Programs with Procedures

We will now extend our treatment of programs to the consideration of programs with procedures. A program $P$ in the extended language consists of $m+1$ modules: $P_0, P_1, \ldots, P_m$, where $P_0$ is the main module, and $P_1, \ldots, p_m$ are procedures which may be called from $P_0$ or from other procedures.

Each module $P_i$ is presented as a flow-graph with its own set of locations $\mathcal{L}_i = \{\ell^i_0, \ell^i_1, \ldots, \ell^i_t\}$. It must have $\ell^i_0$ as its only entry point, $\ell^i_t$ as its only exit, and every other location must be on a path from $\ell^i_0$ to $\ell^i_t$.

The variables of each module $P_i$ are partitioned into $\bar{y} = (\bar{x}; \bar{u}; \bar{z})$. We refer to $\bar{x}, \bar{y}$, and $\bar{z}$ as the input, working, and output variables, respectively. A module cannot modify its own input variables.

Edges in the graph are labeled by an instruction which must be one of

- An assignment $c(\bar{y}) \rightarrow [\bar{v} := f(\bar{y})]$, where the left-hand side variables $\bar{v} \subseteq \{\bar{u}, \bar{z}\}$ may not include any member of $\bar{x}$.

- A procedure call $c(\bar{y}) \rightarrow P_j(\bar{e}; \bar{v})$, where $\bar{e}$ is a list of expressions over $\bar{y}$, and $\bar{v} \subseteq \{\bar{u}, \bar{z}\}$ is a list of distinct variables not including any member of $\bar{x}$. We refer to $\bar{e}$ and $\bar{y}$ as the actual arguments of the call.
Computations of Procedural Programs

A $\xi$-computation of module $P_i$ is a sequence of states and their labeled transitions:

$\sigma : \langle \ell^i_0; (\xi, \top, \top) \rangle \xrightarrow{\lambda_1} \langle \ell^1_1; d^1_1 \rangle \xrightarrow{\lambda_2} \langle \ell^2_2; d^2_2 \rangle \cdots$

The values $\top$ denote uninitialized values. Labels in the transitions are either names of edges in the program or the special label $\text{return}$. Each transition $\langle \ell; d \rangle \xrightarrow{\lambda} \langle \ell'; d' \rangle$ in a computation must be justified by one of the following cases:

**Assignment** There exists an edge $e$ in the program $P$ (not necessarily in $P_i$)

\[
\begin{array}{c}
\ell \\
\xrightarrow{e} c(y) \rightarrow [v := f(y)]
\end{array}
\]

such that $\lambda = e$, $d \models c$, and $d' = (d \text{ with } v = f(d))$, i.e. $d'$ is obtained from $d$ by replacing the values corresponding to the variables $v$ by $f(d)$.

**Procedure Call** There exists an edge $e$ in the program $P$

\[
\begin{array}{c}
\ell \\
\xrightarrow{e} c(y) \rightarrow P_j(e; v)
\end{array}
\]

such that $\lambda = e$, $d \models c$, $\ell' = \ell^j_0$, and $d' = (c'(d); \top; \top)$. Thus, $\ell' = \ell^j_0$ is the first location in the called procedure $P_j$, and $d'$ are the initial values on entry to $P_j$. We assume that the working and result variables in $P_j$ are uninitialized.
Computations Continued: Procedure Return

Finally we consider a transition $\langle \ell; (\xi; \eta; \zeta) \rangle \xrightarrow{\text{return}} \langle \ell'; \vec{d}' \rangle$. To justify such a transition, there must exists a procedure $P_j$ (the procedure from which we return), such that $\ell = \ell_j^t$ (the terminal location of $P_j$), and we should be able to identify a suffix of the current computation of the form

$$\langle \ell_1; \vec{d}_1 \rangle \xrightarrow{e_1} \langle \ell_0^j; (\xi; \overrightarrow{1}; \overrightarrow{1}) \rangle \xrightarrow{e_2} \cdots \xrightarrow{e_k} \langle \ell; (\xi; \eta; \zeta) \rangle$$

such that the segment $\sigma_1$ is balanced (has an equal number of call's and return's), $e_1$ is a call edge of the form

$$\ell_1 \xrightarrow{c(y)} P_j(\vec{e}; \vec{v}) \xrightarrow{e_1} \ell_2$$

$\ell' = \ell_2$, and $\vec{d}' = (\vec{d}_1 \text{ with } \vec{v} = \zeta)$. 
Results of Computations

Given a computation $\sigma$, it is possible to assign to each execution state in $\sigma$ a depth which is a natural number equal to $\#(\text{call}) - \#(\text{return})$ from the beginning of the computation up to the current state. A computation is called maximal if it cannot be extended any further.

Maximal computations fall into one of the possible categories:

- **Terminating computations** – The computation $\sigma$ is finite, and its last state is $\langle l^i; (\xi; \eta; \zeta) \rangle$. We define
  $$\text{val}(\sigma) = \zeta$$

- **Failing Computations** – The computation $\sigma$ is finite, but its last location is not a terminal location of any procedure. This is a case of a deadlock and we write
  $$\text{val}(\sigma) = \text{fail}$$

- **Divergent computations** – $\sigma$ is infinite. Define
  $$\text{val}(\sigma) = \bot$$

For a module $P_i$, we define the meaning of $P_i$ to be the set of all possible outcomes.

$$\mathcal{M}[P_i](\xi) = \mathcal{M}(P_i; \xi) = \{ \text{val}(\sigma) \mid \sigma \text{ is a maximal } \xi\text{-computation of } P_i \}$$

For the entire program $P$, we define

$$\mathcal{M}[P] = \mathcal{M}[P_0]$$
Example: Factorial

Consider the following program for computing the factorial of a natural number.

\[
P_0(x, z) : \ell_0^0 \xrightarrow{e_1} P_1(x; z) \xrightarrow{e_1} \ell_0^0 \quad \text{and} \quad P_1(x, z) : \ell_0^1 \xrightarrow{e_2} x = 0 \rightarrow [z := 1] \xrightarrow{e_2} \ell_1^1 \xrightarrow{e_3} x > 0 \rightarrow P_1(x - 1; z) \xrightarrow{e_4} \ell_1^1 \xrightarrow{e_3} z := x \cdot z
\]

Following is a computation of this program for input \(x = 3\):

\[
\langle \ell_0^0; (3, \bot) \rangle \xrightarrow{e_1} \langle \ell_0^1; (3, \bot) \rangle \xrightarrow{e_3} \langle \ell_0^1; (2, \bot) \rangle \xrightarrow{e_3} \langle \ell_0^1; (1, \bot) \rangle \xrightarrow{e_3} \langle \ell_0^1; (0, \bot) \rangle \xrightarrow{e_2} \langle \ell_1^1; (0, 1) \rangle \xrightarrow{\text{return}}
\]

\[
\langle \ell_1^1; (1, 1) \rangle \xrightarrow{e_4} \langle \ell_1^1; (1, 1) \rangle \xrightarrow{\text{return}}
\]

\[
\langle \ell_1^1; (2, 1) \rangle \xrightarrow{e_4} \langle \ell_1^1; (2, 2) \rangle \xrightarrow{\text{return}}
\]

\[
\langle \ell_1^1; (3, 2) \rangle \xrightarrow{e_4} \langle \ell_1^1; (3, 6) \rangle \xrightarrow{\text{return}}
\]

\[
\langle \ell_1^1; (3, 2) \rangle \xrightarrow{e_4} \langle \ell_1^1; (3, 6) \rangle \xrightarrow{\text{return}}
\]

Consequently, \(M(\text{factorial}, 3) = 6\).
Proving Partial Correctness

We extend the inductive assertion method to deal with procedural programs. A cut-set $C$ is a set of locations in $L = L_0 \cup \cdots \cup L_m$ such that:

1. Every loop in each $P_i$, $i = 0, \ldots, m$ contains at least one location of $C$.

2. For every $i = 0, \ldots, m$, both $l_0^i$ and $l_t^i$ belong to $C$.

2. For every edge $l_i \xrightarrow{e} l_j$ labeled by a procedure call, both $l_i$ and $l_j$ belong to $C$.

An assertion network associates an assertion $\varphi_i^j(y)$ with each location $l_i^j$. For each module $P_k$, we denote $\varphi_k^i$ by $p_k$ and require that $p_k = p_k(x)$ depends only on the input variables of the module. Similarly, we denote $\varphi_k^j$ by $q_k$ and require that $q_k = q_k(x; z)$ depends only on the input and output variables of the module.

The input predicate $p_k(x)$ imposes constraints on the input variables we expect on entry to module $P_k$. The output predicate $q_k(x; z)$ specifies the relation between the output results and the input values.
The Verification Conditions

We consider two types of verification conditions.

Let $\pi$ be a verification path leading from location $\ell_i$ to location $\ell_j$ such that all edges in $\pi$ are labeled by guarded assignment instructions. We refer to such a path as an assignment path. As usual, let $c_\pi$ denote the traversal condition for $\pi$, and let $\vec{y} := f_\pi(\vec{y})$ summarize the data transformation effected by the execution of the path. With such a path we associate the following verification condition:

$$V_\pi : \quad \varphi_i(\vec{y}) \land c_\pi(\vec{y}) \rightarrow \varphi_j(f_\pi(\vec{y}))$$

The other type of verification condition is associated with a procedure call. Consider an edge of the following form:

$$\ell_i \xrightarrow{c(\vec{y}) \rightarrow P_k(\vec{E}; \vec{v})} \ell_j$$

With the (length one) verification path $e$, we associate the following two verification conditions:

$$V_{\text{entry}} : \quad \varphi_i(\vec{y}) \land c(\vec{y}) \rightarrow p_k(\vec{E}(\vec{y}))$$
$$V_{\text{exit}} : \quad \varphi_i(\vec{y}) \land c(\vec{y}) \land q_k(\vec{E}(\vec{y}); \vec{z}') \rightarrow \varphi_j(\vec{y})[\vec{v} \mapsto \vec{z}']$$

where $\varphi_j(\vec{y})[\vec{v} \mapsto \vec{z}']$ is obtained from $\varphi_j(\vec{y})$ by replacing variables in $\vec{v}$ by corresponding variables in $\vec{z}'$. 

Soundness of the Method

An assertion network which satisfies all the verification conditions is called an inductive network. An assertion network is defined to be \( p \)-invariant if every \( p \)-computation \( \sigma \) which reaches location \( \ell \in C \) with data state \( \vec{y} = \vec{d} \) satisfies \( \vec{d} \models \varphi_\ell \).

Claim 18. An inductive assertion network whose assertion at \( \ell_0^0 \) is \( p_0 \) is a \( p_0 \)-invariant network.

The claim can be proved by induction on the number of cut-points which the computation \( \sigma \) visits.

Corollary 19. If the network \( \mathcal{N} \) is inductive for program \( P \), then \( P \) is partially correct w.r.t the specification \( \langle p_0, q_0 \rangle \). Furthermore, if \( \mathcal{N} \) entails the specification \( \langle p, q \rangle \), then \( P \) is partially correct w.r.t \( \langle p, q \rangle \).
**Example: Factorial**

Reconsider the program for computing the factorial of a natural number.

\[ P_0(x, z) : \ell_0^0 \xrightarrow{P_0(x; z)} \ell_t^0 \quad P_1(x, z) : \ell_0^1 \xrightarrow{e_1} \ell_t^1 \]

\[ x > 0 \rightarrow P_1(x - 1; z) \quad \xrightarrow{e_4} z := x \cdot z \]

We will prove that this program is partially correct w.r.t the specification

\[ p : x \geq 0 \quad q : z = x! \]

As the cut-set we take all locations. The proposed assertion network is given by

\[ p_0 = p_1 : x \geq 0 \quad q_0 = q_1 : z = x! \]

\[ \varphi_1 : x > 0 \land z = (x - 1)! \]

This gives rise to the following set of valid verification conditions:

\[ V_{entry}^{e_1} : x \geq 0 \quad \rightarrow x \geq 0 \]
\[ V_{exit}^{e_1} : x \geq 0 \land z' = x! \quad \rightarrow z' = x! \]
\[ q_1(x, z') \quad q_0[z \mapsto z'] \]
\[ V_{e_2} : x \geq 0 \land x = 0 \quad \rightarrow 1 = x! \]
\[ q_0(f_{e_1}(x; z)) \]
\[ V_{e_3} : x \geq 0 \land x > 0 \quad \rightarrow x - 1 \geq 0 \]
\[ p_1(x - 1) \]
\[ V_{exit}^{e_3} : x \geq 0 \land x > 0 \land x > 0 \land z' = (x - 1)! \quad \rightarrow x > 0 \land z' = (x - 1)! \]
\[ q_1(x - 1, z') \]
\[ \varphi_1(x, z') \]
\[ V_{e_4} : x > 0 \land z = (x - 1)! \quad \rightarrow x \cdot z = x! \]
\[ q_1(x, x \cdot z) \]
**Example: Fibonacci**

As another example, consider the following program for the computation of the \(x\)'th element, \(x \geq 0\), of the Fibonacci series:

\[
2, 1, 3, 4, 7, 11, \ldots
\]

where \(a_0 = 2\), \(a_1 = 1\), and \(a_{i+2} = a_{i+1} + a_i\) for \(i \geq 0\).

As the specification, we take

\[
p : x \geq 0 \quad q : z = \alpha^x + \beta^x
\]

where \(\alpha\) and \(\beta\) are the two roots of quadratic equation \(u^2 - u - 1 = 0\). As the cut-set we take all locations. The proposed assertion network is given by

\[
p_0 = p_1 : \quad x \geq 0
\]
\[
q_0 = q_1 : \quad z = \alpha^x + \beta^x
\]
\[
\varphi_1^1 : \quad x \geq 2 \land u_1 = \alpha^{x-1} + \beta^{x-1}
\]
\[
\varphi_2^1 : \quad x \geq 2 \land u_1 = \alpha^{x-1} + \beta^{x-1} \land u_2 = \alpha^{x-2} + \beta^{x-2}
\]
Verification Conditions for Fibonacci

The preceding choices give rise to the following set of valid verification conditions:

\[
\begin{align*}
V_{\text{entry}}^{e_1} &: \quad x \geq 0 & \rightarrow & \quad x \geq 0 \\
V_{\text{exit}}^{e_1} &: \quad x \geq 0 \land z' = \alpha^x + \beta^x & \rightarrow & \quad z' = \alpha^x + \beta^x \\
V_{\text{entry}}^{e_2} &: \quad x \geq 0 \land x \leq 1 & \rightarrow & \quad 2 - x = \alpha^x + \beta^x \\
V_{\text{exit}}^{e_2} &: \quad x \geq 0 \land x \geq 2 & \rightarrow & \quad x - 1 \geq 0 \\
V_{\text{entry}}^{e_3} &: \quad x \geq 0 \land x \geq 2 \land u_1 = \alpha^{x-1} + \beta^{x-1} & \rightarrow & \quad x \geq 2 \land u_1 = \alpha^{x-1} + \beta^{x-1} \\
V_{\text{exit}}^{e_3} &: \quad x \geq 0 \land x \geq 2 \land u_1 = \alpha^{x-1} + \beta^{x-1} & \rightarrow & \quad x - 2 \geq 0 \\
V_{\text{entry}}^{e_4} &: \quad x \geq 2 \land u_1 = \alpha^{x-1} + \beta^{x-1} & \rightarrow & \quad x - 2 \geq 0 \\
V_{\text{exit}}^{e_4} &: \quad x \geq 2 \land u_1 = \alpha^{x-1} + \beta^{x-1} \land u_2 = \alpha^{x-2} + \beta^{x-2} & \rightarrow & \quad u_1 + u_2 = \alpha^x + \beta^x \\
V_{\text{entry}}^{e_5} &: \quad x \geq 2 \land u_1 = \alpha^{x-1} + \beta^{x-1} \land u_2 = \alpha^{x-2} + \beta^{x-2} & \rightarrow & \quad \alpha^x + \beta^x \\
\end{align*}
\]
Proving Success (Deadlock Absence) of Procedural Programs

As in the case of non-procedural programs, we define for each location \( \ell \in \mathcal{L} \), its exit condition

\[
E_\ell : \quad c_1 \lor \cdots \lor c_k
\]

where \( c_1, \ldots, c_k \) are the guards on all edges departing from node \( \ell \). For a cut-set \( \mathcal{C} \) and location \( \ell \not\in \mathcal{C} \) we denote by \( \Pi_{\mathcal{C},\ell} \) the set of paths connecting a location in \( \mathcal{C} \) to \( \ell \) without passing through any other cut-point. For each path \( \pi \in \Pi_{\mathcal{C},\ell} \), let \( src(\pi) \), \( c_\pi \), and \( f_\pi \) denote, respectively, the cut-point at the beginning of path \( \pi \), the summary traversal condition, and data transformation associated with \( \pi \).

The following claim summarizes the general rule for proving success.

**Claim 20.** In order to prove that program \( P \) is \( p \)-successful (i.e., no \( p \)-computation ever deadlocks), it is sufficient to find a network \( \mathcal{N} : \langle \mathcal{C}, \{ \varphi_\ell \mid \ell \in \mathcal{C} \} \rangle \), satisfying the following requirements:

1. The network \( \mathcal{N} \) is inductive.
2. \( p \rightarrow p_0 \)
3. \( \varphi_\ell \rightarrow E_\ell \) for every \( \ell \in \mathcal{C} \)
4. \( \varphi_{src(\pi)}(V) \land c_\pi(V) \rightarrow E_{\ell}(f_\pi(V)) \) for every \( \ell \not\in \mathcal{C} \) and path \( \pi \in \Pi_{\mathcal{C},\ell} \)
Proving Convergence of Procedural Programs

The method starts by constructing an inductive assertion network $\mathcal{N}$. We then choose a well-founded domain $(\mathcal{A}, \succ)$. With each cut-point $\ell \in C$ we associate a ranking function $\delta_\ell$. Ranking functions associated with procedure entry points $\ell_0^k$ depend only on $\vec{x}$, while ranking functions associated with terminal locations $\ell_t^k$ will depend only on $(\vec{x}, \vec{z})$.

We then form descent conditions as follows:

- For an assignment path $\pi$ connecting location $\ell_i$ to $\ell_j$, we form the following descent condition:

$$D_\pi : \quad \varphi_i(\vec{y}) \land c_\pi(\vec{y}) \implies \delta_i(\vec{y}) \succ \delta_j(f_\pi(\vec{y}))$$

- For a call edge of the form:

$$\ell_i \xrightarrow{c(\vec{y})} P_k(\vec{E}; \vec{v}) \xrightarrow{e} \ell_j$$

we form the following two descent conditions:

$$D_{e}^{in} : \quad \varphi_i(\vec{y}) \land c(\vec{y}) \implies \delta(\vec{y}) \succ \delta_0^k(\vec{E}(\vec{y}))$$

$$D_{e}^{out} : \quad \varphi_i(\vec{y}) \land c(\vec{y}) \land q_k(\vec{E}(\vec{y}); \vec{z'}) \implies \delta_i(\vec{y}) \succ \delta_j(\vec{y})[\vec{v} \mapsto \vec{z'}]$$
The Proof Method

The proof method is summarized in the following claim:

Claim 21. [Verifying Termination of Procedural Programs]
Let $P$ be a procedural program with a pre-condition specification $p$. Let $\mathcal{N} : \langle C, \{ \varphi_\ell \mid \ell \in C \} \rangle$ be an assertion network, $(A, \succ)$ be a well-founded domain, and $\{ \delta_\ell \mid \ell \in C \}$ be a network of ranking functions, each mapping states into elements of $A$. If the following requirements are satisfied:

1. The network $\mathcal{N}$ is inductive.
2. $p \rightarrow p_0$
3. The 3 types of verification conditions are valid for all generated verification paths.

then program $P$ is $p$-convergent.
**Example: Factorial**

Reconsider the procedural program for computing the factorial function:

\[
\begin{align*}
P_0(x, z) & : \ell_0^0 \quad P_1(x; z) & : \ell_0^0 \\
& \quad e_1 \quad & \quad e_1 \\
& \quad \ell_t^0 \quad & \quad \ell_t^0
\end{align*}
\]

\[
P_1(x, z) : \ell_0^1 \quad x = 0 \rightarrow [z := 1] \\
& \quad e_2 \quad & \quad e_4 \\
& \quad \ell_t^1 \quad & \quad \ell_t^1
\]

\[
x > 0 \rightarrow P_1(x - 1; z)
\]

\[
z := x \cdot z
\]

As a first step, let us prove the termination of procedure \(P_1\) under the pre-condition \(x \geq 0\). As the well-founded domain we choose \(\mathbb{N} \times \text{lex} [0..2]\). As the termination network 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_0^1)</td>
<td>(x \geq 0)</td>
<td>((</td>
</tr>
<tr>
<td>(\ell_1^1)</td>
<td>(x &gt; 0)</td>
<td>((</td>
</tr>
<tr>
<td>(\ell_t^1)</td>
<td>(1)</td>
<td>((</td>
</tr>
</tbody>
</table>

which leads to the following descent conditions:

\[
\begin{align*}
D_{e_2} : & \quad x \geq 0 \land x = 0 \quad \rightarrow \quad (|x|, 2) \succ (|x|, 0) \\
D_{e_3}^{\text{in}} : & \quad x \geq 0 \land x > 0 \quad \rightarrow \quad (|x|, 2) \succ (|x - 1|, 2) \\
D_{e_3}^{\text{out}} : & \quad x \geq 0 \land x > 0 \quad \rightarrow \quad (|x|, 2) \succ (|x|, 1) \\
D_{e_4} : & \quad x > 0 \quad \rightarrow \quad (|x|, 1) \succ (|x|, 0)
\end{align*}
\]
Verifying Complete Programs

Note that if we wish to verify termination of procedure $P_0$, we need to use the following more cumbersome ranking network:

<table>
<thead>
<tr>
<th>$\ell_i$</th>
<th>$\varphi_{\ell_i}$</th>
<th>$\delta_{\ell_i}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\ell_0^0$</td>
<td>$x \geq 0$</td>
<td>$(</td>
</tr>
<tr>
<td>$\ell_t^0$</td>
<td>1</td>
<td>$(</td>
</tr>
<tr>
<td>$\ell_0^1$</td>
<td>$x \geq 0$</td>
<td>$(</td>
</tr>
<tr>
<td>$\ell_1^1$</td>
<td>$x &gt; 0$</td>
<td>$(</td>
</tr>
<tr>
<td>$\ell_t^1$</td>
<td>1</td>
<td>$(</td>
</tr>
</tbody>
</table>

which includes also ranking for the locations of $P_0$. 
Acyclic Decomposition of Call Graphs

The task of verifying entire program can be somewhat simplified by considering the call graph of a procedural program. This is a graph which contains a node for each procedure $P_i$, $i = 0, \ldots, k$, and a directed edge connecting node $P_i$ to $P_j$ if procedure $P_i$ contains a call to procedure $P_j$. For example, the call graph for program factorial is given by:

Assume that we have constructed the call graph for program $P$ and computed an acyclic decomposition $K_1, \ldots, K_k$ of the procedures, such that if procedure $P_i \in K_i$ calls procedure $P_j \in K_j$ then, necessarily $i \leq j$. We refer to $K_1, \ldots, K_k$ as procedure clusters, and to an edge on which $P_i$ calls $P_j$ where both procedures belong to the same cluster an intra-cluster edge. Using this terminology, we have the following:

**Claim 22. [Improved Version]** In the application of the ranking function method, it is sufficient to require that conditions $D_{e}^{in}$ and $D_{e}^{out}$ of Claim 21 hold for intra-cluster edges $e$. 
Example: Ackerman’s Function

As the next example, we consider Ackerman’s function which can be defined by the following recursive definition:

\[
\begin{align*}
A(0, x_2) &= x_2 + 1 \\
A(x_1 + 1, 0) &= A(x_1, 1) \\
A(x_1 + 1, x_2 + 1) &= A(x_1, A(x_1 + 1, x_2))
\end{align*}
\]

A procedural program for computing \( A(x_1, x_2) \) can be given as follows:

\[
\begin{align*}
P_0(x_1, x_2; z) : \\
\ell_0^0 &\rightarrow P_1(x_1, x_2; z) \\
x_1 = 0 &\rightarrow [z := x_2 + 1] \\
P_1(x_1, x_2; z) : \\
x_1 > 0 \land x_2 = 0 &\rightarrow P_1(x_1 - 1, 1; z) \\
x_1 > 0 \land x_2 > 0 &\rightarrow P_1(x_1, x_2 - 1; u) \\
P_1(x_1 - 1, u; z) &\rightarrow P_1(x_1 - 1, u; z)
\end{align*}
\]

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_0^1 )</td>
<td>( x_1 \geq 0 \land x_2 \geq 0 )</td>
<td>( x_1, x_2, 2 )</td>
</tr>
<tr>
<td>( \ell_1^0 )</td>
<td>( x_1 \geq 0 \land x_2 \geq 0 )</td>
<td>( x_1, x_2, 1 )</td>
</tr>
<tr>
<td>( \ell_1^1 )</td>
<td>( x_1 &gt; 0 \land x_2 &gt; 0 )</td>
<td>( x_1, x_2, 0 )</td>
</tr>
<tr>
<td>( \ell_t^1 )</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
which gives rise to the following descent conditions:

<table>
<thead>
<tr>
<th>Condition</th>
<th>Expression</th>
<th>Transition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D_{e_2}$</td>
<td>$x_1 \geq 0 \land x_2 \geq 0 \land x_1 = 0$</td>
<td>$\rightarrow \ (</td>
</tr>
<tr>
<td>$D_{e_3}^{in}$</td>
<td>$x_1 \geq 0 \land x_2 \geq 0 \land x_1 &gt; 0 \land x_2 = 0$</td>
<td>$\rightarrow \ (</td>
</tr>
<tr>
<td>$D_{e_3}^{out}$</td>
<td>$x_1 \geq 0 \land x_2 \geq 0 \land x_1 &gt; 0 \land x_2 = 0$</td>
<td>$\rightarrow \ (</td>
</tr>
<tr>
<td>$D_{e_4}^{in}$</td>
<td>$x_1 \geq 0 \land x_2 \geq 0 \land x_1 &gt; 0 \land x_2 &gt; 0$</td>
<td>$\rightarrow \ (</td>
</tr>
<tr>
<td>$D_{e_4}^{out}$</td>
<td>$x_1 \geq 0 \land x_2 \geq 0 \land x_1 &gt; 0 \land x_2 &gt; 0$</td>
<td>$\rightarrow \ (</td>
</tr>
<tr>
<td>$D_{e_5}^{in}$</td>
<td>$x_1 &gt; 0 \land x_2 &gt; 0$</td>
<td>$\rightarrow \ (</td>
</tr>
<tr>
<td>$D_{e_5}^{in}$</td>
<td>$x_1 &gt; 0 \land x_2 &gt; 0$</td>
<td>$\rightarrow \ (</td>
</tr>
</tbody>
</table>