**Additional Examples: Integer Division**

\[ 2d \leq r \rightarrow [(d, b) := (2d, 2b)] \]

\[ \ell_1 \quad (q, r, d, b) := (0, x, y, 1) \quad \ell_0 \]

\[ \text{2d} > r? \]

\[ \ell_2 \quad b \neq 1 \rightarrow [(d, b) := (d \div 2, b \div 2)] \quad d > r? \quad d \leq r \rightarrow [(q, r) := (q + b, r - d)] \]

\[ \ell_3 \quad b = 1? \]

\[ \ell_4 \]

**Specification is**

\[ \varphi : x \geq 0 \land y > 0 \quad \psi : \quad x = qy + r \land 0 \leq r < y. \]

**Define**

\[ \chi : x = qy + r \quad \text{power2}(b) : \quad \exists m > 0 : b = 2^m \]

**As the cut-set we choose** \( \{\ell_0, \ell_1, \ell_3, \ell_4\} \). The assertion network is given by

\[ \varphi_0 : \quad x \geq 0 \land y > 0 \]

\[ \varphi_1 : \quad x = qy + r \land \text{power2}(b) \land d = yb \land 0 \leq r \]

\[ \varphi_3 : \quad x = qy + r \land \text{power2}(b) \land d = yb \land 0 \leq r \land 2d > r \]

\[ \varphi_4 : \quad x = qy + r \land 0 \leq r < y \]
Example: Bubble Sort

The assertion network is given by:

\[ \varphi_0 = p : \ n > 0 \]
\[ \varphi_5 = q : \text{sorted}(1, n) \]
\[ \varphi_2 : \ i \in [1..n] \land j \in [1..i] \land A[1..j-1] \leq A[j] \land \text{sorted}(i+1, n) \land A[1..i] \leq A[i+1..n] \]
Proving Convergence

All proofs of convergence rely on the construction of well founded ranking functions.

We define a well-founded domain to be a pair \((A, \succ)\) consisting of a domain \(A\) and an ordering relation \(\succ\) over \(A\) such that there does not exist an infinitely descending sequence

\[
a_0 \succ a_1 \succ \cdots
\]

of \(A\)-elements.

For example, the natural numbers with the \(\succ\) ordering forms a well-founded domain, denoted \((\mathbb{N}, \succ)\). When there is no danger of confusion, we refer to the well-founded domain \((A, \succ)\), simply as \(A\). For elements \(a, b \in A\), we write \(a \succeq b\) if either \(a \succ b\) or \(a = b\).
Composite Well-Founded Domains

Given two well-founded domains \((\mathcal{A}_1, \triangleright_1)\) and \((\mathcal{A}_2, \triangleright_2)\), we introduce two ways to construct a composite well-founded domain.

The cross product \(\mathcal{A}_1 \times \mathcal{A}_2\) is the well-founded domain \((\mathcal{A}, \triangleright)\), where \(\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2\) and

\[
(a_1, a_2) \triangleright (b_1, b_2) \iff (a_1 \triangleright_1 b_1 \land a_2 \triangleright_2 b_2) \lor (a_1 \triangleright_1 b_1 \land a_2 \triangleright_2 b_2)
\]

The lexicographic product \(\mathcal{A}_1 \times_{lex} \mathcal{A}_2\) is the well-founded domain \((\mathcal{A}, \triangleright)\), where \(\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2\) and

\[
(a_1, a_2) \triangleright_{lex} (b_1, b_2) \iff (a_1 \triangleright_1 b_1) \lor (a_1 = b_1 \land a_2 \triangleright_2 b_2)
\]

**Claim 10.** If both \((\mathcal{A}_1, \triangleright_1)\) and \((\mathcal{A}_2, \triangleright_2)\) are well-founded, then so are \(\mathcal{A}_1 \times \mathcal{A}_2\) and \(\mathcal{A}_1 \times_{lex} \mathcal{A}_2\).

**Proof**  It is sufficient to show that \(\mathcal{A}_1 \times_{lex} \mathcal{A}_2\) is well-founded.

Assume to the contrary, that there exists an infinitely descending sequence

\[
(a_1, b_1) \triangleright_{lex} (a_2, b_2) \triangleright_{lex} \cdots
\]

From the definition of \(\triangleright_{lex}\) it follows that the sequence of first pair members satisfies

\[a_1 \triangleright_1 a_2 \triangleright_1 \cdots\].

Since \(\mathcal{A}_1\) is well founded, it follows that there exists some position \(k\) such that \(a_k = a_{k+1} = \cdots\). Therefore, the sequence \(b_k \triangleright_2 b_{k+1} \triangleright_2 \cdots\) must be infinitely descending, contradicting the well-foundedness of \(\mathcal{A}_2\).
The Well-Founded Ranking Functions Method for Proving Convergence of Programs

The following claim outlines and prove the soundness of the ranking function method for verifying convergence.

**Claim 11. [Ranking Functions Method]** Let $P$ be a program with a pre-condition specification $p$. Let $\mathcal{N} : \langle \mathcal{C}, \{ \varphi_\ell \mid \ell \in \mathcal{C} \} \rangle$ be an assertion network, $(\mathcal{A}, \succ)$ be a well-founded domain, and $\{ \delta_\ell \mid \ell \in \mathcal{C} \}$ be a network of ranking functions, each mapping states into elements of $\mathcal{A}$. If the following requirements are satisfied:

1. The network $\mathcal{N}$ is inductive.
2. $p \rightarrow \varphi_{\ell_0}$
3. For every verification path $\pi$ connecting location $\ell_i$ to location $\ell_j$, the condition $\varphi_i(V) \land c_\pi(V) \rightarrow \delta_i(V) \succ \delta_j(f_\pi(V))$ is valid.

then program $P$ is $p$-convergent.

**Proof:** Assume to the contrary, that all requirements of the claim are met, yet there exists a divergent $p$-computation of the form

$$\sigma : \langle \ell_{i_0}, d_0 \rangle \xrightarrow{\pi_0} \langle \ell_{i_1}, d_1 \rangle \xrightarrow{\pi_1} \cdots \xrightarrow{\pi_{k-1}} \langle \ell_{i_k}, d_k \rangle \xrightarrow{\pi_k} \cdots$$

where we explicitly display the sequence of cut-points $\ell_0 = \ell_{i_0}, \ell_{i_1}, \ldots$ visited by $\sigma$ and the verification paths $\pi_0, \pi_1, \ldots$ connecting them.

We can show by induction that the infinite sequence

$$\delta_{\ell_{i_0}}(d_0) \succ \delta_{\ell_{i_1}}(d_1) \succ \cdots \succ \delta_{\ell_{i_k}}(d_k) \succ \cdots$$

is an infinite descending chain of well-founded elements, which is impossible. \[\square\]
Termination Networks

We refer to a network $\mathcal{N} = \langle \mathcal{C}, \{\varphi_\ell, \delta_\ell \mid \ell \in \mathcal{C}\} \rangle$, which assigns an assertion $\varphi_\ell$ and a ranking function $\delta_\ell$ to each cut-point $\ell \in \mathcal{C}$ as a termination network. A termination network which satisfies requirements 1 and 3 of Claim 11 is called a valid network.
Example: Integer Square Root

Consider program \textsc{int-square} with the specification pre-condition $p : x \geq 0$.

\[ (y_1, y_2, y_3) := (0, 0, 1) \]
\[ y_2 := y_2 + y_3 \]
\[ y_2 \geq x? \]
\[ y_2 \leq x \rightarrow [(y_1, y_3) := (y_1 + 1, y_3 + 2)] \]

As a well-founded domain we choose $(\mathbb{N}, >)$. For the cut-set, we choose \{\ell_0, \ell_2, \ell_3\}. As assertion and ranking networks, we choose

<table>
<thead>
<tr>
<th>$\ell_i$</th>
<th>$\varphi_{\ell_i}$</th>
<th>$\delta_{\ell_i}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\ell_0$</td>
<td>$x \geq 0$</td>
<td>$x + 2$</td>
</tr>
<tr>
<td>$\ell_2$</td>
<td>$y_3 &gt; 0 \land y_2 \leq x + y_3 + 2$</td>
<td>$</td>
</tr>
<tr>
<td>$\ell_3$</td>
<td>$1$</td>
<td>$0$</td>
</tr>
</tbody>
</table>

Examination of this example shows that checking descent along the paths $\ell_0 \rightarrow \ell_1 \rightarrow \ell_2$ and $\ell_2 \rightarrow \ell_3$ is completely unnecessary since such paths can appear at most once in any computation.
An Improved Method

To make use of the above observation, we introduce the notion of an acyclic decomposition.

An acyclic decomposition of a program $P$ is a sequence of subsets of the nodes of $P$, $K_1, \ldots, K_k$ such that $K_1, \ldots, K_k$ is a partition of all the locations of $P$ and, if there is an edge from a location $\ell_i \in K_i$ to a location $\ell_j \in K_j$ then, necessarily $i \leq j$. We refer to $K_1, \ldots, K_k$ as the components of the decomposition. A trivial case of a decomposition is that of taking $K_1 = P$ yielding a decomposition with a single component.

The other extreme is when we decompose the graph of $P$ into its maximal strongly connected components MSCC’s.

For a given acyclic decomposition, we say that a verification path is an intra-component path if it is fully contained in a single component of the decomposition.

**Claim 12. [Improved Version]** In the application of the ranking function method, it is sufficient to require that condition 3 of Claim 11 holds for intra-component verification paths.

**Proof:** Assume that we successfully managed to apply the improved method using the well-founded domain $(\mathcal{A}, \succ)$ with ranking functions $\{\delta_\ell \mid \ell \in C\}$. We extend the well-founded domain into $\tilde{\mathcal{A}} = [1..k] \times_{lex} \mathcal{A}$. As the extended ranking function, we take $\tilde{\delta}_\ell(V) = (k+1-i, \delta_\ell(V))$ for every location $\ell \in C$ which belongs to component $K_i$ in the acyclic decomposition. 

\[\blacksquare\]
Examples: The GCD Program

We will illustrate the method on several examples. Consider first the program for computing the gcd of two positive integers:

\[ (y_1, y_2) := (x_1, x_2) \]

\[ y_1 < y_2 \rightarrow y_2 := y_2 - y_1 \]

\[ y_1 > y_2 \rightarrow y_1 := y_1 - y_2 \]

\[ y_1 = y_2? \]

As the well-founded domain we choose \((\mathbb{N}, >)\). Decomposing the program into its MSCC yields the decomposition \(\{\ell_0\}, \{\ell_1\}, \{\ell_2\}\). It follows that the only intra-component verification paths are \(\ell_1 \rightarrow \text{left} \rightarrow \ell_1\) and \(\ell_1 \rightarrow \text{right} \rightarrow \ell_1\). Consequently, we propose the following assertion and ranking-function networks:

<table>
<thead>
<tr>
<th>(\ell_i)</th>
<th>(\varphi_{\ell_i})</th>
<th>(\delta_{\ell_i})</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\ell_0)</td>
<td>(x_1 &gt; 0 \land x_2 &gt; 0)</td>
<td>0</td>
</tr>
<tr>
<td>(\ell_1)</td>
<td>(y_1 &gt; 0 \land y_2 &gt; 0)</td>
<td>(</td>
</tr>
<tr>
<td>(\ell_2)</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

For example, the descent condition for the path \(\pi : \ell_1 \rightarrow \text{right} \rightarrow \ell_1\) is given by

\[
\varphi_1 \land \varphi_{c_\pi} \rightarrow |y_1 + y_2| > |(y_1 - y_2) + y_2|
\]

which is obviously valid.
Additional Example: Integer Division

2\(d \leq r \rightarrow [(d, b) := (2d, 2b)]\)

\(\ell_1\)

\(2d > r?\)

\(\ell_2\)

\(b \neq 1 \rightarrow [(d, b) := (d \div 2, b \div 2)]\)

\(d > r?\)  \(d \leq r \rightarrow [(q, r) := (q + b, r - d)]\)

\(\ell_3\)

\(b = 1?\)

\(\ell_4\)

As the cut-set we choose \(\{\ell_0, \ell_1, \ell_3, \ell_4\}\). An MSCC decomposition yields \(\{\ell_0\}, \{\ell_1\}, \{\ell_2, \ell_3\}, \{\ell_4\}\). Consequently, we take the termination network to be

<table>
<thead>
<tr>
<th>(\ell_i)</th>
<th>(\varphi_{\ell_i})</th>
<th>(\delta_{\ell_i})</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\ell_0)</td>
<td>(x \geq 0 \land y &gt; 0)</td>
<td>0</td>
</tr>
<tr>
<td>(\ell_1)</td>
<td>(b &gt; 0 \land d &gt; 0)</td>
<td>(</td>
</tr>
<tr>
<td>(\ell_3)</td>
<td>(b &gt; 0)</td>
<td>(</td>
</tr>
<tr>
<td>(\ell_4)</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

For example, the descent condition for the path \(\ell_1 \rightarrow \ell_1\) is

\[b > 0 \land d > 0 \land 2d \leq r \quad \rightarrow \quad |r - d| > |r - 2d|\]
Example: Bubble Sort

\[ \ell_0 \]

\[ i := n \]

\[ \ell_1 \]

\[ i = 0? \]

\[ j = i \rightarrow [i := i - 1] \]

\[ i \neq 0 \rightarrow [j := 1] \]

\[ \ell_2 \]

\[ j < i? \]

\[ j := j + 1 \]

\[ \ell_3 \]

\[ A[j] \leq A[j + 1]? \]


\[ \ell_4 \]

As the well-founded domain we take \( \mathbb{N} \times_{lex} \mathbb{N} \). The termination network is given by:

<table>
<thead>
<tr>
<th>( \ell_i )</th>
<th>( \varphi_{\ell_i} )</th>
<th>( \delta_{\ell_i} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \ell_0 )</td>
<td>( n &gt; 0 )</td>
<td>0</td>
</tr>
<tr>
<td>( \ell_2 )</td>
<td>( 1 \leq j \leq i \leq n ) (</td>
<td>i</td>
</tr>
<tr>
<td>( \ell_5 )</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Obviously, on the path \( \ell_2 \rightarrow \ell_1 \rightarrow \ell_2 \), the rank \(|i|\) decreases while \(|i - j|\) may possibly increase. On both paths of the form \( \ell_2 \rightarrow \ell_3 \rightarrow \ell_4 \rightarrow \ell_2 \) rank \(|i - j|\) decreases while \(|i|\) stays the same.
Dependence on the Cut-Set

As for the inductive assertion method, we proceed to show that the proof method of well-founded ranking is independent of the particular cut-set selected for the proof.

Claim 13. [Termination networks can be extended] Let $\mathcal{N} = \langle \mathcal{C}, \{\varphi_l, \delta_l \mid l \in \mathcal{C}\}\rangle$ be a $p$-valid termination network, and $\tilde{l} \notin \mathcal{C}$ a location not in $\mathcal{C}$. There exists a $p$-valid termination network over the extended cut-set $\tilde{\mathcal{C}} = \mathcal{C} \cup \{\tilde{l}\}$ which agrees with $\mathcal{N}$ on the assertions $\varphi_l$ for all $l \in \mathcal{C}$.

Proof: Assume that the well-founded domain used for the network $\mathcal{N}$ is $(A, \succ)$. The extended network has the form $\tilde{\mathcal{N}} = \langle \tilde{\mathcal{C}}, \{\psi_l, \tilde{\delta}_l \mid l \in \tilde{\mathcal{C}}\}\rangle$, where $\tilde{\mathcal{C}} = \mathcal{C} \cup \{\tilde{l}\}$. For all $l \in \mathcal{C}$, we take $\psi_l = \varphi_l$. For $\tilde{l}$, we take $\psi_{\tilde{l}} = \text{pre}(\tilde{l}, \mathcal{N})$ as defined in the proof of Claim 4.

As the well-founded domain, we take $A \times_{\text{lex}} \{0, 1\} \cup \{\bot\}$, where $\bot$ is a special element, considered to be smaller than any element in $A \times_{\text{lex}} \{0, 1\}$. For all $l \in \mathcal{C}$, we take $\tilde{\delta}_l = (\delta_l, 0)$. For $\tilde{l}$, we take

$$\tilde{\delta}_{\tilde{l}}(V) = \max_{\pi \in \Pi_{\tilde{l}, \tilde{\mathcal{C}}}} \left( \text{if } c_\pi(V) \text{ then } \delta_{\text{dest}(\pi)}(f_\pi(V), 1) \text{ else } \bot \right)$$

Thus, if there exists at least one path $\pi$ connecting $\tilde{l}$ to some $l \in \mathcal{C}$ such that $V$ satisfies $c_\pi$, then we take $\tilde{\delta}_{\tilde{l}}(V)$ to be the maximum of $\delta_{\text{dest}(\pi)}(f_\pi(V), 1)$ over all such paths. On the other hand, if for all such paths $\pi$ $V \not\models c_\pi$, then $\tilde{\delta}_{\tilde{l}}(V) = \bot$. 
Proof Continued

Invoking the proof of Claim 4, we establish that the assertion network \( \{ \psi_\ell \mid \ell \in \tilde{C} \} \) is inductive. It remains to show that the descent condition 3 holds for every verification path \( \pi \) in the cut-set \( \tilde{C} \). Again, it is sufficient to consider only “new” verification paths, i.e., paths which either depart or arrive to the new cut-point \( \tilde{\ell} \).

Paths departing from \( \tilde{\ell} \)

Let \( \pi_2 \) be a “new” verification path leading from \( \tilde{\ell} \) to \( \ell_2 \in C \), and let \( V \) be a data state such that \( V \models c_\pi \). Since \( \pi_2 \in \Pi_{\tilde{\ell}, C} \) and \( V \models c_\pi \), \( \pi \) is one of the paths over which the maximum is taken in the definition of \( \tilde{\delta}_\ell(V) \). It follows that

\[
\tilde{\delta}_\ell(V) \geq (\delta_{\ell_2}(f_{\pi_2}(V)), 1),
\]

which implies

\[
\tilde{\delta}_\ell(V) \succ (\delta_{\ell_2}(f_{\pi_2}(V)), 0) = \tilde{\delta}_{\ell_2}(f_{\pi_2}(V)).
\]

We can therefore conclude the descent condition for path \( \pi_2 \) which requires

3. \( \psi_\ell(V) \land c_{\pi_2}(V) \rightarrow \tilde{\delta}_\ell(V) \succ \tilde{\delta}_{\ell_2}(f_{\pi_2}(V)) \)
Proof Continued: Paths Arriving to $\tilde{\ell}$

Let $\pi_1$ be a "new" verification path leading from $\ell_1 \in C$ to $\tilde{\ell}$, and let $V_1$ be a data state satisfying $\psi_{\ell_1} \land c_{\pi_1}$. Let $V = f_{\pi_1}(V_1)$ be the data state obtained at the end of path $\pi_1$, when execution reaches $\tilde{\ell}$. We consider two cases:

- In the first case, there is no path $\pi_2$ connecting $\tilde{\ell}$ to $C$ such that $V \models c_{\pi_2}$. In this case $\tilde{\delta}_\ell(V) = \bot$ and there is obviously a descent between $\tilde{\delta}_{\ell_1}(V_1) = (\delta_{\ell_1}(V_1), 0)$ and $\tilde{\delta}_\ell(V) = \tilde{\delta}_\ell(f_{\pi_1}(V_1)) = \bot$.

- In the other case, there exist one or more paths $\pi_2$ connecting $\tilde{\ell}$ to $C$ such that $V \models c_{\pi_2}$. Let us pick the path $\pi_2$ such that $\tilde{\delta}_\ell(V) = (\delta_{\ell_2}(f_{\pi_2}(V)), 1)$. Assume that $\pi_2$ connects $\tilde{\ell}$ to $\ell_2 \in C$. The combined path $\pi = \pi_1 \circ \pi_2$ is one of the "old" verification paths and leads from $\ell_1$ to $\ell_2$. Since we assumed that the termination network $\mathcal{N}$ satisfied all the descent conditions, it satisfies in particular

$$\varphi_{\ell_1}(V_1) \land c_{\pi}(V_1) \rightarrow \delta_{\ell_1}(V_1) \succ \delta_{\ell_2}(f_{\pi}(V_1))$$

Recall that the traversal condition and data transformation of a composed path are related to these of its constituents by

$$c_{\pi}(V_1) = c_{\pi_1}(V_1) \land c_{\pi_2}(f_{\pi_1}(V_1)) \quad \text{and} \quad f_{\pi}(V_1) = f_{\pi_2}(f_{\pi_1}(V_1))$$

Using these relations and the facts that $\psi_{\ell_1} = \varphi_{\ell_1}$, $V_1 \models \psi_{\ell_1} \land c_{\pi_1}$, $V = f_{\pi_1}(V_1)$, $V \models c_{\pi_2}$, we can conclude that $\delta_{\ell_1}(V_1) \succ \delta_{\ell_2}(f_{\pi}(V_1)) = \delta_{\ell_2}(f_{\pi_2}(V))$. It follows that

$$\tilde{\delta}_{\ell_1}(V_1) = (\delta_{\ell_1}(V_1), 0) \succ (\delta_{\ell_2}(f_{\pi_2}(V)), 1) = \tilde{\delta}_\ell(V) = \tilde{\delta}_\ell(f_{\pi_1}(V))$$

establishing the descent from $\ell_1$ to $\tilde{\ell}$. \[\square\]
Removing Cut-Points

In the previous discussion we have shown that it is always possible to add more cut-points to a valid termination network, while maintaining validity. We will now show that it is also possible to remove cut-points, provided the remaining set is still a cut-set.

**Claim 14.** Let \( \mathcal{N} = \langle \mathcal{C}, \{ \varphi_\ell, \delta_\ell \mid \ell \in \mathcal{C} \} \rangle \) be a valid termination network. Let \( \tilde{\ell} \in \mathcal{C} \) be a location in \( \mathcal{C} \) such that \( \overline{\mathcal{C}} = \mathcal{C} - \{\tilde{\ell}\} \) is a cut-set. Then the network \( \overline{\mathcal{N}} = \langle \overline{\mathcal{C}}, \{ \varphi_\ell, \delta_\ell \mid \ell \in \overline{\mathcal{C}} \} \rangle \), obtained by removing \( \tilde{\ell} \) and \( \varphi_{\tilde{\ell}} \) from \( \mathcal{N} \), is also valid.

**Proof:** Inductiveness of the reduced assertion network follows from the arguments of Claim 5. It remains to show that the reduced ranking functions maintain the descent conditions over all verification paths.

We only need to consider “new” verification paths, i.e. paths which exist in \( \overline{\mathcal{N}} \) but not in \( \mathcal{N} \). Such a path \( \pi \) connecting \( \ell_1 \in \overline{\mathcal{C}} \) to \( \ell_2 \in \overline{\mathcal{C}} \) must be the fusion \( \pi = \pi_1 \circ \pi_2 \) of two paths, where path \( \pi_1 \) connects \( \ell_1 \) to \( \tilde{\ell} \), while \( \pi_2 \) connects \( \tilde{\ell} \) to \( \ell_2 \). Let \( V_1 \) be a data state which satisfies \( \varphi_1(V_1) \) and \( c_\pi(V_1) = c_{\pi_1}(V_1) \land c_{\pi_2}(f_{\pi_1}(V_1)) \). Let \( V = f_{\pi_1}(V_1) \) be the data state arising at \( \tilde{\ell} \) while traversing the path \( \pi \). Due to the \( \mathcal{N} \)-valid verification condition \( \varphi_{\ell_1}(V_1) \land c_{\pi_1}(V_1) \rightarrow \varphi_{\tilde{\ell}}(f_{\pi_1}(V_1)) \) and the \( \mathcal{N} \)-valid descent condition \( \varphi_{\ell_1}(V_1) \land c_{\pi_1}(V_1) \rightarrow \delta_{\ell_1}(V_1) \gg \delta_{\tilde{\ell}}(f_{\pi_1}(V_1)) \), we can conclude that \( \varphi_{\tilde{\ell}}(V) = 1 \) and \( \delta_{\ell_1}(V_1) \gg \delta_{\tilde{\ell}}(V) \). As \( c_{\pi_2}(V) = c_{\pi_2}(f_{\pi_1}(V_1)) \) is implied by \( c_{\pi}(V_1) \), we can use the \( \mathcal{N} \)-valid descent condition \( \varphi_{\tilde{\ell}}(V) \land c_{\pi_2}(V) \rightarrow \delta_{\tilde{\ell}}(V) \gg \delta_{\ell_2}(f_{\pi_2}(V)) \), to conclude \( \delta_{\tilde{\ell}}(V) \gg \delta_{\ell_2}(f_{\pi_2}(V)) = \delta_{\ell_2}(f_{\pi_1}(V_1)) \). Together with \( \delta_{\ell_1}(V_1) \gg \delta_{\tilde{\ell}}(V) \), this implies

\[
\delta_{\ell_1}(V_1) \gg \delta_{\ell_2}(f_{\pi}(V_1))
\]

as required.

---

*Sequential Program Analysis, NYU, Spring, 2003*
Method is Independent of the Choice of the Cut-Set

The preceding discussions can be summarized by the statement that the success or failure of an application of the ranking functions method is independent of the particular choice of the cut-set $C$.

Technically, this can be summarized by the following corollary:

**Corollary 15.** Let $p$ be a pre-condition specification for program $P$, and let $C_1$ and $C_2$ be two cut-sets. Then, there exists a valid termination network $N_1$ based on $C_1$ and entailing $p$ iff there exists a valid termination network $N_2$ based on $C_2$ and entailing $p$.

The proof of this statement can be obtained by starting with a $C_1$-based valid termination network $N_1$ and incrementally adding missing locations, using Claim 13, until we obtain a full network. Then we start removing locations which do not belong to $C_2$, relying on Claim 14, until we obtain a network $N_2$ based on $C_2$. Since $\ell_0$ belongs to both $N_1$ and $N_2$, its associated assertion is preserved throughout the entire process. Therefore, if $N_1$ entails $p$ then so does $N_2$. 
**König’s Lemma**

In preparation for proving the completeness of the ranking function method, we present a useful lemma. A tree is said to be of **finite degree** if each node has only finitely many direct descendants. A tree is called **infinite** if it contains infinitely many nodes.

**Lemma 16. [König]** An infinite tree of finite degree contains an infinite path.

**Proof:** Let $T$ be an infinite tree of finite degree. Let $n_0$ be the root of the tree and $n_1, \ldots, n_k$ its immediate descendants.

Each of the descendants $n_1, \ldots, n_k$ is the root of a subtree. Since $T$ is infinite, at least one of the subtrees $T_1, \ldots, T_k$ must be infinite. Assume it is $T_i$. Consider $n_{i1}, \ldots, n_{im}$, the immediate descendants of $n_i$. They, in turn, are the roots of a set of subtrees, at least one of which must be infinite. Continuing in this manner, we trace an infinite path within the tree $T$ where, at each stage, we consider a root of an infinite subtree. It follows that $T$ contains an infinite path.
Completeness of the Method

Finally, we state and prove the completeness of the ranking functions method for verifying $p$-termination of programs.

**Claim 17. [Completeness of the ranking functions method]** Let $P$ be a program which is $p$-terminating. Then there exists a valid termination network which entails $p$.

**Proof:** Let $C$ be an arbitrary cut-set. As the assertion associated with location $\ell$ we take the minimal predicate $M_\ell$, as defined in the proof of Claim 7. As shown in that proof, this assertion network is inductive and entails $p$.

As the well-founded domain, we take $(\mathbb{N}, >)$. As the ranking function $\delta_\ell$, we take the function $L_\ell(V)$ whose value for $V = d$ is defined by

If $d \not\models \varphi_\ell$ then $L_\ell(d) = 0$. Otherwise, $L_\ell(d)$ equals the length of the longest computation segment, initiated at location $\ell$ with data state $V = d$.

We have to show that $L_\ell(d)$ is always defined. The only possibility for $L_\ell(d)$ to be undefined is that $d \models \varphi_\ell$ but the length of $d$-computation segments departing from $\ell$ is unbounded. In this case, we construct a tree as follows:
The root of the tree is the execution state \( \langle \ell, d \rangle \). As immediate descendants of \( \langle \ell, d \rangle \) we list the execution states \( \langle \ell_1, d_1 \rangle, \ldots, \langle \ell_k, d_k \rangle \) which can be obtained by a single computation step from \( \langle \ell, d \rangle \). As direct descendants of these second generation nodes, we list all executions states which can be obtained by a single computation step from these states.

This tree is obviously of finite degree, since there are only finitely many edges departing from each location in the program graph. According to König’s lemma, either the tree is finite, or it has an infinite path.

An infinite path in this tree implies the existence of an infinite computation segment \( \pi^\infty \) whose first state is \( \langle \ell, d \rangle \). Since \( d \models \varphi_\ell = M_\ell \) there exists a \( p \)-computation segment of the form \( \pi = \langle \ell_0, d_0 \rangle, \ldots, \langle \ell, d \rangle \). Obviously, the concatenation \( \pi \circ \pi^\infty \) is a divergent \( p \)-computation of program \( P \) which, according to the hypothesis of the claim, is impossible.

We thus conclude that the tree rooted at \( \langle \ell_0, d \rangle \) is finite, which implies that there exists a longest computation segment departing from \( \langle \ell_0, d \rangle \). This shows that the function \( L_\ell(V) \) is always defined.
Proof Concluded

It remains to show that the ranking functions $L_\ell(V)$ satisfy the descent conditions. Consider a verification path $\pi$ leading from $\ell_1$ to $\ell_2$. Assume that $\varphi_{\ell_1}(d_1) = c_\pi(d_1) = 1$. We have to show that $\delta_{\ell_1}(d_1) > \delta_{\ell_2}(f_\pi(d_1))$.

Let $d_2 = f_\pi(d_1)$. Since $\varphi_{\ell_1}(d_1) = c_\pi(d_1) = 1$ and the assertion network is inductive, we can conclude that $\varphi_{\ell_2}(d_2) = 1$. Therefore, $L_{\ell_2}(d_2)$ is the length of some computation segment $\sigma = \langle \ell_2, d_2 \rangle, \ldots$. Similarly, $L_{\ell_1}(d_1)$ is the length of the longest computation segment departing from $\langle \ell_1, d_1 \rangle$. Since $\pi \circ \sigma$ is one of the computation segments departing from $\langle \ell_1, d_1 \rangle$, we have that $L_{\ell_1}(d_1) \geq |\pi| + |\sigma| = |\pi| + L_{\ell_2}(d_2)$. Since $|\pi| > 0$, we conclude that $L_{\ell_1}(d_1) > L_{\ell_2}(d_2)$. \qed
**Additional Examples: An Efficient GCD Program**

We conclude our discussion of methods for proving termination by presenting two more examples. The first is an efficient program for computing the \( gcd \) of two positive integers, using subtractions and left and right shifts (multiplication and division by 2).

As our assertion network, we take

\[
\begin{align*}
\varphi_0 &: \quad x_1 > 0 \land x_2 > 0 \\
\varphi_1 &: \quad y_1 > 0 \land y_2 > 0 \\
\varphi_2 &: \quad y_1 > 0 \land y_2 > 0 \land odd(y_1)
\end{align*}
\]

In order to prove partial correctness we could just add the conjunct \( gcd(x_1, x_2) = gcd(y_1, y_2) \cdot y_3 \) to \( \varphi_1 \) and \( \varphi_2 \). As ranking function for \( \ell_1 \), we can take

\[
\delta_1 : \quad y_1
\]
Efficient GCD Example Continued

To choose a ranking function at \( \ell_2 \) we try a linear combination of the form \( ay_1 + by_2 \). This linear combination must be descending on all paths from \( \ell_2 \) to itself. This leads to the following inequalities:

\[
ay_1 + by_2 > ay_1 + b\frac{y_2}{2} \\
ay_1 + by_2 > ay_2 + b\left|\frac{y_1 - y_2}{2}\right| \\
2ay_1 + 2by_2 > 2ay_1 + by_2 \\
2ay_1 + 2by_2 > 2ay_2 + b|y_1 - y_2|
\]

The first inequality can be simplified to \( b > 0 \). The second inequality can be expanded into two cases, depending on whether \( y_1 \geq y_2 \) or \( y_1 < y_2 \). This gives rise to the following two inequalities:

\[
2ay_1 + 2by_2 > 2ay_2 + b(y_1 - y_2) \\
2ay_1 + 2by_2 > 2ay_2 + b(y_2 - y_1)
\]

Collecting the coefficients of \( y_1 \) and \( y_2 \), we get

\[
y_1(2a - b) + y_2(3b - 2a) > 0 \\
y_1(2a + b) + y_2(b - 2a) > 0
\]

Since these inequalities should hold for arbitrary positive values of \( y_1 \) and \( y_2 \) we must have all coefficients nonnegative, and their sum positive. Thus, we have to find \( a \) and \( b \) satisfying

\[
b > 0 \\
2a - b \geq 0 \\
3b - 2a \geq 0 \\
2a + b \geq 0 \\
b - 2a \geq 0
\]

A possible solution is \( a = 1, b = 2 \), which leads to the ranking function

\[
\delta_2 : \quad y_1 + 2 \cdot y_2
\]

guaranteed to descend on all self edges departing from \( \ell_2 \).
Structured GCD

Consider the following program which also computes the \textit{gcd} of two positive integers:

\[
\begin{align*}
\ell_0 & : (y_1, y_2) := (x_1, x_2) \\
\ell_1 & : y_1 = y_2? \\
\ell_2 & : y_1 > y_2 \rightarrow [y_1 := y_1 - y_2] \\
\ell_3 & : y_1 \leq y_2? \\
\ell_4 & : y_1 \neq y_2?
\end{align*}
\]

As the assertion network we can take

\[
\begin{align*}
\varphi_0 : & \quad x_1 > 0 \land x_2 > 0 \\
\varphi_2 = \varphi_3 : & \quad y_1 > 0 \land y_2 > 0
\end{align*}
\]

It is obvious that the function \(\delta : y_1 + y_2\) should be a component of the ranking at both \(\ell_2\) and \(\ell_3\). However, \(\delta\) by itself is insufficient, because it does not decrease on the verification path \(\ell_2 \rightarrow \ell_3\). Therefore, we will be looking for ranking functions of the form \(\delta_i : (y_1 + y_2, \eta_i)\) for \(i = 2, 3\).
Structured GCD Continued

This characteristic example has the feature that it is easy to identify a primary ranking function (i.e., $y_1 + y_2$) which is known to decrease on every loop. The question is how to augment it by the secondary rank $\eta_i(V)$ such that there will be a decrease on every intra-component verification path.

A heuristics that often works is to let $\eta_i$ be the maximal number of cut-points which are encountered on a path departing from $l_i$ until we reach an edge which causes the primary rank to decrease or until we reach the terminal location $l_t$. For example, $\eta_2$ can be defined as follows, distinguishing between three cases:

$$\begin{align*}
  y_1 > y_2 & \quad 0 \quad \text{The first edge departing from } l_2 \text{ decrements } y_1 + y_2 \\
  y_1 < y_2 & \quad 1 \quad \text{The path departing from } l_2 \text{ is } l_2 \rightarrow l_1 \rightarrow \text{decrement}(y_1 + y_2) \\
  y_1 = y_2 & \quad 2 \quad \text{The path departing from } l_2 \text{ is } l_2 \rightarrow l_1 \rightarrow l_4
\end{align*}$$

Consequently, we can take

$$\delta_2(y_1, y_2) = (y_1 + y_2, \ \text{if } y_1 > y_2 \ \text{then } 0 \ \text{else-if } y_1 < y_2 \ \text{then } 1 \ \text{else } 2)$$

Similarly, we can take

$$\delta_3(y_1, y_2) = (y_1 + y_2, \ \text{if } y_1 > y_2 \ \text{then } 1 \ \text{else-if } y_1 < y_2 \ \text{then } 0 \ \text{else } 1)$$

A more compact representation of similar ranking functions can be given by

$$\begin{align*}
  \delta_2 & : (y_1 + y_2, \ 2(y_1 \leq y_2)) \\
  \delta_3 & : (y_1 + y_2, \ y_1 \geq y_2)
\end{align*}$$