D(e) \wedge \downarrow wlp(v := e, \downarrow q)\}\ v := e\ \{\downarrow q\}$ backward assignment

  - **Forward Assignment Axion** (total correctness version):

    $\{D(e) \wedge \downarrow p\}\ v := e\ \{(D(e) \wedge \downarrow p)[v_0\ /\ v] \wedge v = e[v_0\ /\ v]\}$ forward assignment

  - Sequence rule, strengthen precondition rule, weaken postcondition rule, conjunction rule, disjunction rule, conditional rule 1, nondeterministic conditional rule is still the same. The only difference is that now the antecedent triples and the consequent triples are safe and provable under total correctness.
    For example, conjunction rule:
    1. $\{p_1\}\ S\ \{q_1\}$
    2. $\{p_2\}\ S\ \{q_2\}$
    3. $\{p_1 \wedge p_2\}\ S\ \{q_1 \wedge q_2\}$ conjunction 1,2

    This rule still holds in the proof for total correctness, but now $\vdash_{tot} \{p_1\}\ S\ \{q_1\}$, $\vdash_{tot} \{p_2\}\ S\ \{q_2\}$, and $p_1, q_1, p_2, q_2$ are all safe, thus the consequent $\vdash_{tot} \{p_1 \wedge p_2\}\ S\ \{q_1 \wedge q_2\}$, and $p_1 \wedge p_2, q_1 \wedge q_2$ are safe.

  - **Conditional Rule 2** (total correctness version):
    1. $\{p_1\}\ S_1\ \{q_1\}$
    2. $\{p_2\}\ S_2\ \{q_2\}$
    3. $\{D(B) \wedge (B \rightarrow p_1) \wedge (\neg B \rightarrow p_2)\}\ \mathbf{if}\ B\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2\ \mathbf{fi}\ \{q_1 \vee q_2\}$ if − else 1,2

    What's special about conditional rule 2 is that it contains a fresh expression $B$ in the consequent.

1. Create a full proof outline for $\{b[x] < y\}\ y := y + 1; z := y\ \{q\}$ under total correctness. Find a possible $q$.
   - First, we add domain predicate in the precondition, then we can use forward assignment to find a provable $q$.

   - We can create the following proof outline:
     $\{b[x] < y \wedge 0 \leq x < size(b)\}$
     $y := y + 1; \{b[x] < y_0 \wedge 0 \leq x < size(b) \wedge y = y_0 + 1\}$
     $z := y\ \{b[x] < y_0 \wedge 0 \leq x < size(b) \wedge y = y_0 + 1 \wedge z = y\}$

2. Under total correctness, create a full proof outline for provable triple $\{p\}$ **if** $x \geq 0$ **then** $x := sqrt(x)$ **else** $x := y/b[k]$ **fi** $\{0 \leq x < y\}$. Find a possible $p$.
   - The postcondition is already safe.
   - If we use $wp$ of the whole conditional statement as $p$, we might not be able to prove it, so here we use Conditional Rule 2 to create $p$.

   - We can create the following proof outline:
     $\{p\}$
     **if** $x \geq 0$ **then**
     $\{0 \leq sqrt(x) < y \wedge x \geq 0\}\ x := sqrt(x)\ \{0 \leq x < y\}$
     **else**
     $\{0 \leq y/b[k] < y \wedge 0 \leq k < size(b) \wedge b[k] \neq 0\}\ x := y/b[k]\ \{0 \leq x < y\}$
     **fi** $\{0 \leq x < y\}$

     Where $p \equiv (x \geq 0 \rightarrow 0 \leq sqrt(x) < y \wedge x \geq 0) \wedge (x < 0 \rightarrow 0 \leq y/b[k] < y \wedge 0 \leq k < size(b) \wedge b[k] \neq 0)$

- Examples 1,2 have other solutions, here I only provide one of the possible proofs.

3. Under total correctness, create a full proof outline for provable triple $\{T\}$ **if** $x \geq 0$ **then** $x := sqrt(x)$ **else** $x := y/b[k]$ **fi** $\{q\}$. Find a possible $q$.
   - $T$ is a safe precondition, but the program has the probability to create a runtime error so we cannot take all states into consideration. To create a proof under total correctness, we need to strengthen the precondition to $\{T \wedge D(IF)\} \equiv T \wedge (x \geq 0 \rightarrow x \geq 0) \wedge (x < 0 \rightarrow 0 \leq k < size(b) \wedge b[k] \neq 0)$
     $\Leftrightarrow x < 0 \rightarrow 0 \leq k < size(b) \wedge b[k] \neq 0$
     $\Leftrightarrow x \geq 0 \vee 0 \leq k < size(b) \wedge b[k] \neq 0$

   - We can use Conditional Rule 1 to create the following proof outline:
     $\{x \geq 0 \vee 0 \leq k < size(b) \wedge b[k] \neq 0\}$
     **if** $x \geq 0$ **then**
     $\{(x \geq 0 \vee 0 \leq k < size(b) \wedge b[k] \neq 0) \wedge x \geq 0\}\ \{x \geq 0\}\ x := sqrt(x)\ \{x_0 \geq 0 \wedge x = sqrt(x_0)\}$
     **else**
     $\{(x < 0 \rightarrow 0 \leq k < size(b) \wedge b[k] \neq 0) \wedge x < 0\}$
     $\{0 \leq k < size(b) \wedge b[k] \neq 0\}\ x := y/b[k]\ \{0 \leq k < size(b) \wedge b[k] \neq 0 \wedge x = y/b[k]\}$
     **fi**
     $\{x_0 \geq 0 \wedge x = sqrt(x_0) \vee\ 0 \leq k < size(b) \wedge b[k] \neq 0 \wedge x = y/b[k]\}$

Proof of Convergence with Loop Bound

- Remember that we only prove validity 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 body runs $5$ iterations.

- In general, we cannot always find the exact number of iterations a loop runs, but if we can upper bound the number of (remaining) iterations of a loop by a finite natural number, 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 Functions

- What properties do we need for $t$ so that it can be a bound for loop $W \equiv \{\textbf{inv } p\}\ \{\textbf{bd } t\}\ \textbf{while } B \textbf{ do } S \textbf{ od}$ ? Here we list two requirements.
- $\textbf{inv } p \Rightarrow t \geq 0$.
  - Loop invariant $p$ is always true after each iteration of a while loop, it needs to logically implies 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.

4. Show a partial proof outline under total correctness to show that the loop in the following program converges and $p_1$ is a correct loop invariant.

$\{n \geq 0\}$
$k \coloneqq 0;\ s \coloneqq 0;$
$\{\textbf{inv } p_1 \equiv 0 \leq k \leq n \wedge s = sum(0\,, k)\}$
$\{\textbf{bd } ?\}\ \textbf{while } k < n \textbf{ do}$
$\quad s \coloneqq s + k + 1; k \coloneqq k + 1$
**od**
$\{s = sum\ (0, n)\}$

  - First, let's look for a bound function, it needs to be non-negative and reduces after each iteration. 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.

$\{n \geq 0\}$
$k \coloneqq 0;\ s \coloneqq 0;$
$\{\textbf{inv } p_1 \equiv 0 \leq k \leq n \wedge s = sum(0\,, k)\}$
$\{\textbf{bd } n - k\}\ \textbf{while } k < n \textbf{ do}$

$\{p_1 \wedge k < n \wedge n - k = t_0\}$
$\{p_1[k+1 \,/\, k][s+k+1 \,/\, s] \wedge n - (k+1) < t_0\}$
$s \coloneqq s + k + 1; \{p_1[k+1 \,/\, k] \wedge n - (k+1) < t_0\}$
$k \coloneqq k + 1$
$\{p_1 \wedge n - k < t_0\}$

**od**

$\{p_1 \wedge k \geq n\}$

$\{s = sum\ (0, n)\}$

5. Let $W \equiv \{\mathbf{inv}\ p\}\ \{\mathbf{bd}\ t\}$ **while** $B$ **do** $S$ **od.** Decide true or false for each of the following statements about loop bound expressions.

   a. $t$ cannot be a constant.
   True. A constant cannot reduce after each iteration. So, in Example 4, we cannot use $n$ 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 equivalent to c.

   e. $\vDash \{p \wedge B \wedge t = t_0\}\ S\ \{t = t_0 - 1\}$
   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 the number of iterations of $W$, then $N$ is $O(t)$.
   True. $t$ is the upper bound of $N$.

   g. Let $N$ be the number of iterations of $W$, then $N$ is $\Theta(t)$.
   False. We don't require $t$ is a tight upper bound of $N$.

   h. $p \wedge \neg B \Rightarrow t = 0$.
   False. We don't require $t$ to go down to $0$ after

- There is no algorithm to find a bound expression for a loop. But here we have some guidelines 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 $x$ who appears in loop condition $B$ gets decreased then we add "$+x$" to $t$; if the value of a variable $y$ who appears in loop condition $B$ gets increased then we add "$-y$" to $t$.
  c) Adjust the constant for each term or add constant terms to the expression if $t$ can be negative.

6. 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 $y$ 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 good 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 $2y - x - n \geq 0$, and $2y - x - n$ is a good 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 use the help from the loop invariant, we 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 $n - (x + y) > 0$; so $n - (x + y)$ is a good bound expression.