Proving Termination of Textual Programs

Next, we consider methods for proving termination of textual programs.

A termination annotated statement is an annotated statement, where each annotation position now holds a pair \( \{p, \delta\} \) where \( p \) is an assertion and \( \delta \) is a ranking function mapping states into a well-founded domain \( \langle A, > \rangle \).

For example, following is a termination annotated version of program INT-MULTIPLY:

\[
\begin{array}{l}
\{true, x_1\} (u_1, u_2, z) := (x_1, x_2, 0) \\
\{true, u_1\} \textbf{while } u_1 \neq 0 \textbf{ do} \\
\quad \begin{cases}
\{u_1 > 0, u_1 \div 2\} \textbf{ if } \text{odd}(u_1) \textbf{ then} \\
\quad \{\text{odd}(u_1), u_1 \div 2\} (u_1, z) := (u_1 - 1, z + u_2) \\
\quad \{\text{even}(u_1), u_1 \div 2\} (u_1, u_2) := (u_1 \div 2, 2u_2) \\
\end{cases} \\
\{true, u_1\}
\end{array}
\]

A termination annotated statement \( \{p_1, \delta_1\} \overline{S} \{p_2, \delta_2\} \) generates a set of verification conditions and a set of descent conditions. The definition of the generated verification conditions is identical to their definition for the case of partial correctness.
Generated Descent Conditions

Each termination annotated statement generates a set of descent conditions, as follows:

- Each annotated assignment \( \{ p_1, \delta_1 \} \bar{y} := \bar{c} \{ p_2, \delta_2 \} \) generates the descent condition:
  \[ D(\{ p_1, \delta_1 \} \bar{y} := \bar{c} \{ p_2, \delta_2 \} ) = \{ p_1 \rightarrow \delta_1 \geq \delta_2[\bar{c}/\bar{y}] \} \]

- Each annotated concatenation \( \{ p_1, \delta_1 \} S_1 \{ p_2, \delta_2 \} S_2 \{ p_3, \delta_3 \} \) generates the descent conditions:
  \[ D(\{ p_1, \delta_1 \} S_1 \{ p_2, \delta_2 \} S_2 \{ p_3, \delta_3 \} ) = D(\{ p_1, \delta_1 \} S_1 \{ p_2, \delta_2 \} ) \cup D(\{ p_2, \delta_2 \} S_2 \{ p_3, \delta_3 \} ) \]

- The descent conditions generated by the annotated conditional \( \{ p_0, \delta_0 \} [\text{if } c \text{ then } \{ p_1, \delta_1 \} S_1 \text{ else } \{ p_2, \delta_2 \} S_2] \{ p_3, \delta_3 \} \) are:
  \[ D(\{ p_0, \delta_0 \} [\text{if } c \text{ then } \{ p_1, \delta_1 \} S_1 \text{ else } \{ p_2, \delta_2 \} S_2] \{ p_3, \delta_3 \} ) = \{ p_0 \land c \rightarrow \delta_0 \geq \delta_1, p_0 \land \neg c \rightarrow \delta_0 \geq \delta_2 \} \cup \]
  \[ D(\{ p_1, \delta_1 \} S_1 \{ p_3, \delta_3 \} ) \cup D(\{ p_2, \delta_2 \} S_2 \{ p_3, \delta_3 \} ) \]

- Each annotated while statement \( \{ p_0, \delta_0 \} [\text{while } c \text{ then } \{ p_1, \delta_1 \} S_1] \{ p_2, \delta_2 \} \) generates the descent conditions:
  \[ D(\{ p_0, \delta_0 \} [\text{while } c \text{ then } \{ p_1, \delta_1 \} S_1] \{ p_2, \delta_2 \} ) = \{ p_0 \land c \rightarrow \delta_0 \geq \delta_1, p_0 \land \neg c \rightarrow \delta_0 \geq \delta_2 \} \cup D(\{ p_1, \delta_1 \} S_1 \{ p_0, \delta_0 \} ) \]
**Example: Program** INT-MULTIPLY

Reconsider the termination annotated program **INT-MULTIPLY**:

\[
\{true, x_1\} (u_1, u_2, z) := (x_1, x_2, 0) \\
\{true, u_1\} \textbf{ while } u_1 \neq 0 \textbf{ do} \\
\begin{cases} \\
\{u_1 > 0, u_1 \div 2\} \textbf{ if } \text{odd}(u_1) \textbf{ then} \\
\{\text{odd}(u_1), u_1 \div 2\} (u_1, z) := (u_1 - 1, z + u_2) \\
\{\text{even}(u_1), u_1 \div 2\} (u_1, u_2) := (u_1 \div 2, 2u_2) \\
\end{cases} \\
\{true, u_1\}
\]

This program gives rise to the following verification conditions:

\[D_0 : \quad true \quad \rightarrow \quad x_1 \geq u_1[x_1, x_2, 0/u_1, u_2, z]\]

\[D_1^+ : \quad u_1 \neq 0 \quad \rightarrow \quad u_1 > (u_1 \div 2)\]

\[D_1^- : \quad u_1 = 0 \quad \rightarrow \quad u_1 \geq u_1\]

\[D_2^+ : \quad \text{odd}(u_1) \quad \rightarrow \quad (u_1 \div 2) \geq (u_1 \div 2)\]

\[D_2^- : \quad \neg\text{odd}(u_1) \quad \rightarrow \quad (u_1 \div 2) \geq (u_1 \div 2)\]

\[D_3 : \quad \text{odd}(u_1) \quad \rightarrow \quad (u_1 \div 2) \geq (u_1 - 1) \div 2\]

\[D_4 : \quad \text{even}(u_1) \quad \rightarrow \quad (u_1 \div 2) \geq u_1[u_1 \div 2/u_1]\]

all of which are valid.
Soundness of the Method

Let \( \{p_0, \delta_0\} \vdash \{p_n, \delta_n\} \) be a termination annotated program. Such annotation is said to be valid if all verification conditions and all descent conditions generated by \( \{p_0, \delta_0\} \vdash \{p_n, \delta_n\} \) are valid.

**Claim 28. [Soundness of termination proofs]** Let \( \{p_0, \delta_0\} \vdash \{p_n, \delta_n\} \) be a valid termination annotated program. Then \( P \) is partially correct w.r.t the specification \( \langle p_0, p_n \rangle \), and \( P \) is \( p_0 \)-convergent.

**Proof:** The only way for a textual program to produce a divergent computation is that the computation remains forever within a certain `while` loop. Since every repeated entry into this loop causes the ranking to decrease, and all other statements within the loop do not allow it to increase, such a divergent computation will generate an infinitely descending chain of values over a well-founded domain. Since this is impossible, we conclude that a program with valid termination annotation cannot generate a divergent computation. □
Textual Procedural Programs

A textual procedural program has the form

\[
\begin{bmatrix}
\text{Program } P_0(\bar{x}; \bar{z}) \\
S_0(\bar{x}; \bar{y}; \bar{z})
\end{bmatrix}
\quad \begin{bmatrix}
\text{Proc } P_1(\bar{x}; \bar{z}) \\
S_1(\bar{x}; \bar{y}; \bar{z})
\end{bmatrix}
\ldots
\begin{bmatrix}
\text{Proc } P_k(\bar{x}; \bar{z}) \\
S_k(\bar{x}; \bar{y}; \bar{z})
\end{bmatrix}
\]

where $\bar{x}$, $\bar{z}$ are the formal input and result parameters, respectively. $\bar{y}$ are the working variables. Each $S_i(\bar{x}; \bar{y}; \bar{z})$ for $i = 0,\ldots,k$ is a statement which, in addition to the regular statements (assignment, concatenation, conditional, and while) may also be a procedure call of the form $P_j(\bar{e}; \bar{v})$, where $j > 0$, $\bar{e}$ is a list of expressions serving as the actual input arguments, and $\bar{v} \subseteq \bar{y} \cup \bar{z}$ is a list of variables which serve as actual destinations of the results of the procedure call.

For example, following is a textual procedural program which computes the function $z = x!$.

\[
\begin{bmatrix}
\text{Program } P_0(x; z) \\
P_1(x; z)
\end{bmatrix}
\quad \begin{bmatrix}
\text{Proc } P_1(x; z) \\
\text{if } x = 0 \text{ then } z := 1 \\
\text{else} \\
P_1(x - 1; z) \\
z := z \cdot x
\end{bmatrix}
\]
Annotating Textual Procedural Programs

Procedural programs are annotated in a way similar to the annotation of while-programs. The only restriction is that the annotation of the body of module $P_i$ must have the form $\{\varphi_i(x)\} S_i \{\psi(x; z)\}$, for $i = 0, \ldots, k$.

For example, following is an annotation of the factorial program:

\begin{align*}
\textbf{Program} & \quad P_0(x; z) \\
& \{x \geq 0\} P_1(x; z) \{z = x!\}
\end{align*}

\begin{align*}
\textbf{Proc} & \quad P_1(x; z) \\
& \{x \geq 0\} \\
& \quad \text{if } x = 0 \text{ then } \{x = 0\} z := 1 \\
& \quad \{x > 0\} P_1(x - 1; z) \\
& \quad \text{else } \{x > 0 \land z = (x - 1)!\} \\
& \quad z := z \cdot x \\
& \{z = x!\}
\end{align*}
Verification Conditions for Annotated Procedural Programs

To the set of rules defining the verification conditions generated by an annotated statement, add the following:

- Each annotated procedure call \( \{p\} P_j(\vec{e}; \vec{v}) \{q\} \) generates the verification conditions:

\[
\text{Ver}(\{p\} P_j(\vec{e}; \vec{v}) \{q\}) = \{p \rightarrow \varphi_j(\vec{e}), \quad p \land \psi_j(\vec{e}; \vec{z}') \rightarrow q[\vec{z}' / \vec{v}]\}
\]

Claim 29. [Inductive Assertions for Procedural Textual Programs]

Let \( P \) be an annotated procedural program. If all the verification conditions for this program are valid, then \( P \) is partially correct w.r.t \( (\varphi_0, \psi_0) \) where \( \{\varphi_0\} S_0 \{\psi_0\} \) is the annotation of the body of the main module \( P_0(\vec{x}; \vec{z}) \).

The claim can be proven by reduction to procedural flow-graphs and showing that we obtain an inductive full assertion network.
Application to Program Factorial

The annotated program:

\[
\text{Program } P_0(x; z) \\
\{m_0 : x \geq 0\} P_1(x; z) \{z = x!\}
\]

\[
\text{Proc } P_1(x; z) \\
\{\ell_0 : x \geq 0\} \\
\quad \text{if } x = 0 \text{ then } \{\ell_1 : x = 0\} z := 1 \\
\quad \{\ell_2 : x > 0\} P_1(x - 1; z) \\
\quad \text{else } \{\ell_3 : x > 0 \land z = (x - 1)!\} \\
\quad \quad z := z \cdot x \\
\{z = x!\}
\]

generates the following verification conditions:

\[
\begin{align*}
V_{m0}^{in} : & \quad x \geq 0 & \rightarrow & \quad x \geq 0 \\
V_{m0}^{out} : & \quad x \geq 0 \land z' = x! & \rightarrow & \quad z' = x! \\
V_0^+ : & \quad x \geq 0 \land x = 0 & \rightarrow & \quad x = 0 \\
V_0^- : & \quad x \geq 0 \land \neg(x = 0) & \rightarrow & \quad x > 0 \\
V_{\ell_0} : & \quad x = 0 & \rightarrow & \quad 1 = x! \\
V_{\ell_2}^{in} : & \quad x > 0 & \rightarrow & \quad x - 1 \geq 0 \\
V_{\ell_2}^{out} : & \quad x > 0 \land z' = (x - 1)! & \rightarrow & \quad x > 0 \land z' = (x - 1)! \\
V_{\ell_3} : & \quad x > 0 \land z = (x - 1)! & \rightarrow & \quad z \cdot x = x!
\end{align*}
\]

all of which are valid.
Proving Termination of Procedural Programs

As for the case of non-procedural programs, we extend the annotations to a pair \( \{ p, \delta \} \) where \( p \) is an assertion and \( \delta \) is a ranking function mapping states into a well-founded domain \( \langle A, \succ \rangle \).

To the set of rules defining the descent conditions generated by an annotated statement, add the following:

- Each annotated procedure call \( \{ p_1, \delta_1 \} P_j(\vec{e}; \vec{v}) \), \( \{ p_2, \delta_2 \} \) generates the descent conditions:

\[
D(\{ p_1, \delta_1 \} P_j(\vec{e}; \vec{v}), \{ p_2, \delta_2 \}) = \begin{cases} 
 p_1 & \rightarrow \delta_1 \succ \delta_0(\vec{e}), \\
 p_1 \land \psi_j(\vec{e}; \vec{z}') & \rightarrow \delta_1 \succeq \delta_2[\vec{z}'/\vec{v}] 
\end{cases}
\]

Note that strict descent is only required on the entry conditions.
Apply to Factorial

The termination annotated program:

\[
\begin{align*}
\textbf{Program } & \quad P_0(x; z) \\
& \quad \{m_0 : x \geq 0, |x| + 1\} \\
P_1(x; z) & \quad \{\text{true}, |x|\}
\end{align*}
\]

\[
\begin{align*}
\textbf{Proc } & \quad P_1(x; z) \\
& \quad \{\ell_0 : x \geq 0, |x|\} \\
& \quad \text{if } x = 0 \text{ then } \{\ell_1 : x = 0, |x|\} z := 1 \\
& \quad \{\ell_2 : x > 0, |x|\} P_1(x - 1; z) \\
\text{else } & \quad \{\ell_3 : \text{true}, |x|\} \\
& \quad z := z \cdot x \\
& \quad \{\text{true}, |x|\}
\end{align*}
\]

generates the following descent conditions:

\[
\begin{align*}
D_{m_0}^{in} & : x \geq 0 \quad \rightarrow \quad |x| + 1 > |x| \\
D_{m_0}^{out} & : x \geq 0 \land \text{true} \quad \rightarrow \quad |x| + 1 \geq |x| \\
D^{+}_{\ell_0} & : x \geq 0 \land x = 0 \quad \rightarrow \quad |x| \geq |x| \\
D^{-}_{\ell_0} & : x \geq 0 \land \neg(x = 0) \quad \rightarrow \quad |x| \geq |x| \\
D_{\ell_1} & : x = 0 \quad \rightarrow \quad |x| \geq |x| \\
D_{\ell_2}^{in} & : x > 0 \quad \rightarrow \quad |x| > |x - 1| \\
D_{\ell_2}^{out} & : x > 0 \land \text{true} \quad \rightarrow \quad |x| \geq |x| \\
D_{\ell_3} & : \text{true} \quad \rightarrow \quad |x| \geq |x|
\end{align*}
\]

all of which are valid.
Soundness of the Proof Method

Claim 30. [Verifying Termination of Textual Procedural Programs]
Let \( P \) be a termination annotated procedural program, such that all verification conditions and all descent conditions are valid. Then \( P \) is \( p_0 \)-convergent where \( \{p_0, \delta_0\} S_0 \{p_n, \delta_n\} \) is the annotation of the body of main module \( P_0(\bar{x}; \bar{z}) \).

Proof:
Assume to the contrary, that the termination annotation of program \( P \) is valid, yet there exists a \( p_0 \)-divergent computation \( \sigma : s_0, s_1, \ldots \) of \( P \).

For an execution state \( s \), we define the level of state \( s \) to be the number of procedure calls minus procedure returns up to the occurrence of \( s \). Thus, the level of \( s \) expresses the depth of the recursion at this state. We consider two distinct cases.

In the first case, there exists an integer \( L \) such that \( \sigma \) contains infinitely many states at level \( L \). With no loss of generality, assume that \( L \) is the minimal level with this property. Let \( s^1, s^2, \ldots \) be the infinite subsequence of \( \sigma \) such that each \( s^i \) is at level \( L \), and there does not exist a state beyond \( s^1 \) with a level lower than \( L \). In this case, it can be shown that there exists a procedure \( P_j \) such that all states \( s_i \) belong to the same computation of \( P_j \). Such a computation can diverge only if all these states are contained within a certain while-loop of \( P_j \). Since the ranking strictly decreases on each reentry into the loop, this is impossible.

The remaining case is that there exists no level with infinitely many states of that level. This means that every level is visited only finitely many times. In this case, we should be able to identify a subsequence of states \( s^1, s^2, \ldots \) such that, for each \( s_i \) there exists no state later than \( s^i \) with the same level as \( s^i \). This implies
that state $s^i$ is a state executing some procedure call. If we compare the rankings $\delta^1, \delta^2, \ldots$ at these sequence of states, we find again (due to the strict decrease $D^{in}$ required on procedure call) an infinite descending sequence of ranks, which is impossible.

We conclude that program $P$ cannot have a divergent computation.