Unity Program structure

program → Program program-name
  declare declare-section
  always always-section
  initially initially-section
  assign assign-section
  end

The *declare-section* declares variables and their types. The *always-section* gives names to expressions. Reference to these names within other expressions is interpreted as textual substitution. The *initially-section* specifies the initial values of variables.
Assignment Statement

\[
\text{assignment-statement} \quad \rightarrow \quad \text{assignment-component} \mid \{ \parallel \text{assignment-component}\}
\]

\[
\text{assignment-component} \quad \rightarrow \quad \text{enumerated-assignment} \mid \text{quantified-assignment}
\]

The \(\parallel\) separator separates between assignment components which are executed simultaneously.
Enumerated Assignment

\[
\begin{align*}
\text{enumerated-assignment} & \rightarrow \text{variable-list} := \text{expression-list} \\
\text{variable-list} & \rightarrow \text{variable}\{, \text{variable}\} \\
\text{expr-list} & \rightarrow \text{simple-expr-list} | \text{cond-expr-list} \\
\text{simple-expr-list} & \rightarrow \text{expr}\{, \text{expr}\} \\
\text{cond-expr-list} & \rightarrow \text{simple-expr-list} \text{ if } \text{boolean-expr} \\
& \quad \{\sim \text{simple-expr-list} \text{ if } \text{boolean-expr}\}
\end{align*}
\]

Examples:

- \(x, y := y, x\) — Exchange \(x\) and \(y\).
- \(x := y \text{ if } y \geq 0 \sim -y \text{ if } y \leq 0\) — same as \(x := |y|\).
A Computational Model

A fair transition system (FTS) $\mathcal{D} = \langle V, \Theta, T, J, C \rangle$ consists of:

- $V$ – A finite set of typed state variables. A $V$-state $s$ is an interpretation of $V$. We denote by $s[x]$ the value assigned to variable $x \in V$ by state $s$. $\Sigma_V$ – the set of all $V$-states.

- $\Theta$ – An initial condition. A satisfiable assertion that characterizes the initial states.

- $T$ – A set of transitions. Each transition $\tau \in T$ is associated with a transition relation, which is an assertion $\rho[\tau](V, V')$, referring to both unprimed (current) and primed (next) versions of the state variables. For example, $x' = x + 1$ corresponds to the assignment $x := x + 1$.

- $J \subseteq T$ – A subset of just (weakly fair) transitions. Each of these transitions must be taken infinitely many times if it is continuously enabled.

- $C \subseteq T$ – A subset of compassionate (strongly fair) transitions. Each of these transitions must be taken infinitely many times if it is enabled infinitely many times.
Some Definitions

State $s'$ is defined to be a $\tau$-successor of state $s$ if $\langle s, s' \rangle \models \rho[\tau]$, where $\langle s, s' \rangle$ is a joint interpretation which interprets each unprimed variable $x$ as $s[x]$ and each primed variable $x'$ as $s'[x]$.

A transition $\tau$ is enabled on state $s$, written $s \models En[\tau]$, if $s$ satisfies the assertion $\exists V'.\rho[\tau](V, V')$, implying that $s$ has some $\tau$-successor.

Let $\sigma : s_0, s_1, \ldots$ be an infinite set of states. We say that transition $\tau \in \mathcal{T}$ is enabled at position $j \geq 0$ if it is enabled on $s_j$, We say that $\tau$ is taken at position $j \geq 0$ if $s_{j+1}$ is a $\tau$-successor of $s_j$.

We define the total transition relation to be the disjunction

$$\rho : \bigvee_{\tau \in \mathcal{T}} \rho[\tau]$$
Computations

Let $D$ be an FTS for which the above components have been identified. We define a computation of $D$ to be an infinite sequence of states

$$\sigma : s_0, s_1, s_2, \ldots,$$

satisfying the following requirements:

- **Initiality**: $s_0$ is initial, i.e., $s_0 \models \Theta$.
- **Consecution**: For each $j = 0, 1, \ldots$, state $s_{j+1}$ is a $\rho$-successor of the state $s_j$.
- **Justice**: For each $\tau \in J$, $\sigma$ contains either infinitely many positions at which $\tau$ is disabled, or infinitely many positions at which $\tau$ is taken. Equivalently, if $\tau$ is continuously enabled from a certain position on, it must be taken infinitely many times in $\sigma$.
- **Compassion**: For each $\tau \in C$, if $\sigma$ contains infinitely many positions at which $\tau$ is enabled, it must also contain infinitely many positions at which $\tau$ is taken.
Proof Rules: Proving Invariance

The following rule can be used (and is complete) for verifying invariance properties.

Rule INV

For an assertion $\varphi$,

I1. $\Theta \rightarrow \varphi$
I2. $\varphi \land \rho \rightarrow \varphi'$
I3. $\varphi \rightarrow p$

\[ \square p \]

By premises I1 and I2, $\varphi$ is an invariant of the system. That is, all reachable states satisfy $\varphi$. Since, by premise I3, $\varphi$ implies $p$, it follows that $p$ is also a $D$-invariant.
Possible Domains for the Ranking Function

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

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

of \(\mathcal{A}\)-elements.

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

Given two well-founded domains \((A_1, \succ_1)\) and \((A_2, \succ_2)\), we construct a composite well-founded domain.

The lexicographic product \(A_1 \times \text{lex} \ A_2\) is the well-founded domain \((A, \succ)\), where \(A = A_1 \times A_2\) and
\[
(a_1, a_2) \succ \text{lex} (b_1, b_2) \iff (a_1 \succ_1 b_1) \lor (a_1 = b_1 \land a_2 \succ_2 b_2)
\]

**Claim 1.** If both \((A_1, \succ_1)\) and \((A_2, \succ_2)\) are well-founded, then so is \(A_1 \times \text{lex} A_2\).

**Proof** It is sufficient to show that \(A_1 \times \text{lex} A_2\) is well-founded.
Assume to the contrary, that there exists an infinitely descending sequence
\[
(a_1, b_1) \succ \text{lex} (a_2, b_2) \succ \text{lex} \cdots
\]
From the definition of \(\succ \text{lex}\) it follows that the sequence of first pair members satisfies \(a_1 \succeq_1 a_2 \succeq_1 \cdots\). Since \(A_1\) is well founded, it follows that there exists some position \(k\) such that \(a_k = a_{k+1} = \cdots\). Therefore, the sequence \(b_k \succ_2 b_{k+1} \succ_2 \cdots\) must be infinitely descending, contradicting the well-foundedness of \(A_2\). ☐
Rule WELL

For a well-founded domain \((A, \succ)\),
Helpfulness function \(h : J \mapsto \Sigma \mapsto \text{boolean}\),
assertions \(p, q\),
and ranking function \(\delta : \Sigma \mapsto A\)

W1. \(p \land \varphi \Rightarrow q \lor \bigvee_{\tau \in J} h[\tau]\)

For each \(\tau \in J\)

W2. \(h[\tau] \land \rho \Rightarrow (h[\tau] \land \delta = \delta') \lor q' \lor \left(\delta \succ \delta' \land \bigvee_{t \in J} h[t]'ight)\)

W3. \(h[\tau] \land \rho[\tau] \Rightarrow q' \lor \left(\delta \succ \delta' \land \bigvee_{t \in J} h[t]'ight)\)

W4. \(h[\tau] \Rightarrow En[\tau]\)

\(p \Rightarrow \Diamond q\)
Soundness of Rule **WELL**

**Claim 2.** Rule **WELL** is sound for proving the response property $p \Rightarrow \Diamond q$.

**Proof** Assume that the premises of rule **WELL** are valid. Let $\sigma : s_0, s_1, \ldots$ be a computation of $D$ and let $p$ hold at position $j$. We have to show that there exists a position $k \geq j$ such that $q$ holds at position $k$.

Assume to the contrary, that no position beyond $j$ satisfies $q$. By premise W1, state $s_j$ must satisfy $h[\tau]$, for some $\tau \in J$. Let us denote by $\tau_j$ the transition $\tau$ such that $h[\tau]$ holds at state $s_j$, and by $d_j \in A$ the value of $\delta$ at state $s_j$. By premise W2, the successor state $s_{j+1}$ must also satisfy $h[\tau]$, for some $\tau \in J$. Denote this transition by $\tau_{j+1}$. In this way we proceed to establish an infinite sequence of transitions $\tau_j, \tau_{j+1}, \ldots$ where, for each $k \geq j$, $\tau_k \in J$ and $s_k \models h[\tau_k]$. Let us denote by $d_j, d_{j+1}, \ldots$ the sequence of values of the ranking function at the respective states. By premises W2 and W3, the sequence $d_j \succeq d_{j+1} \succeq \cdots$ is non-increasing. Since this is an infinite non-increasing sequence over a well-founded domain, there must exist an index $n$, such that $d_n = d_{n+1} = \cdots$, and consequently (due to W2) $\tau_n = \tau_{n+1} = \cdots$.

By premise W3, we can have $\tau_n = \tau_{n+1} = \cdots$ only if transition $\tau_n$ is never taken beyond position $n$. On the other hand, due to W4, transition $\tau_n$ is continuously
enabled beyond position $n$. Thus, $\sigma$ violates the justice requirement associated with transition $\tau_n$, and therefore is not a computation, contrary to our original assumption.

We conclude that there must exists a position $k \geq j$ satisfying $q$. □
Unity Program as an FTS

Let

\[ P : \text{Program declarations initially-section } a_1, \ldots, a_k \text{ end} \]

be a Unity program, where \( a_1, \ldots, a_k \) are the assignment statements. We construct an FTS \( D_P \) as follows:

- For the program variables \( V \), we take all the variables declared in the declaration section.
- For the initial condition \( \Theta \), we take the conjunction of all the constraints imposed in the initially section.
- For the transitions, we take \( T = \{a_1, \ldots, a_k\} \). For each assignment \( a_i \) we define its transition relation, as follows:
  - If \( a_i \) has the form \( \vec{u} := \vec{e} \), then \( \rho[a_i] : \vec{u}' = \vec{e} \land pres(V - \vec{u}) \).
  - If \( a_i \) has the form \( \vec{u} := (\vec{e}_1 \text{ if } p_1) \sim \cdots \sim (\vec{e}_m \text{ if } p_m) \), then \( \rho[a_i] \) is given by

\[
\left( p_1 \land \vec{u} = \vec{e}_1 \lor \cdots \lor p_m \land \vec{u} = \vec{e}_m \lor pres(\vec{u}) \land \bigwedge_{i=1}^{m} \neg p_i \right) \land pres(V - \vec{u})
\]

- \( J = T \) and \( C = \emptyset \).
Example: A GCD Program

Consider the following program

Program GCD
  declare  a, b, x, y : natural
  initially x = a ∧ y = b ∧ a > 0 ∧ b > 0
  assign a1 : x := x − y if x > y
     □ a2 : y := y − x if y > x
end

Note that this program gives rise to two transitions. Their transition relations are respectively given by

ρ₁ : (x', y') = (x − y, y) ∧ x > y ∨ (x', y') = (x, y) ∧ x ≤ y
ρ₂ : (x', y') = (x, y − x) ∧ x < y ∨ (x', y') = (x, y) ∧ x ≥ y

We wish to prove for this program the properties:

Invariance  FP ⇒ (x = gcd(a, b))
Response   ◊ FP
An Auxiliary Rule for Partial Correctness

In many cases, the invariance property we wish to prove has the form $FP \Rightarrow p$. The following rule enables us to prove such properties directly.

**Rule PAR-COR**

For assertions $\varphi$ and $p$,

P1. $\Theta \rightarrow \varphi$

P2. $\varphi \land \rho \rightarrow \varphi'$

P3. $\left( \bigwedge_{\tau \in T} \rho[\tau](V, V) \right) \land \varphi(V) \rightarrow p(V)$

\[ FP \Rightarrow p \]
Proving $FP \Rightarrow (x = \gcd(a, b))$

As the auxiliary inductive assertion we pick

$$\varphi : (\gcd(x, y) = \gcd(a, b)) \land x > 0 \land y > 0.$$  

This gives rise to the following premises:

\begin{align*}
P1. & \quad x = a \land y = b \land a > 0 \land b > 0 \quad \rightarrow \quad (\gcd(x, y) = \gcd(a, b)) \land x > 0 \land y > 0 \\
P2_1. & \quad (\gcd(x, y) = \gcd(a, b)) \land x > 0 \land y > 0 \land (x' = (x > y) \ ? \ (x - y) : x) \quad \rightarrow \\
& \quad (\gcd(x', y) = \gcd(a, b)) \land x' > 0 \land y > 0
\end{align*}
Optimized Versions of Rules

In view of the special structure of Unity programs, it is possible to derive a specialized form of the main rules. For an assignment of the form

\[ a : \vec{u} := (\vec{e}_1 \text{ if } p_1) \sim \cdots \sim (\vec{e}_m \text{ if } p_m), \]

we refer to \( \vec{u} := \vec{e}_i \text{ if } p_i \), \( i = 1, \ldots, m \) as the cases of assignment \( a \). An unconditional assignment \( \vec{u} := \vec{e} \) is considered to have a single case which is the assignment itself. We often may take a conjunction or disjunction over all cases within an assignment, or over all cases within the program.

For a case \( \kappa \), given by \( \vec{u}_\kappa := \vec{e}_\kappa \text{ if } p_\kappa \), we define the transition relation:

\[ \rho[\kappa] : p_\kappa \land \vec{u}'_\kappa = \vec{e}_\kappa \land \pres(V - \vec{u}_\kappa) \]

For the case that \( \kappa \) corresponds to a unconditional statement, we take \( p_\kappa = 1 \) (True).

In the following slides, we will reconsider the various rules and present an optimized version for them.
Optimized Version of \text{INV}

The optimized versions of rule \text{INV} is given as follows:

\begin{center}
\begin{tabular}{|c|}
\hline
\textbf{Rule INV} \\
For an assertion $\varphi$, \\
I1. $\Theta \rightarrow \varphi$ \\
I2. For all cases $\kappa \in P$: $\varphi \land \rho[\kappa] \rightarrow \varphi'$ \\
I3. $\varphi \rightarrow p$ \\
\hline
\end{tabular}
\end{center}

Note that premise P2 has to be checked separately for each case $\kappa \in P$. In particular, we do not take into account the case that an assertion does not modify the state. This is because such cases certainly preserve the validity of $\varphi$. 
Optimized Version of Rule PAR-COR

Rule PAR-COR
For assertions $\varphi$ and $p$,

P1. $\Theta \rightarrow \varphi$

P2. For all cases $\kappa \in P$: $\varphi \land \rho[\kappa] \rightarrow \varphi'$

P3. $\varphi(V) \land \left( \bigwedge_{\kappa \in P} (p_\kappa(V) \rightarrow \vec{u}_\kappa = \vec{e}_\kappa(V)) \right) \rightarrow p(V)$

\[ FP \Rightarrow p \]

In addition to distribution of premise P2 across the various cases, this version contains an optimized version of premise P3. According to this version the fixed point $FP$ is characterized by the conjunction $\bigwedge_{\kappa \in P} (p_\kappa(V) \rightarrow \vec{u}_\kappa = \vec{e}_\kappa(V))$. The conjunction claims that every case preserves the state.
An Optimized Version of Rule **WELL**

**Rule WELL**

For a well-founded domain \((A, \succ)\),

Helpfulness function

\[
h : J \mapsto \Sigma \mapsto \text{boolean},
\]

assertions

\[
p, q,
\]

a \(D\)-invariant

\[
\varphi,
\]

and ranking function

\[
\delta : \Sigma \mapsto A
\]

W1. \[
p \land \varphi \implies q \lor \bigvee_{\tau \in J} h[\tau]
\]

For each assignment \(a \in P\)

W2. \[
\forall \kappa \in P :: h[a] \land \rho[\kappa] \implies (h[a]' \land \delta = \delta') \lor q' \lor \left(\delta \succ \delta' \land \bigvee_{t \in J} h[t]'\right)
\]

W3. \[
\forall \kappa \in a :: h[a] \land \rho[\kappa] \implies q' \lor \left(\delta \succ \delta' \land \bigvee_{t \in J} h[t]'\right)
\]

W4. \[
h[a] \implies \bigvee_{\kappa \in a} p_{\kappa}
\]

\[
p \implies \Diamond q
\]