Universidade Federal de Pernambuco
Centro de Informática

Circus Type Rules

Manuela de Almeida Xavier

June, 2006

Contents

1 Type Rules

1.1 Program

$$\frac{(\mathit{ExtractProcDefs}\ \mathit{cpl}) \rhd \mathit{cpl} : \textbf{CircusParagraphList}}{\Gamma_{\emptyset} \rhd \mathit{cpl} : \textbf{Program}}$$

1.2 List of *Circus* Paragraphs

$$\frac{\Gamma \rhd \mathit{cp} : \textbf{CircusParagraph}}{\Gamma \rhd \mathit{cp} : \textbf{CircusParagraphList}} \qquad \frac{\begin{array}{c}\Gamma \rhd \mathit{cp} : \textbf{CircusParagraph} \\ (\Gamma \oplus (\mathit{Defs}\ \mathit{cp}\ \Gamma)) \rhd \mathit{cpl} : \textbf{CircusParagraphList}\end{array}}{\Gamma \rhd \mathit{cp}\ \mathit{cpl} : \textbf{CircusParagraphList}}$$

1.3 *Circus* Paragraphs

$$\frac{(\mathit{NewDefs}\ p\ \Gamma.\mathit{defNames}) \qquad \Gamma \rhd p : \textbf{ZParagraph}}{\Gamma \rhd p : \textbf{CircusParagraph}}$$

$$\frac{n \notin \Gamma.\mathit{defNames} \qquad \Gamma \rhd \mathit{cs} : \textbf{CSExpression}}{\Gamma \rhd \textbf{chanset}\ n == \mathit{cs} : \textbf{CircusParagraph}}$$

$$\frac{\Gamma \rhd \mathit{cd} : \textbf{CDeclaration}}{\Gamma \rhd \textbf{channel}\ \mathit{cd} : \textbf{CircusParagraph}} \qquad \frac{\Gamma \rhd p : \textbf{ProcessDeclaration}}{\Gamma \rhd p : \textbf{CircusParagraph}}$$

1.4 Declaration of Channels

$$\frac{\mathit{NoRep}\ \mathit{ln} \qquad (\mathit{NotInto}\ \mathit{ln}\ \Gamma.\mathit{defNames})}{\Gamma \rhd \mathit{ln} : \textbf{CDeclaration}} \qquad \frac{\begin{array}{c}\Gamma \rhd \mathit{ln} : \textbf{CDeclaration} \\ \Gamma \rhd e : \textbf{Expression}(\mathbb{P}\ T)\end{array}}{\Gamma \rhd (\mathit{ln} : e) : \textbf{CDeclaration}}$$

$$\frac{\begin{array}{c}(\mathit{NoRep}\ \mathit{ln}_1) \\ (\Gamma \oplus (\mathit{zDefs} = \mathit{ExtractGenTypes}\ \mathit{ln}_1)) \rhd (\mathit{ln}_2 : e) : \textbf{CDeclaration}\end{array}}{\Gamma \rhd ([\mathit{ln}_1]\mathit{ln}_2 : e) : \textbf{CDeclaration}}$$

$$\frac{\Gamma \rhd \mathit{cd}_1 : \textbf{CDeclaration} \qquad (\Gamma \oplus \mathit{Chan}\ \mathit{cd}_1) \rhd \mathit{cd}_2 : \textbf{CDeclaration}}{\Gamma \rhd \mathit{cd}_1;\ \mathit{cd}_2 : \textbf{CDeclaration}}$$

1.5 Expression of Channel Sets

$$\frac{}{\Gamma \rhd \{\!|\ |\!\} : \mathbf{CSExpression}} \qquad \frac{(\mathit{Into}\ \mathit{ln}\ \ \mathrm{dom}(\Gamma.\mathit{channels}))}{\Gamma \rhd \{\!|\mathit{ln}|\!\} : \mathbf{CSExpression}} \qquad \frac{n \in \mathrm{dom}(\Gamma.\mathit{chansets})}{\Gamma \rhd n : \mathbf{CSExpression}}$$

$$\frac{\Gamma \rhd \mathit{ce}_1 : \mathbf{CSExpression} \qquad \Gamma \rhd \mathit{ce}_2 : \mathbf{CSExpression}}{\Gamma \rhd \mathit{ce}_1 \cup \mathit{ce}_2 : \mathbf{CSExpression}}$$

$$\frac{\Gamma \rhd \mathit{ce}_1 : \mathbf{CSExpression} \qquad \Gamma \rhd \mathit{ce}_2 : \mathbf{CSExpression}}{\Gamma \rhd \mathit{ce}_1 \cap \mathit{ce}_2 : \mathbf{CSExpression}}$$

$$\frac{\Gamma \rhd \mathit{ce}_1 : \mathbf{CSExpression} \qquad \Gamma \rhd \mathit{ce}_2 : \mathbf{CSExpression}}{\Gamma \rhd \mathit{ce}_1 \setminus \mathit{ce}_2 : \mathbf{CSExpression}}$$

1.6 Process Declaration

$$\frac{\begin{array}{c} n \notin \Gamma.\mathit{defNames} \qquad \Gamma \rhd p : \mathbf{ProcessDefinition} \\ (\mathit{DeclareNewChans}\ (\mathit{FindImplicitChans}\ p\ \Gamma)\ \Gamma) \end{array}}{\Gamma \rhd \mathbf{process}\ n \mathrel{\widehat=} p : \mathbf{ProcessDeclaration}}$$

$$\frac{\begin{array}{c} (n \notin \Gamma.\mathit{defNames}) \qquad (\mathit{NoRep}\ \mathit{ln}) \qquad \Gamma' \rhd p : \mathbf{ProcessDefinition} \\ (\mathit{DeclareNewChans}\ (\mathit{FindImplicitChans}\ p\ \Gamma)\ \Gamma) \end{array}}{\Gamma \rhd \mathbf{process}\ n[\mathit{ln}]\ \mathrel{\widehat=} p : \mathbf{Generic\_Process}(\mathit{ln})}$$

$$\mathbf{where}\ \Gamma' = (\Gamma \oplus (\mathit{localZDefs} = \mathit{ExtractGenTypes}\ \mathit{ln}))$$

$$\frac{\Gamma \rhd d : \mathbf{Declaration} \qquad (\Gamma \oplus (\mathit{params} = \mathit{Decs}\ \ d)) \rhd p : \mathbf{Process}}{\Gamma \rhd d \bullet p : \mathbf{Process\_Parametrised}(d)}$$

$$\frac{\Gamma \rhd d : \mathbf{Declaration} \qquad \Gamma \rhd p : \mathbf{Process}}{\Gamma \rhd d \odot p : \mathbf{Process\_Indexing}(d)}$$

$$\frac{\Gamma \rhd p : \mathbf{Generic\_Process}(\mathit{ln})}{\Gamma \rhd p : \mathbf{ProcessDeclaration}} \qquad \frac{\Gamma \rhd p : \mathbf{Process\_Parametrised}(d)}{\Gamma \rhd p : \mathbf{ProcessDefinition}}$$

$$\frac{\Gamma \rhd p : \mathbf{Process\_Indexing}(d)}{\Gamma \rhd p : \mathbf{ProcessDefinition}} \qquad \frac{\Gamma \rhd p : \mathbf{Process}}{\Gamma \rhd p : \mathbf{ProcessDefinition}}$$

1.7 List of Process Paragraphs

$$\frac{\Gamma \rhd p : \textbf{PParagraph}}{\Gamma \rhd p : \textbf{PParagraphList}}$$

$$\frac{\begin{array}{c}\Gamma \rhd p : \textbf{PParagraph} \qquad (\Gamma \oplus \Gamma') \rhd pl : \textbf{PParagraphList} \\ \mathit{NotRedeclare}((\Gamma.\mathit{localDefNames}), (\Gamma'.\mathit{localDefNames}))\end{array}}{\Gamma \rhd p\ pl : \textbf{PParagraphList}}$$

$$\textbf{where } \Gamma' = (\mathit{DefsP}\ p\ \Gamma)$$

1.8 Processes

$$\frac{\begin{array}{c}\Gamma_1 \rhd pl_1 : \textbf{PParagraphList} \qquad \Gamma_2 \rhd sc : \textbf{StateParagraph}(d) \\ \Gamma_3 \rhd pl_2 : \textbf{PParagraphList} \qquad \Gamma_4 \rhd a : \textbf{Action} \\ \mathit{NotRedeclare}((\Gamma_1.\mathit{localDefNames}), (\Gamma'.\mathit{localDefNames})) \\ \mathit{NotRedeclare}((\Gamma_2.\mathit{localDefNames}), (\Gamma''.\mathit{localDefNames})) \\ \mathit{NotRedeclare}((\Gamma_3.\mathit{localDefNames}), (\Gamma'''.\mathit{localDefNames}))\end{array}}{\Gamma \rhd \textbf{begin}\ pl_1\ \textbf{state}\ sc\ pl_2\ \bullet\ a\ \textbf{end} : \textbf{Process}}$$

$$\begin{array}{l}\textbf{where } \Gamma_1 = \Gamma \oplus (\mathit{ExtractActDefs}\ pl_1) \oplus (\mathit{ExtractActDefs}\ pl_2), \\ \quad \Gamma_2 = \Gamma_1 \oplus \Gamma',\ \Gamma_3 = \Gamma_2 \oplus \Gamma'',\ \Gamma_4 = \Gamma_3 \oplus \Gamma''', \\ \quad \Gamma' = (\mathit{DefsPL}\ pl_1\ \Gamma_1),\ \Gamma'' = (\mathit{DefsState}\ sc\ d), \\ \quad \Gamma''' = (\mathit{DefsPL}\ pl_2\ \Gamma_3),\end{array}$$

$$\frac{n \in \Gamma.\mathit{processes} \qquad (\mathit{IsNormalProc}\ n\ \Gamma)}{\Gamma \rhd n : \textbf{Process}}$$

$$\frac{\Gamma \rhd p : \textbf{Process} \qquad \Gamma \rhd cs : \textbf{CSExpression}}{\Gamma \rhd p \setminus cs : \textbf{Process}}$$

$$\frac{\Gamma \rhd p_1 : \textbf{Process} \qquad \Gamma \rhd p_2 : \textbf{Process}}{\Gamma \rhd p_1;\ p_2 : \textbf{Process}}$$

$$\frac{\Gamma \rhd p_1 : \textbf{Process} \qquad \Gamma \rhd p_2 : \textbf{Process}}{\Gamma \rhd p_1 \Box p_2 : \textbf{Process}}$$

$$\frac{\Gamma \rhd p_1 : \textbf{Process} \qquad \Gamma \rhd p_2 : \textbf{Process}}{\Gamma \rhd p_1 \sqcap p_2 : \textbf{Process}}$$

$$\frac{\Gamma \rhd p_1 : \textbf{Process} \qquad \Gamma \rhd p_2 : \textbf{Process}}{\Gamma \rhd p_1 \interleave p_2 : \textbf{Process}}$$

$$\frac{\begin{array}{c}\Gamma \rhd p_1 : \textbf{Process} \\ \Gamma \rhd p_2 : \textbf{Process} \qquad \Gamma \rhd cs : \textbf{CSExpression}\end{array}}{\Gamma \rhd p_1 \llbracket\, cs \,\rrbracket\, p_2 : \textbf{Process}}$$

$$\frac{\begin{array}{c}\Gamma \rhd d \bullet p : \textbf{Process\_Parametrised}(d) \\ \Gamma \rhd le : \textbf{ExpressionList}(T) \qquad (\mathit{Check}\ (\mathit{Decs}\ d)\ T)\end{array}}{\Gamma \rhd (d \bullet p)(le) : \textbf{Process}}$$

$$\frac{n \in \mathrm{dom}(\Gamma.\mathit{parProcesses}) \quad \Gamma \rhd \mathit{le} : \mathbf{ExpressionList}(T) \qquad (\mathit{Check}\ (\Gamma.\mathit{parProcesses}\ n)\ T)}{\Gamma \rhd n(\mathit{le}) : \mathbf{Process}}$$

$$\frac{\Gamma \rhd d \odot p : \mathbf{Process\_Indexing}(d) \quad \Gamma \rhd \mathit{le} : \mathbf{ExpressionList}(T) \qquad (\mathit{Check}\ (\mathit{Decs}\ d)\ T)}{\Gamma \rhd (d \odot p)\lfloor \mathit{le} \rfloor : \mathbf{Process}}$$

$$\frac{n \in \mathrm{dom}(\Gamma.\mathit{indexProcesses}) \qquad \Gamma \rhd \mathit{le} : \mathbf{ExpressionList}(T) \quad (\mathit{Check}\ (\Gamma.\mathit{indexProcesses}\ n)\ T)}{\Gamma \rhd n\lfloor \mathit{le} \rfloor : \mathbf{Process}}$$

$$\frac{n \in \mathrm{dom}(\Gamma.\mathit{genProcesses}) \qquad \Gamma \rhd \mathit{le} : \mathbf{ExpressionList}(\mathbb{P}\ T) \quad \sharp \mathit{le} == \sharp(\Gamma.\mathit{genProcesses}\ n)}{\Gamma \rhd n[\mathit{le}] : \mathbf{Process}}$$

$$\frac{\begin{array}{cc} (n \in \mathrm{dom}(\Gamma.\mathit{genProcesses})) & (n \in \mathrm{dom}(\Gamma.\mathit{parProcesses})) \\ \Gamma \rhd \mathit{le}_1 : \mathbf{ExpressionList}(\mathbb{P}\ T) & \sharp \mathit{le}_1 == \sharp(\Gamma.\mathit{genProcesses}\ n) \\ \Gamma \rhd \mathit{le}_2 : \mathbf{ExpressionList}(U) & (\mathit{Check}\ (\Gamma.\mathit{parProcesses}\ n)\ U) \end{array}}{\Gamma \rhd n[\mathit{le}_1](\mathit{le}_2) : \mathbf{Process}}$$

$$\frac{\begin{array}{cc} (n \in \mathrm{dom}(\Gamma.\mathit{genProcesses})) & (n \in \mathrm{dom}(\Gamma.\mathit{indexProcesses})) \\ \Gamma \rhd \mathit{le}_1 : \mathbf{ExpressionList}(\mathbb{P}\ T) & \sharp \mathit{le}_1 == \sharp(\Gamma.\mathit{genProcesses}\ n) \\ \Gamma \rhd \mathit{le}_2 : \mathbf{ExpressionList}(U) & (\mathit{Check}\ (\Gamma.\mathit{indexProcesses}\ n)\ U) \end{array}}{\Gamma \rhd n[\mathit{le}_1]\lfloor \mathit{le}_2 \rfloor : \mathbf{Process}}$$

$$\frac{\begin{array}{c} \Gamma \rhd p : \mathbf{Process} \qquad \sharp \mathit{ln}_1 == \sharp \mathit{ln}_2 \qquad (\mathit{Into}\ \mathit{ln}_1\ \mathrm{dom}(\Gamma.\mathit{channels})) \\ (\mathit{Into}\ \mathit{ln}_2\ \mathrm{dom}(\Gamma.\mathit{channels})) \qquad (\mathit{CheckChansRenaming}\ \mathit{ln}_1\ \mathit{ln}_2\ \Gamma) \end{array}}{\Gamma \rhd p[\mathit{ln}_1 := \mathit{ln}_2] : \mathbf{Process}}$$

$$\frac{\Gamma \rhd d : \mathbf{Declaration} \quad (\mathit{FiniteDecs}\ d) \qquad (\Gamma \oplus (\mathit{params} = \mathit{Decs}\ d)) \rhd p : \mathbf{Process}}{\Gamma \rhd\ \fatsemi\ d \bullet p : \mathbf{Process}}$$

$$\frac{\Gamma \rhd d : \mathbf{Declaration} \quad (\mathit{FiniteDecs}\ d) \qquad (\Gamma \oplus (\mathit{params} = \mathit{Decs}\ d)) \rhd p : \mathbf{Process}}{\Gamma \rhd\ \Box\ d \bullet p : \mathbf{Process}}$$

$$\frac{\Gamma \rhd d : \mathbf{Declaration} \quad (\mathit{FiniteDecs}\ d) \qquad (\Gamma \oplus (\mathit{params} = \mathit{Decs}\ d)) \rhd p : \mathbf{Process}}{\Gamma \rhd\ \sqcap\ d \bullet p : \mathbf{Process}}$$

$$\frac{(\mathit{FiniteDecs}\ d) \qquad \begin{array}{c}\Gamma \rhd d : \mathbf{Declaration} \\ (\Gamma \oplus (\mathit{params} = \mathit{Decs}\ d)) \rhd p : \mathbf{Process}\end{array}}{\Gamma \rhd\ ||| \ d \bullet p : \mathbf{Process}}$$

$$\frac{\begin{array}{cc}\Gamma \rhd d : \mathbf{Declaration} & \Gamma \rhd cs : \mathbf{CSExpression} \\ (\mathit{FiniteDecs}\ d) & (\Gamma \oplus (\mathit{params} = \mathit{Decs}\ d)) \rhd p : \mathbf{Process}\end{array}}{\Gamma \rhd\ \llbracket\ cs \rrbracket\ d \bullet p : \mathbf{Process}}$$

1.9 Expression of Name Sets

$$\frac{}{\Gamma \rhd \{\ \} : \mathbf{NSExpression}} \qquad \frac{(\mathit{Into}\ \mathit{ln}\ \ \mathrm{dom}(\Gamma.\mathit{localVars}))}{\Gamma \rhd \{\mathit{ln}\} : \mathbf{NSExpression}} \qquad \frac{n \in (\Gamma.\mathit{namesets})}{\Gamma \rhd n : \mathbf{NSExpression}}$$

$$\frac{\Gamma \rhd \mathit{ns}_1 : \mathbf{NSExpression} \qquad \Gamma \rhd \mathit{ns}_2 : \mathbf{NSExpression}}{\Gamma \rhd \mathit{ns}_1 \cup \mathit{ns}_2 : \mathbf{NSExpression}}$$

$$\frac{\Gamma \rhd \mathit{ns}_1 : \mathbf{NSExpression} \qquad \Gamma \rhd \mathit{ns}_2 : \mathbf{NSExpression}}{\Gamma \rhd \mathit{ns}_1 \cap \mathit{ns}_2 : \mathbf{NSExpression}}$$

$$\frac{\Gamma \rhd \mathit{ns}_1 : \mathbf{NSExpression} \qquad \Gamma \rhd \mathit{ns}_2 : \mathbf{NSExpression}}{\Gamma \rhd \mathit{ns}_1 \setminus \mathit{ns}_2 : \mathbf{NSExpression}}$$

1.10 Process Paragraphs

$$\frac{(\mathit{NewDefs}\ p\ (\Gamma.\mathit{localDefNames})) \qquad \Gamma \rhd p : \mathbf{ZParagraph}}{\Gamma \rhd p : \mathbf{PParagraph}}$$

$$\frac{n \notin (\Gamma.\mathit{localDefNames}) \qquad \Gamma \rhd a : \mathbf{ActionDefinition}}{\Gamma \rhd n \mathrel{\widehat=} a : \mathbf{PParagraph}}$$

$$\frac{n \notin (\Gamma.\mathit{localDefNames}) \qquad \Gamma \rhd \mathit{ns} : \mathbf{NSExpression}}{\Gamma \rhd \mathbf{nameset}\ n == \mathit{ns} : \mathbf{PParagraph}}$$

1.11 Action Definitions

$$\frac{\Gamma \rhd a : \mathbf{ParamAction}}{\Gamma \rhd a : \mathbf{ActionDefinition}} \qquad \frac{\Gamma \rhd a : \mathbf{Action}}{\Gamma \rhd a : \mathbf{ActionDefinition}}$$

1.12 Actions

$$\frac{\Gamma \rhd d : \mathbf{Declaration} \qquad (\Gamma \oplus (params = Decs\ \ d)) \rhd a : \mathbf{Action}}{\Gamma \rhd d \bullet a : \mathbf{Action\_Parametrised}(d)}$$

$$\frac{\Gamma \rhd p : \mathbf{Action\_Parametrised}(d)}{\Gamma \rhd p : \mathbf{ParamAction}} \qquad \frac{\Gamma \rhd sc : \mathbf{Schema\text{-}Exp}}{\Gamma \rhd sc : \mathbf{Action}}$$

$$\frac{n \in (\Gamma.actions) \qquad n \notin (\Gamma.parActions)}{\Gamma \rhd n : \mathbf{Action}} \qquad \frac{\Gamma \rhd c : \mathbf{ParCommand}}{\Gamma \rhd c : \mathbf{Action}}$$

$$\frac{}{\Gamma \rhd Skip : \mathbf{Action}} \qquad \frac{}{\Gamma \rhd Stop : \mathbf{Action}} \qquad \frac{}{\Gamma \rhd Chaos : \mathbf{Action}}$$

$$\frac{\begin{array}{c}\Gamma \rhd a : \mathbf{Action} \qquad \sharp ln_1 == \sharp ln_2 \qquad (Into\ \ ln_1\ \ \mathrm{dom}(\Gamma.localVars)) \\ (Into\ \ ln_2\ \ \mathrm{dom}(\Gamma.localVars)) \qquad (CheckVarsRenaming\ \ ln_1\ \ ln_2\ \Gamma)\end{array}}{\Gamma \rhd a[ln_1 := ln_2] : \mathbf{Action}}$$

$$\frac{\Gamma \rhd c : \mathbf{Communication} \qquad (\Gamma \oplus VarsC\ \ c\ \Gamma) \rhd a : \mathbf{Action}}{\Gamma \rhd c \to a : \mathbf{Action}}$$

$$\frac{\Gamma \rhd p : \mathbf{Predicate} \qquad \Gamma \rhd a : \mathbf{Action}}{\Gamma \rhd p\ \&\ a : \mathbf{Action}}$$

$$\frac{\Gamma \rhd a_1 : \mathbf{Action} \qquad \Gamma \rhd a_2 : \mathbf{Action}}{\Gamma \rhd a_1;\ a_2 : \mathbf{Action}} \qquad \frac{\Gamma \rhd a_1 : \mathbf{Action} \qquad \Gamma \rhd a_2 : \mathbf{Action}}{\Gamma \rhd a_1 \Box a_2 : \mathbf{Action}}$$

$$\frac{\Gamma \rhd a_1 : \mathbf{Action} \qquad \Gamma \rhd a_2 : \mathbf{Action}}{\Gamma \rhd a_1 \sqcap a_2 : \mathbf{Action}} \qquad \frac{\Gamma \rhd a_1 : \mathbf{Action} \qquad \Gamma \rhd a_2 : \mathbf{Action}}{\Gamma \rhd a_1 \interleave a_2 : \mathbf{Action}}$$

$$\frac{\begin{array}{c}\Gamma \rhd a_1 : \mathbf{Action} \qquad \Gamma \rhd a_2 : \mathbf{Action} \\ \Gamma \rhd ns_1 : \mathbf{NSExpression} \qquad \Gamma \rhd ns_2 : \mathbf{NSExpression}\end{array}}{\Gamma \rhd a_1 \| [ns_1 \mid ns_2] \| a_2 : \mathbf{Action}}$$

$$\frac{\begin{array}{c}\Gamma \rhd a_1 : \mathbf{Action} \qquad \Gamma \rhd a_2 : \mathbf{Action} \qquad \Gamma \rhd ns_1 : \mathbf{NSExpression} \\ \Gamma \rhd ns_2 : \mathbf{NSExpression} \qquad \Gamma \rhd cs : \mathbf{CSExpression}\end{array}}{\Gamma \rhd a_1 \llbracket ns_1 \mid cs \mid ns_2 \rrbracket a_2 : \mathbf{Action}}$$

$$\frac{\Gamma \rhd a : \mathbf{Action} \qquad \Gamma \rhd cs : \mathbf{CSExpression}}{\Gamma \rhd a \setminus cs : \mathbf{Action}}$$

$$\frac{n \in \mathrm{dom}(\Gamma.\mathit{parActions}) \qquad \Gamma \rhd \mathit{le} : \mathbf{ExpressionList}(T) \\ (\mathit{Check}\ (\Gamma.\mathit{parActions}\ n)\ T)}{\Gamma \rhd n(\mathit{le}) : \mathbf{Action}}$$

$$\frac{\Gamma \rhd (d \bullet a) : \mathbf{Action\_Parametrised}(d) \\ \Gamma \rhd \mathit{le} : \mathbf{ExpressionList}(T) \qquad (\mathit{Check}\ (\mathit{Decs}\ d)\ T)}{\Gamma \rhd (d \bullet a)(\mathit{le}) : \mathbf{Action}}$$

$$\frac{(\Gamma \oplus (\mathit{definedActs} = \{n\})) \rhd a : \mathbf{Action}}{\Gamma \rhd\ \mu\, n \bullet a : \mathbf{Action}}$$

$$\frac{\Gamma \rhd d : \mathbf{Declaration} \qquad (\mathit{FiniteDecs}\ d) \\ (\Gamma \oplus (\mathit{params} = \mathit{Decs}\ d)) \rhd a : \mathbf{Action}}{\Gamma \rhd\ \fatsemi\, d \bullet a : \mathbf{Action}}$$

$$\frac{\Gamma \rhd d : \mathbf{Declaration} \qquad (\mathit{FiniteDecs}\ d) \\ (\Gamma \oplus (\mathit{params} = \mathit{Decs}\ d)) \rhd a : \mathbf{Action}}{\Gamma \rhd\ \Box\ d \bullet a : \mathbf{Action}}$$

$$\frac{\Gamma \rhd d : \mathbf{Declaration} \qquad (\mathit{FiniteDecs}\ d) \\ (\Gamma \oplus (\mathit{params} = \mathit{Decs}\ d)) \rhd a : \mathbf{Action}}{\Gamma \rhd\ \sqcap\ d \bullet a : \mathbf{Action}}$$

$$\frac{\Gamma \rhd d : \mathbf{Declaration} \qquad (\mathit{FiniteDecs}\ d) \\ (\Gamma \oplus (\mathit{params} = \mathit{Decs}\ d)) \rhd a : \mathbf{Action}}{\Gamma \rhd\ |||\ d \bullet a : \mathbf{Action}}$$

$$\frac{\Gamma \rhd d : \mathbf{Declaration} \qquad (\Gamma \oplus (\mathit{params} = \mathit{Decs}\ d)) \rhd a : \mathbf{Action} \\ (\mathit{FiniteDecs}\ d) \qquad \Gamma \rhd \mathit{ns} : \mathbf{NSExpression}}{\Gamma \rhd\ |||\ d \bullet \|[\mathit{ns}]\| a : \mathbf{Action}}$$

$$\frac{\Gamma \rhd d : \mathbf{Declaration} \qquad (\mathit{FiniteDecs}\ d) \qquad \Gamma \rhd \mathit{ns} : \mathbf{NSExpression} \\ \Gamma \rhd \mathit{cs} : \mathbf{CSExpression} \qquad (\Gamma \oplus (\mathit{params} = \mathit{Decs}\ d)) \rhd a : \mathbf{Action}}{\Gamma \rhd\ [\![\ \mathit{cs}\ ]\!]\ d \bullet [\![\ \mathit{ns}\ ]\!]\ a : \mathbf{Action}}$$

1.13 Communication

$$\frac{n \in \mathrm{dom}(\Gamma.\mathit{channels}) \qquad \Gamma.\mathit{channels}\ n == \mathit{wt}}{\Gamma \rhd n : \mathbf{Communication}}$$

$$\frac{\begin{array}{c} n \in \mathrm{dom}(\Gamma.\mathit{channels}) \\ ((\Gamma.\mathit{channels}\ n) \neq \mathit{wt}) \qquad \sharp \mathit{cpl} \leq (\mathit{NumTypesChan}\ (\Gamma.\mathit{channels}\ n)) \\ (\mathit{NoConflicts}\ (\mathit{Extract}\ \mathit{cpl}\ T)) \qquad \Gamma \rhd \mathit{cpl} : \mathbf{CParameterList}(T) \end{array}}{\Gamma \rhd n\ \mathit{cpl} : \mathbf{Communication}}$$

$$\textbf{where}\ T = \Gamma.\mathit{channels}\ n$$

$$\frac{\begin{array}{c} n \in \mathrm{dom}(\Gamma.\mathit{genericChannels}) \qquad \Gamma \rhd \mathit{le} : \mathbf{ExpressionList}(\mathbb{P}\ T') \\ \sharp \mathit{le} == \sharp(\Gamma.\mathit{genericChannels}\ n) \qquad \sharp \mathit{cpl} \leq (\mathit{NumTypesChan}\ (\Gamma.\mathit{channels}\ n)) \\ (\mathit{NoConflicts}\ (\mathit{Extract}\ \mathit{cpl}\ T)) \qquad \Gamma' \rhd \mathit{cpl} : \mathbf{CParameterList}(T) \end{array}}{\Gamma \rhd n[\mathit{le}]\mathit{cpl} : \mathbf{Communication}}$$

$$\begin{array}{l} \textbf{where}\ \Gamma' = \Gamma \oplus (\mathit{InstantiateTypesGenChan}\ (\Gamma.\mathit{genericsChannels}\ n)\ \mathit{le}\ n\ T), \\ \qquad\quad T = \Gamma.\mathit{channels}\ n \end{array}$$

1.14 List of Communication Parameters

$$\frac{\begin{array}{c} \Gamma \rhd \mathit{cp} : \mathbf{CParameter}(\mathit{GetTypeHead}\ T) \\ (\Gamma \oplus (\mathit{ExtractVarsCP}\ \mathit{cp}\ T)) \rhd \mathit{cpl} : \mathbf{CParameterList}(\mathit{GetTypeTail}\ T) \end{array}}{\Gamma \rhd \mathit{cp}\ \mathit{cpl} : \mathbf{CParameterList}(T)}$$

$$\frac{\Gamma \rhd \mathit{cp} : \mathbf{CParameter}(T)}{\Gamma \rhd \mathit{cp} : \mathbf{CParameterList}(T)}$$

1.15 Communication Parameters

$$\frac{}{\Gamma \rhd ?\mathit{cp} : \mathbf{CParameter}(T)} \qquad \frac{(\Gamma \oplus (\mathit{ExtractCP}\ \mathit{cp}\ T)) \rhd p : \mathbf{Predicate}}{\Gamma \rhd (?\mathit{cp} : p) : \mathbf{CParameter}(T)}$$

$$\frac{\Gamma \rhd e : \mathbf{Expression}(T') \qquad T' == T}{\Gamma \rhd !e : \mathbf{CParameter}(T)} \qquad \frac{\Gamma \rhd e : \mathbf{Expression}(T') \qquad T' == T}{\Gamma \rhd .e : \mathbf{CParameter}(T)}$$

1.16 Parametrised Commands

$$\frac{\Gamma \rhd \mathit{pd} : \mathbf{QualifiedDeclaration} \qquad (\Gamma \oplus (\mathit{VarsParDec}\ \mathit{pd})) \rhd c : \mathbf{Command}}{\Gamma \rhd \mathit{pd} \bullet c : \mathbf{ParCommand}(\mathit{pd})}$$

$$\frac{\Gamma \rhd d : \mathbf{Declaration}}{\Gamma \rhd \mathbf{val}\ d : \mathbf{QualifiedDeclaration}} \qquad \frac{\Gamma \rhd d : \mathbf{Declaration}}{\Gamma \rhd \mathbf{res}\ d : \mathbf{QualifiedDeclaration}}$$

$$\frac{\Gamma \rhd d : \textbf{Declaration}}{\Gamma \rhd \textbf{valres}\ d : \textbf{QualifiedDeclaration}}$$

$$\frac{\Gamma \rhd pd_1 : \textbf{QualifiedDeclaration} \qquad \Gamma' \rhd pd_2 : \textbf{QualifiedDeclaration}}{\Gamma \rhd pd_1;\ pd_2 : \textbf{QualifiedDeclaration}}$$

$$\textbf{where}\ \Gamma' = \Gamma \oplus (\mathit{VarsParDec}\ pd_1)$$

1.17 Commands

$$\frac{\begin{array}{c}(\mathit{NoRep}\ \mathit{ln}) \qquad (\mathit{Into}\ \mathit{ln}\ \operatorname{dom}(\Gamma.\mathit{localVars})) \\ \Gamma \rhd \mathit{pre} : \textbf{Predicate} \qquad \Gamma \rhd \mathit{post} : \textbf{Predicate}\end{array}}{\Gamma \rhd \mathit{ln} : [\mathit{pre}, \mathit{post}] : \textbf{Command}}$$

$$\frac{\begin{array}{c}\mathit{NoRep}\ \mathit{ln} \qquad (\mathit{Into}\ \mathit{ln}\ \operatorname{dom}(\Gamma.\mathit{localVars})) \\ \sharp \mathit{ln} == \sharp \mathit{le} \qquad (\mathit{Verify}\ \mathit{ln}\ \mathit{le}\ \Gamma)\end{array}}{\Gamma \rhd \mathit{ln} := \mathit{le} : \textbf{Command}}$$

$$\frac{\Gamma \rhd d : \textbf{Declaration} \qquad (\Gamma \oplus (\mathit{VarsDec}\ d)) \rhd a : \textbf{Action}}{\Gamma \rhd \textbf{var}\ d \bullet a : \textbf{Command}}$$

$$\frac{\Gamma \rhd p : \textbf{Predicate}}{\Gamma \rhd [p] : \textbf{Command}} \qquad \frac{\Gamma \rhd p : \textbf{Predicate}}{\Gamma \rhd \{p\} : \textbf{Command}}$$

$$\frac{\begin{array}{c}\Gamma \rhd d \bullet c : \textbf{ParCommand}(d) \\ \Gamma \rhd \mathit{le} : \textbf{ExpressionList}(T) \qquad \mathit{Check}((\mathit{DecsParDec}\ d)\ T)\end{array}}{\Gamma \rhd (d \bullet c)(\mathit{le}) : \textbf{Command}}$$

$$\frac{\Gamma \rhd \mathit{gl} : \textbf{GActionList}}{\Gamma \rhd \textbf{if}\ \mathit{gl}\ \textbf{fi} : \textbf{Command}} \qquad \frac{\Gamma \rhd g : \textbf{GAction}}{\Gamma \rhd g : \textbf{GActionList}}$$

$$\frac{\Gamma \rhd g : \textbf{GAction} \qquad \Gamma \rhd \mathit{gl} : \textbf{GActionList}}{\Gamma \rhd g \ \square\ \mathit{gl} : \textbf{GActionList}}$$

$$\frac{\Gamma \rhd p : \textbf{Predicate} \qquad \Gamma \rhd a : \textbf{Action}}{\Gamma \rhd p \rightarrow a : \textbf{GAction}}$$

1.18 Association between Z and *Circus* Type Rules

$$\frac{\Gamma \rhd e : \mathbf{Expression}(T_1) \qquad \Gamma \rhd le : \mathbf{ExpressionList}(T_2)}{\Gamma \rhd e, le : \mathbf{ExpressionList}(T_1, T_2)}$$

$$\frac{\Gamma \rhd e : \mathbf{Expression}(T)}{\Gamma \rhd e : \mathbf{ExpressionList}(T)}$$

$$\frac{(\mathit{CircusToZTypeEnv}\ \Gamma) \vdash^{\mathcal{D}} s \fatsemi \sigma}{\Gamma \rhd s : \mathbf{StateParagraph}(\sigma)} \qquad \frac{(\mathit{CircusToZTypeEnv}\ \Gamma) \vdash^{\mathcal{D}} p \fatsemi \sigma}{\Gamma \rhd p : \mathbf{ZParagraph}}$$

$$\frac{(\mathit{CircusToZTypeEnv}\ \Gamma) \vdash^{\mathcal{E}} e \fatsemi \tau}{\Gamma \rhd e : \mathbf{Expression}(\tau)} \qquad \frac{(\mathit{CircusToZTypeEnv}\ \Gamma) \vdash^{\mathcal{E}} sc \fatsemi \tau}{\Gamma \rhd sc : \mathbf{Schema\text{-}Exp}}$$

$$\frac{(\mathit{CircusToZTypeEnv}\ \Gamma) \vdash^{\mathcal{P}} p}{\Gamma \rhd p : \mathbf{Predicate}} \qquad \frac{(\mathit{CircusToZTypeEnv}\ \Gamma) \vdash^{D\mathcal{E}} d \fatsemi \sigma}{\Gamma \rhd d : \mathbf{Declaration}}$$

2 Function Definitions

2.1 Chan

Function that extracts all channel declarations of a CDeclaration, and returns an environment that contains the extracted channels.

$$Chan : CDec \rightarrow TEnv$$

$$\begin{array}{ll}
Chan(n) & = (channels = \{n \mapsto wt\}) \\
Chan(n, ln) & = (Chan\ n) \oplus (Chan\ ln) \\
Chan(n : e) & = (channels = \{n \mapsto e\}) \\
Chan(n, ln : e) & = (Chan\ n : e) \oplus (Chan\ ln : e) \\
Chan([ln_1]ln_2 : e) & = \textbf{if}\ (RefGenParam\ e\ (Set\ ln_1)) \\
 & \quad \textbf{then}\ (genericChannels = (GenChans\ ln_2\ (ExtractGenTypes\,\mathrm{ln}_1))) \oplus \\
 & \qquad\qquad\qquad\qquad \oplus\ (Chan\ ln_2 : e) \\
 & \quad \textbf{else}\ (Chan\ ln_2 : e) \\
Chan(c;\ cd) & = (Chan\ c)\ \oplus\ (Chan\ cd)
\end{array}$$

2.2 Check

Function that verifies if the number of elements of a pair set is equals to the number of elements of a type list, and if each first element of the pair has the same type of the corresponding element of the type list.

$$Check : \mathbb{P}(NAME \times T) \rightarrow SeqType \rightarrow Bool$$

$$Check\ ds\ ts \Leftrightarrow \sharp ds == \sharp ts \wedge Compare\ ds\ ts$$

2.3 CheckChansRenaming

Function that verifies the channels of a process renaming. Each name within the second list has to have the same type of the corresponding channel of the first list passed as argument of the function.

$$CheckChansRenaming : SeqName \nrightarrow SeqName \nrightarrow TEnv \nrightarrow Bool$$

$$\begin{array}{ll}
CheckChansRenaming\ x\ y\ \Gamma \Leftrightarrow & (\Gamma.channels\ x == \Gamma.channels\ y) \\
CheckChansRenaming\ (x, xs)\ (y, ys)\ \Gamma \Leftrightarrow & (CheckChansRenaming\ x\ y\ \Gamma) \wedge \\
 & (CheckChansRenaming\ xs\ ys\ \Gamma)
\end{array}$$

2.4 CheckVarsRenaming

Function that verifies the variables of an action renaming. Each name within the second list has to have the same type of the corresponding variables of the first list passed as argument of the function.

$$CheckVarsRenaming : SeqName \nrightarrow SeqName \nrightarrow TEnv \nrightarrow Bool$$

$$\begin{array}{ll}
CheckVarsRenaming\ x\ y\ \Gamma \Leftrightarrow & (\Gamma.localVars\ x) == (\Gamma.localVars\ y) \\
CheckVarsRenaming\ (x, xs)\ (y, ys)\ \Gamma \Leftrightarrow & (CheckVarsRenaming\ x\ y\ \Gamma) \wedge \\
 & (CheckVarsRenaming\ xs\ ys\ \Gamma)
\end{array}$$

2.5 Compare

Function that compares the type of each element of a pair set with the type of the corresponding element of a type list.

$$Compare : \mathbb{P}(NAME \times T) \rightarrow SeqType \rightarrow Bool$$

$$Compare\ \{x \mapsto T\}\ U \Leftrightarrow T == U$$
$$Compare\ \{x \mapsto T, ds\}\ (U, ts) \Leftrightarrow T == U \wedge (Compare\ \ ds\ \ ts)$$

2.6 DeclareNewChans

Function that receives a set of channel declarations and an environment, and verifies if the channel declarations are new in the environment, or if they are redeclarations with same type. In these cases, the function returns *true*. Case the set has redeclarations of channels with different types, the function returns *false*.

$$DeclareNewChans : \mathbb{P}(NAME \times T) \nrightarrow TEnv \nrightarrow Bool$$

$$DeclareNewChans\ \ ds\ \Gamma \Leftrightarrow\ (DiffDecs\ \ ds) \wedge (NewChans\ \ ds\ \Gamma)$$

2.7 Decs

Function that receives a declaration (that can be simple or compound) and returns a pair set whose the first element is the name of a variable, constant or componente of a declared schema, and the second one is its type.

$$Decs : Dec \rightarrow \mathbb{P}(NAME \times T)$$

$$Decs(x : T) = \{x \mapsto T\}$$
$$Decs(x, xs : T) = \{x \mapsto T\} \cup Decs(xs : T)$$
$$Decs(xs;\ ys) = Decs(xs) \cup Decs(ys)$$

2.8 DecsParDec

Function that receives a parametrised command declaration, and returns a pair set whose the first element is the name of a declared variable, and the second one is the corresponding variable type.

$$DecsParDec : ParDec \rightarrow \mathbb{P}(NAME \times T)$$

$$DecsParDec\ (sym\ \ d) = Decs\ \ d$$
$$DecsParDec\ (d;\ \ ds) = (DecsParDec\ \ d) \cup (DecsParDec\ \ ds)$$
$$\mathbf{where} :\ \mathbf{sym} =\ \mathbf{val}\ \mid\ \mathbf{res}\ \mid\ \mathbf{valres}$$

2.9 Defs

Function that extracts the global definitions (channels, channel sets, processes, constants and types) of a CircusParagraph. It returns an environment with the extracted definitions.

$$Defs : CPar \nrightarrow TEnv \nrightarrow TEnv$$

$$
\begin{array}{ll}
Defs(\mathbf{channel}\ cd)\ \Gamma & = Chan\ cd \\
Defs(\mathbf{chanset}\ n == cs)\ \Gamma & = (chansets = \{n \mapsto (FindCCSE\ \ cs\ \Gamma)\}) \\
Defs(\mathbf{process}\ n \mathrel{\widehat=} p)\ \Gamma & = (channels = (FindImplicitChans\ \ p\ \Gamma), \\
 & \quad definedProcs = \{n\}, \\
 & \quad usedChansProc = (FindCP\ \ p\ \Gamma)) \\
Defs(\mathbf{process}\ n[ln] \mathrel{\widehat=} p)\ \Gamma & = (channels = (FindImplicitChans\ \ p\ \Gamma), \\
 & \quad definedProcs = \{n\}, \\
 & \quad usedChansProc = (FindCP\ \ p\ \Gamma)) \\
Defs(\mathtt{ZED}\ [ln]\ \mathtt{END})\ \Gamma & = (zDefs = ExtractGivenTypes\ ln) \\
Defs(\mathtt{AX}\ d \mid p\ \mathtt{END})\ \Gamma & = (zDefs = Decs\ \ d) \\
Defs(\mathtt{SCH}\ n\ \ d \mid p\ \mathtt{END})\ \Gamma & = (zDefs = \{n \mapsto \mathbb{P}[(Decs\ \ d)]\}) \\
Defs(\mathtt{GENAX}\ [ln]\ \ d \mid p\ \mathtt{END})\ \Gamma & = (zDefs = Decs\ \ d \cup ExtractGenTypes\ ln) \\
Defs(\mathtt{GENSCH}\ n\ [ln]\ \ d \mid p\ \mathtt{END})\ \Gamma & = (zDefs = \{n \mapsto \mathbb{P}[(Decs\ \ d)]\} \cup ExtractGenTypes\ ln) \\
Defs(\mathtt{ZED}\ n == e\ \mathtt{END})\ \Gamma & = (zDefs = \{n \mapsto e\}) \\
Defs(\mathtt{ZED}\ n\ [ln] == e\ \mathtt{END})\ \Gamma & = (zDefs = \{n \mapsto e\} \cup ExtractGenTypes\ ln) \\
Defs(\mathtt{ZED}\ fts\ \mathtt{END})\ \Gamma & = (zDefs = ExtractFreeTypes\ fts) \\
Defs(\mathtt{ZED}\ \vdash ?\ p\ \mathtt{END})\ \Gamma & = \Gamma_\emptyset \\
Defs(\mathtt{ZED}\ [ln]\ \vdash ?\ p\ \mathtt{END})\ \Gamma & = \Gamma_\emptyset \\
Defs(\mathtt{ZED}\ ot\ \mathtt{END})\ \Gamma & = \Gamma_\emptyset
\end{array}
$$

2.10 DefsAction

Function that verifies if an action is parametrised, and updates the environment with information of the parametrised action.

$$DefsAction : NAME \rightarrow\!\!\!\!\!\!+\ ParAction \rightarrow\!\!\!\!\!\!+\ TEnv$$

$$
\begin{array}{ll}
DefsAction\ \ n\ \ (d \bullet a) & = (parActions = \{n \mapsto (Decs\ \ d)\}) \\
DefsAction\ \ n\ \ a & = \Gamma_\emptyset
\end{array}
$$

2.11 DefsP

Function that extracts all definitions (of actions, name sets, constants and local types) of a process paragraph (PParagraph). It returns an environment with the extracted definitions.

$$DefsP : PPar \rightarrow\!\!\!\!\!\!+\ TEnv \rightarrow\!\!\!\!\!\!+\ TEnv$$

$$
\begin{array}{ll}
DefsP(n \mathrel{\widehat=} a)\ \Gamma & = (definedActs = \{n\}) \\
DefsP(\mathbf{nameset}\ n == ns)\ \Gamma & = (namesets = \{n\}) \\
DefsP(\mathtt{ZED}\ [ln]\ \mathtt{END})\ \Gamma & = (localZDefs = (ExtractGivenTypes\ ln)) \\
DefsP(\mathtt{AX}\ d \mid p\ \mathtt{END})\ \Gamma & = (localZDefs = (Decs\ \ d)) \\
DefsP(\mathtt{SCH}\ n\ \ st\ \mathtt{END})\ \Gamma & = (DefsPSchema\ \ n\ \ st\ \Gamma) \\
DefsP(\mathtt{GENAX}\ [ln]\ \ d \mid p\ \mathtt{END})\ \Gamma & = (localZDefs = (Decs\ \ d \cup ExtractGenTypes\ ln)) \\
DefsP(\mathtt{GENSCH}\ n\ [ln]\ \ st\ \mathtt{END})\ \Gamma & = (localZDefs = ExtractGenTypes\ ln) \oplus \\
 & \quad (DefsPSchema\ \ n\ \ st\ \Gamma) \\
DefsP(\mathtt{ZED}\ n == e\ \mathtt{END})\ \Gamma & = (localZDefs = \{n \mapsto e\}) \\
DefsP(\mathtt{ZED}\ n\ [ln] == e\ \mathtt{END})\ \Gamma & = (localZDefs = (\{n \mapsto e\} \cup ExtractGenTypes\ ln)) \\
DefsP(\mathtt{ZED}\ fts\ \mathtt{END})\ \Gamma & = (localZDefs = (ExtractFreeTypes\ fts)) \\
DefsP(\mathtt{ZED}\ \vdash ?\ p\ \mathtt{END})\ \Gamma & = \Gamma_\emptyset \\
DefsP(\mathtt{ZED}\ [ln]\ \vdash ?\ p\ \mathtt{END})\ \Gamma & = \Gamma_\emptyset \\
DefsP(\mathtt{ZED}\ ot\ \mathtt{END})\ \Gamma & = \Gamma_\emptyset
\end{array}
$$

2.12 DefsPL

Function that extracts all definitions (of action, name sets, constants and local types) of a list of process paragraph (PParagraph). It returns an environment with the extracted definitions.

$$
\begin{aligned}
&\mathit{DefsPL} : \mathit{PParList} \nrightarrow \mathit{TEnv} \nrightarrow \mathit{TEnv}\\
&\mathit{DefsPL}\ \emptyset\ \Gamma = \Gamma_\emptyset\\
&\mathit{DefsPL}\ (p\ \mathit{pl})\ \Gamma = (\mathit{DefsP}\ p\ \Gamma) \oplus (\mathit{DefsPL}\ \mathit{pl}\ \Gamma)
\end{aligned}
$$

2.13 DefsProc

Function that extracts all definitions of parametrised and indexed processes. It returns an environment with the extracted definitions.

$$
\begin{aligned}
&\mathit{DefsProc} : \mathit{NAME} \nrightarrow \mathit{ProcDef} \nrightarrow \mathit{TEnv}\\
&\mathit{DefsProc}\ n\ (d \bullet p) = (\mathit{parProcesses} = \{n \mapsto \mathit{Decs}\ d\})\\
&\mathit{DefsProc}\ n\ (d \odot p) = (\mathit{indexProcesses} = \{n \mapsto \mathit{Decs}\ d\})\\
&\mathit{DefsProc}\ n\ p \qquad\quad = \Gamma_\emptyset
\end{aligned}
$$

2.14 DefsPSchema

Function that extracts all definitions (of action or local types) of a schema. It returns an environment with the extracted definitions.

$$
\begin{aligned}
&\mathit{DefsPSchema} : \mathit{NAME} \nrightarrow \mathit{SchemaText} \nrightarrow \mathit{TEnv} \nrightarrow \mathit{TEnv}\\
&\mathit{DefsPSchema}\ n\ (d \mid p)\ \Gamma = \mathbf{if}\ (\mathit{RefState}\ d\ \Gamma)\\
&\qquad\qquad \mathbf{then}\ (\mathit{actions} = \{n\},\\
&\qquad\qquad\qquad \mathit{localZDefs} = \{n \mapsto \mathbb{P}[(\mathit{Decs}\ d)]\})\\
&\qquad\qquad \mathbf{else}\ (\mathit{localZDefs} = \{n \mapsto \mathbb{P}[(\mathit{Decs}\ d)]\})
\end{aligned}
$$

2.15 DefsState

Function that extracts all state definitions of a schema passed as argument. It returns an environment that maps each extracted state component to its type.

$$
\begin{aligned}
&\mathit{DefsState} : \mathit{ZPar} \nrightarrow \mathbb{P}(\mathit{NAME} \times T) \nrightarrow \mathit{TEnv}\\
&\mathit{DefsState}(\mathtt{SCH}\ n\ \mathit{st}\ \mathtt{END})\ d \quad = (\mathit{localVars} = d,\\
&\qquad\qquad \mathit{localZDefs} = \{n \mapsto \mathbb{P}[d]\},\\
&\qquad\qquad \mathit{state} = n)\\
&\mathit{DefsState}(\mathtt{ZED}\ n\ ==\ e\ \mathtt{END})\ d = (\mathit{localVars} = d,\\
&\qquad\qquad \mathit{localZDefs} = \{n \mapsto \mathbb{P}[d]\},\\
&\qquad\qquad \mathit{state} = n)
\end{aligned}
$$

2.16 DiffDecs

Function that verifies if the declarations of a set passed as argument are distincts and if there are not redeclarations.

$$
\begin{aligned}
&\mathit{DiffDecs} : \mathbb{P}(\mathit{NAME} \times T) \rightarrow \mathit{Bool}\\
&\mathit{DiffDecs}\ \mathit{ds} \Leftrightarrow \sharp(\mathrm{dom}\ \mathit{ds}) == \sharp \mathit{ds}
\end{aligned}
$$

2.17 Extract

Function that determines the types of the input variables of a communication. It receives as argument the list of communication parameters (CParameter) and the corresponding channel type.

$$Extract : CParList \rightarrow T \rightarrow \mathbb{P}(NAME \times T)$$

$$\begin{array}{l} Extract\ cp\ \ T = (ExtractVarsCP\ \ cp\ \ T) \\ Extract\ (cp\ \ cpl)\ \ T = (Extract\ \ cp\ \ (GetTypeHead\ \ T)) \cup (Extract\ \ cpl\ \ (GetTypeTail\ \ T)) \end{array}$$

2.18 ExtractAct

Function that extracts the action definitions of a process paragraphs. It returns an environment with all extracted action definitions.

$$ExtractAct : PPar \nrightarrow TEnv$$

$$\begin{array}{l} ExtractAct\ \ (n \mathrel{\widehat=} a) = (actions = \{n\}) \oplus (DefsAction\ \ n\ \ a) \\ ExtractAct\ \ pp\ \ = \Gamma_{\emptyset} \end{array}$$

2.19 ExtractActDefs

Function that extracts the action definition of a list of process paragraphs. It returns an environment with all extracted action definitions.

$$ExtractActDefs : PParList \nrightarrow TEnv$$

$$\begin{array}{l} ExtractActDefs\ \ p\ \ = ExtractAct\ \ p \\ ExtractActDefs\ \ (p\ \ pl)\ \ = (ExtractActDefs\ \ p) \oplus (ExtractActDefs\ \ pl) \end{array}$$

2.20 ExtractFreeTypes

Function that receives a list of free types, and returns a set with these free types.

$$ExtractFreeTypes : FreeTypesList \rightarrow \mathbb{P}(NAME \times T)$$

$$\begin{array}{l} ExtractFreeTypes(ft\ \&\ fts) = ExtractFreeTypes\ \ ft \cup ExtractFreeTypes\ \ fts \\ ExtractFreeTypes(n ::= lb) = \{n \mapsto \mathbb{P}(\texttt{GIVEN}\ n)\} \cup (ExtractFT\ \ n\ \ lb) \end{array}$$

2.21 ExtractFT

Function that receives the name of a free type and a list of values of this types, and returns a set that maps each values to the free type.

$$ExtractFT : NAME \rightarrow BranchList \rightarrow \mathbb{P}(NAME \times T)$$

$$\begin{array}{ll} ExtractFT\ \ f\ \ h & = \{h \mapsto \texttt{GIVEN}\ f\} \\ ExtractFT\ \ f\ \ (h \ll e \gg) & = \{h \mapsto \mathbb{P}(e \times \texttt{GIVEN}\ f)\} \\ ExtractFT\ \ f\ \ (b \mid bl) & = (ExtractFT\ \ f\ \ b) \cup (ExtractFT\ \ f\ \ bl) \end{array}$$

2.22 ExtractGenTypes

Function that receives a lista of generic types, and returns a set that maps each generic type to its corresponding type (like defined in the Z type system).

$$ExtractGenTypes : SeqName \rightarrow \mathbb{P}(NAME \times T)$$

$$ExtractGenTypes\ n = \{n \mapsto \mathbb{P}(\mathtt{GENTYPE}\ n)\}$$
$$ExtractGenTypes(n, ln) = (ExtractGenTypes\ n) \cup (ExtractGenTypes\ ln)$$

2.23 ExtractGivenTypes

Function that receives a list of given types, and returns a set that maps each given type to its corresponding type (like defined in the Z type system).

$$ExtractGivenTypes : SeqName \rightarrow \mathbb{P}(NAME \times T)$$

$$ExtractGivenTypes\ n = \{n \mapsto \mathbb{P}(\mathtt{GIVEN}\ n)\}$$
$$ExtractGivenTypes(n, ln) = (ExtractGivenTypes\ n) \cup (ExtractGivenTypes\ ln)$$

2.24 ExtractProc

Function that extracts the process definitions of a *Circus* paragraphs. It returns an environment with all extracted process definitions.

$$ExtractProc : CPar \twoheadrightarrow TEnv$$

$$\begin{aligned} ExtractProc\ (\mathbf{process}\ n \mathrel{\widehat=} p) &= (processes = \{n\}) \oplus (DefsProc\ n\ p) \\ ExtractProc\ (\mathbf{process}\ n[ln] \mathrel{\widehat=} p) &= (processes = \{n\}, \\ &\quad\ genProcesses = \{n \mapsto ExtractGenTypes\ ln\}) \oplus \\ &\quad \oplus (DefsProc\ n\ p) \\ ExtractProc\ cp &= \Gamma_\emptyset \end{aligned}$$

2.25 ExtractProcDefs

Function that extracted the process definitions of a list of *Circus* paragraphs. It returns an environment with all extracted process definitions.

$$ExtractProcDefs : CParList \twoheadrightarrow TEnv$$

$$ExtractProcDefs\ cp = ExtractProc\ cp$$
$$ExtractProcDefs\ (cp\ cpl) = (ExtractProcDefs\ cp) \oplus (ExtractProcDefs\ cpl)$$

2.26 ExtractVars

Function that receives the name and type of a variable, and returns a set that maps each variable and its variations (x', $x?$, $x!$) to their corresponding type.

$$ExtractVars : NAME \rightarrow T \rightarrow \mathbb{P}(NAME \times T)$$

$$ExtractVars\ x\ T = \{x \mapsto T, x' \mapsto T, x? \mapsto T, x! \mapsto T\}$$

2.27 ExtractVarsCP

Function that extracts the input variables of a CParameter. It returns an environment with the extracted variables.

$$ExtractVarsCP : CPar \nrightarrow T \nrightarrow TEnv$$

$$\begin{array}{l}
ExtractVarsCP\ (?x)\ T = (localVars = (ExtractVars\ x\ T)) \\
ExtractVarsCP\ (?x : p)\ T = (localVars = (ExtractVars\ x\ T)) \\
ExtractVarsCP\ (!e)\ T = \Gamma_{\emptyset} \\
ExtractVarsCP\ (.e)\ T = \Gamma_{\emptyset}
\end{array}$$

2.28 FindCA

Function that verifies the use of channels within an action, and returns a set with the names of used channels.

$$FindCA : ParAction \nrightarrow TEnv \nrightarrow \mathbb{P}\, NAME$$

$$\begin{array}{l}
FindCA(d \bullet a)\ \Gamma = FindCA\ a\ \Gamma \\
FindCA(\mu\ \ n \bullet a)\ \Gamma = FindCA\ a\ \Gamma \\
FindCA\ sc\ \Gamma = \emptyset \\
FindCA\ n\ \Gamma = \emptyset \\
FindCA(a[ln_1 := ln_2])\ \Gamma = FindCA\ a\ \Gamma \\
FindCA\ base\ \Gamma = \emptyset \\
FindCA(c \rightarrow a)\ \Gamma = (FindCComm\ c) \cup (FindCA\ a\ \Gamma) \\
FindCA(p\ \&\ a)\ \Gamma = FindCA\ a\ \Gamma \\
FindCA(a_1\ sym_0\ a_2)\ \Gamma = (FindCA\ a_1\ \Gamma) \cup (FindCA\ a_2\ \Gamma) \\
FindCA(a_1\ \ [\![ns_1 \mid cs \mid ns_2]\!]\ \ a_2)\ \Gamma = (FindCA\ a_1\ \Gamma)\ \cup\ (FindCA\ a_2\ \Gamma) \cup \\
\qquad\qquad\qquad\qquad\qquad\qquad (FindCCSE\ cs\ \Gamma)
\end{array}$$

$$\begin{array}{l}
FindCA(a_1\ sym_1\ a_2)\ \Gamma = (FindCA\ a_1\ \Gamma)\ \cup\ (FindCA\ a_2\ \Gamma) \cup \\
\qquad\qquad\qquad\qquad (FindCCSE\ cs_1\ \Gamma) \cup (FindCCSE\ cs_2\ \Gamma) \\
FindCA(a \setminus cs)\ \Gamma = (FindCA\ a\ \Gamma) \cup (FindCCSE\ cs\ \Gamma) \\
FindCA(a(le))\ \Gamma = FindCA\ a\ \Gamma \\
FindCA(n(le))\ \Gamma = \emptyset \\
FindCA(left_0\ \ d \bullet a)\ \Gamma = FindCA\ a\ \Gamma \\
FindCA(|||\ d \bullet ||[ns]||a)\ \Gamma = FindCA\ a\ \Gamma \\
FindCA([\![\ cs\ ]\!]\ d \bullet [\![\ ns\ ]\!]\ a)\ \Gamma = (FindCA\ a\ \Gamma) \cup (FindCCSE\ cs\ \Gamma) \\
FindCA\ c\ \Gamma = FindCCommand\ c\ \Gamma \\
\textbf{where}:\ \ base = Skip\ \mid\ Stop\ \mid\ Chaos, \\
\qquad\qquad sym_0 = ;\ \ \mid \Box \mid \sqcap \mid\ \ ||[ns_1 \mid ns_2]||, \\
\qquad\qquad sym_1 =\ \ [\![cs_1 |\!| cs_2]\!]\ \mid\ [\![ns_1 \mid cs_1 |\!| cs_2 \mid ns_2]\!], \\
\qquad\qquad left_0 = \fatsemi \mid \Box \mid \sqcap
\end{array}$$

2.29 FindCComm

Function that verifies the use of channels within a communication, and returns a set with the names of used channels.

$FindCComm : Comm \rightarrow \mathbb{P}\, NAME$

$FindCComm(n) = \{n\}$
$FindCComm(n\ cp) = \{n\}$
$FindCComm(n\ [le]\ cp) = \{n\}$

2.30 FindCCommand

Function that verifies the use of channels within a command, and returns a set with the names of used channels.

$FindCCommand : ParCommand \nrightarrow TEnv \nrightarrow \mathbb{P}\, NAME$

$FindCCommand(ln : [p_1, p_2])\ \Gamma = \emptyset$
$FindCCommand(ln := le)\ \Gamma = \emptyset$
$FindCCommand(\mathbf{var}\, d \bullet a)\ \Gamma = FindCA\ a\ \Gamma$
$FindCCommand(\{p\})\ \Gamma = \emptyset$
$FindCCommand([p])\ \Gamma = \emptyset$
$FindCCommand(\mathbf{if}\ ga\ \mathbf{fi})\ \Gamma = FindCGA\ ga\ \Gamma$
$FindCCommand((d \bullet c))\ \Gamma = FindCCommand\ c\ \Gamma$
$FindCCommand((d \bullet c)(le))\ \Gamma = FindCCommand\ c\ \Gamma$

2.31 FindCCSE

Function that verifies the use of channels within aCSExpression. It returns a set with the names of referred channels with the CSExpression.

$FindCCSE : CSExp \nrightarrow TEnv \nrightarrow \mathbb{P}\, NAME$

$FindCCSE(\{\!|\ |\!\})\ \Gamma = \emptyset$
$FindCCSE(\{\!| ln |\!\})\ \Gamma = Set\ ln$
$FindCCSE\ n\ \Gamma = \Gamma.chansets\ n$
$FindCCSE(cs_1\ middle\ cs_2)\ \Gamma = (FindCCSE\ cs_1\ \Gamma) \cup (FindCCSE\ cs_2\ \Gamma)$
$\mathbf{where}:\ middle = \ \cup\ \mid\ \cap\ \mid \setminus$

2.32 FindCGA

Function that verifies the use of channels within a guarded action, and returns a set with the names of used channels.

$FindCGA : GAction \nrightarrow TEnv \nrightarrow \mathbb{P}\, NAME$

$FindCGA(p \rightarrow a)\ \Gamma = FindCA\ a\ \Gamma$
$FindCGA(ga \square lga)\ \Gamma = (FindCGA\ ga\ \Gamma) \cup (FindCGA\ lga\ \Gamma)$

2.33 FindCP

Function that verifies the use of channels within a process. It returns a set with the names of used channels within the process.

$FindCP : ProcDef \nrightarrow TEnv \nrightarrow \mathbb{P}\, NAME$

$FindCP(d \ \odot \ p) \ \Gamma = \mathrm{dom}(ImplicitChans \ d \ p \ \Gamma)$
$FindCP(d \ \bullet \ p) \ \Gamma = FindCP \ p \ \Gamma$
$FindCP(\mathbf{begin} \ pp_1 \ \mathbf{state} \ s \ pp_2 \bullet a) \ \Gamma = (FindCPP \ pp_1 \ \Gamma) \cup (FindCPP \ pp_2 \ \Gamma) \cup$
$\cup \ (FindCA \ a \ \Gamma)$
$FindCP \ n \ \Gamma = (\Gamma.usedChansProc \ n)$
$FindCP(p_1 \ sym_0 \ p_2) \ \Gamma = (FindCP \ p_1 \ \Gamma) \cup (FindCP \ p_2 \ \Gamma)$
$FindCP(p \setminus cs) \ \Gamma = (FindCP \ p \ \Gamma) \cup (FindCCSE \ cs \ \Gamma)$
$FindCP(p_1 \ \llbracket cs \rrbracket \ p_2) \ \Gamma = (FindCP \ p_1 \ \Gamma) \cup (FindCP \ p_2 \ \Gamma) \cup (FindCCSE \ cs \ \Gamma)$
$FindCP((d \bullet p)(le)) \ \Gamma = FindCP \ p \ \Gamma$
$FindCP(n(le)) \ \Gamma = (\Gamma.usedChansProc \ n)$
$FindCP(n[le_1](le_2)) \ \Gamma = (\Gamma.usedChansProc \ n)$
$FindCP((d \odot p)\lfloor le \rfloor)) \ \Gamma = (FindCP \ (d \odot p) \ \Gamma)$
$FindCP(n\lfloor le \rfloor)) \ \Gamma = (\Gamma.usedChansProc \ n)$
$FindCP(n[le_1]\lfloor le_2 \rfloor)) \ \Gamma = (\Gamma.usedChansProc \ n)$
$FindCP(p \ [ln_1 := ln_2]) \ \Gamma = (FindCP \ p \ \Gamma) \cup (Set \ ln_1) \cup (Set \ ln_2)$
$FindCP(n[le]) \ \Gamma = (\Gamma.usedChansProc \ n)$
$FindCP(left \ d \ \bullet \ p) \ \Gamma = FindCP \ p \ \Gamma$
$FindCP(\llbracket cs \rrbracket \ d \ \bullet \ p) \ \Gamma = (FindCP \ p \ \Gamma) \cup (FindCCSE \ cs \ \Gamma)$
where : $sym_0 = \ ; \ \mid \Box \mid \sqcap \mid \interleave,$
$left = \fatsemi \mid \Box \mid \sqcap \mid \interleave$

2.34 FindCPP

Function that verifies the use of channels within a list of process paragraph, and returns a set with the names of used channels.

$FindCPP : PParList \rightarrow TEnv \rightarrow \mathbb{P}\, NAME$

$FindCPP \ par \ \Gamma = \emptyset$
$FindCPP(\mathbf{nameset} \ n == ns) \ \Gamma = \emptyset$
$FindCPP(n \mathrel{\widehat=} a) \ \Gamma = (FindCA \ a \ \Gamma)$
$FindCPP(p \ pl) \ \Gamma = (FindCPP \ p \ \Gamma) \cup (FindCPP \ pl \ \Gamma)$

2.35 FindImplicitChans

Function that extracts all definitions of implicit channels of a process. It returns a pair set whose the first element of each pair has the name of a channel, and the second one has the channels type.

$FindImplicitChans : ProcDef \rightarrow TEnv \rightarrow \mathbb{P}(NAME \rightarrow T)$

$FindImplicitChans(d \odot p) \ \Gamma = ImplicitChans \ d \ p \ \Gamma$
$FindImplicitChans(d \bullet p) \ \Gamma = FindImplicitChans \ p \ \Gamma$
$FindImplicitChans(\mathbf{begin} \ pp_1 \ \mathbf{state} \ s \ pp_2 \bullet a) \ \Gamma = \emptyset$
$FindImplicitChans(n) \ \Gamma = \emptyset$
$FindImplicitChans(p_1 \ sym_0 \ p_2) \ \Gamma = (FindImplicitChans \ p_1 \ \Gamma) \cup$
$\cup \ (FindImplicitChans \ p_2 \ \Gamma)$
$FindImplicitChans(p \setminus cs) \ \Gamma = FindImplicitChans \ p \ \Gamma$
$FindImplicitChans((d \bullet p)(le)) \ \Gamma = FindImplicitChans \ p \ \Gamma$

$FindImplicitChans(n(le))\ \Gamma = \emptyset$
$FindImplicitChans(n[le_1](le_2))\ \Gamma = \emptyset$
$FindImplicitChans((d \odot p)\lfloor le \rfloor))\ \Gamma = (FindImplicitChans\ (d \odot p)\ \Gamma)$
$FindImplicitChans(n\lfloor le \rfloor))\ \Gamma = \emptyset$
$FindImplicitChans(n[le_1]\lfloor le_2 \rfloor))\ \Gamma = \emptyset$
$FindImplicitChans(p[ln_1 := ln_2])\ \Gamma = chansProc\ \cup$
$\qquad \cup\ (ImplicitRenameChans\ chansProc\ ln_1\ ln_2\ \Gamma)$
$FindImplicitChans(n[le])\ \Gamma = \emptyset$
$FindImplicitChans(left\ d\ \bullet\ p)\ \Gamma = FindImplicitChans\ p\ \Gamma$
where : $sym_0 = ;\ \mid \Box \mid \sqcap \mid ||| \mid\ |[cs]|,$
$left = ⨟ \mid \Box \mid \sqcap \mid ||| \mid\ |[\ cs]|,$
$chansProc = (FindImplicitChans\ p\ \Gamma)$

2.36 FiniteDecs

Function that receives a declaration and verifies if the variables of this declaration have as type finite sets.

$FiniteDecs : Dec \nrightarrow Bool$

$FiniteDecs\ (x : t) \Leftrightarrow (IsFinite\ t)$
$FiniteDecs\ (x, xs : t) \Leftrightarrow (IsFinite\ t)$
$FiniteDecs\ (xs;\ ys) \Leftrightarrow (FiniteDecs\ xs) \wedge (FiniteDecs\ ys)$

2.37 GenChans

Function that receives a list of channel names and a set of generic parameters, and returns a ser that maps each channel to the set of generic parameters.

$GenChans : SeqName \rightarrow \mathbb{P}(NAME \times T) \rightarrow \mathbb{P}(NAME \times \mathbb{P}(NAME \times T))$

$GenChans\ n\ gl = \{n \mapsto gl\}$
$GenChans\ (n, ln)\ gl = (GenChans\ n\ gl) \cup (GenChans\ ln\ gl)$

2.38 GetTypeHead

Function that extracts the first simple type of a tuple of types.

$GetTypeHead : T \rightarrow T$

$GetTypeHead\ t = t$
$GetTypeHead\ (t \times tl) = t$

2.39 GetTypeTail

Function that returns a type without the first simple type of a typle of types.

$GetTypeTail : T \rightarrow T$

$GetTypeTail\ (t \times tl) = tl$

2.40 ImplicitChans

Function that extracts the implicit channels of a indexed process. It returns a set that maps each extracted implicit channels to its corresponding type.

$$
\begin{aligned}
&\mathit{ImplicitChans} : \mathit{Dec} \rightarrow\!\!\!\!\!+ \mathit{ProcDef} \rightarrow\!\!\!\!\!+ \mathit{TEnv} \rightarrow\!\!\!\!\!+ \mathbb{P}(\mathit{NAME} \times T)\\
&\mathit{ImplicitChans}(d, p, \Gamma) = \{N : \mathit{NAME};\ n : \mathit{NAME} \mid N \in (\mathit{FindCP}\ p\ \Gamma) \wedge\\
&\qquad\qquad n = \mathit{MakeName}(N, d) \bullet n \mapsto \mathit{MakeType}(\Gamma.\mathit{channels}\ N, d)\}
\end{aligned}
$$

2.41 ImplicitRenameChans

Function that extracts the channels (of the list of channel types on the right side of a process renaming) that renames the implicit channels of a process. It receives as argument a set of implicit channels of a process, the list of names of channels that will be renamed, the list of names of channels that will be replaced, and the environment.

$$
\begin{aligned}
&\mathit{ImplicitRenameChans} : \mathbb{P}(\mathit{NAME} \times T) \rightarrow\!\!\!\!\!+ \mathit{SeqName} \rightarrow\!\!\!\!\!+ \mathit{SeqName} \rightarrow\!\!\!\!\!+ \mathit{TEnv} \rightarrow\!\!\!\!\!+ \mathbb{P}(\mathit{NAME} \times T)\\
&\mathit{ImplicitRenameChans}\ \mathit{set}\ n_1\ n_2\ \Gamma = \textbf{if}\ (n_1 \in \mathit{set})\ \textbf{then}\ \{n_2 \mapsto \Gamma.\mathit{channels}\ n_2\}\\
&\qquad \textbf{else}\ \emptyset\\
&\mathit{ImplicitRenameChans}\ \mathit{set}\ (n_1, \mathit{ln}_1)\ (n_2, \mathit{ln}_2)\ \Gamma = (\mathit{ImplicitRenameChans}\ \mathit{set}\ n_1\ n_2\ \Gamma) \cup\\
&\qquad (\mathit{ImplicitRenameChans}\ \mathit{set}\ \mathit{ln}_1\ \mathit{ln}_2\ \Gamma)
\end{aligned}
$$

2.42 InstantiateTypesGenChan

Function that updates the environment replacing the generic types of a channel by the types passed as arguments. It receives the set of generic types of a channel, the list of new types, the name of generic channel, and the type of this channel.

$$
\begin{aligned}
&\mathit{InstantiateTypesGenChan} : \mathbb{P}(\mathit{NAME} \times T) \rightarrow\!\!\!\!\!+ \mathit{SeqType} \rightarrow\!\!\!\!\!+ \mathit{NAME} \rightarrow\!\!\!\!\!+ T \rightarrow\!\!\!\!\!+ \mathit{TEnv}\\
&\mathit{InstantiateTypesGenChan}\{n : k\}\ t\ c\ T =\\
&\qquad (\mathit{channels} = \{c \mapsto (\mathit{ReplaceChanType}\ n\ t\ T)\})\\
&\mathit{InstantiateTypesGenChan}\ \{n : k, \mathit{ns}\}\ (t, \mathit{ts})\ c\ T =\\
&\qquad (\mathit{InstantiateTypesGenChan}\ \{n : k\}\ t\ c\ T) \oplus\\
&\qquad (\mathit{InstantiateTypesGenChan}\ \{\mathit{ns}\}\ \mathit{ts}\ c\ T)
\end{aligned}
$$

2.43 Into

Function that verifies if the elements of a list are into a set passed as argument.

$$
\begin{aligned}
&\mathit{Into} : \mathit{SeqName} \rightarrow \mathbb{P}\ \mathit{NAME} \rightarrow \mathit{Bool}\\
&\mathit{Into}\ n\ \mathit{set} \Leftrightarrow (n \in \mathit{set})\\
&\mathit{Into}\ (n, \mathit{ln})\ \mathit{set} \Leftrightarrow (n \in \mathit{set}) \wedge (\mathit{Into}\ \mathit{ln}\ \mathit{set})
\end{aligned}
$$

2.44 IsFinite

Function that verifies if a type is finite. As a type is a set of possible values that a variable can have, the function verifies if this type é a finite set..

$$
\begin{aligned}
&\mathit{IsFinite} : T \rightarrow \mathit{Bool}\\
&\mathit{IsFinite}\ t \Leftrightarrow (t = \emptyset) \vee (\exists\, n : \mathbb{N} \bullet f : \{k \in \mathbb{N};\ k \leq n\} \rightarrow t)
\end{aligned}
$$

2.45 IsNormalProc

Function that verifies se a name of process passed as argument is not a name of parametrised, indexed or generic process.

$$IsNormalProc : NAME \nrightarrow TEnv \nrightarrow Bool$$

$$\begin{aligned} IsNormalProc\ n\ \Gamma \Leftrightarrow & (n \notin \mathrm{dom}(\Gamma.parProcesses)) \wedge (n \notin \mathrm{dom}(indexProcesses)) \wedge \\ & \wedge (n \notin \mathrm{dom}(genProcesses)) \end{aligned}$$

2.46 MakeName

Function that construct the name of an implicit channel from the name of original channel and declared variables.

$$MakeName : NAME \rightarrow Dec \rightarrow NAME$$

$$\begin{aligned} MakeName(N, i : t) &= N\_i \\ MakeName(N, (i, is : t)) &= MakeName(N\_i, is : t) \\ MakeName(N, (d;\ ds)) &= MakeName(MakeName(N, d), ds) \end{aligned}$$

2.47 MakeType

Function that constructs and returns the type, possible compound, of an implicit channel from the originla type of the channel and the types of declared variables.

$$MakeType : T \rightarrow Dec \rightarrow T$$

$$MakeType\ t\ d = (MakeTypeChan\ d) \times t$$

2.48 MakeTypeChan

Function that constructs a compound type from the types of a declaration.

$$MakeTypeChan : Dec \rightarrow T$$

$$\begin{aligned} MakeTypeChan\ (x : t) &= t \\ MakeTypeChan\ (x, xs : t) &= t \times (MakeTypeChan\ (xs : t)) \\ MakeTypeChan\ (d;\ ds) &= (MakeTypeChan\ d) \times (MakeTypeChan\ ds) \end{aligned}$$

2.49 NewChans

Function that verifies if a set of channel declarations passed as argument declares new channels, which do not exist in the environment; or if they already exist, they have the same types.

$$NewChans : \mathbb{P}(NAME \times T) \nrightarrow TEnv \nrightarrow Bool$$

$$\begin{aligned} NewChans\ \{x : t\}\ \Gamma \Leftrightarrow & x \notin \mathrm{dom}(\Gamma.defNames) \vee \\ & \vee (x \in \mathrm{dom}(\Gamma.channels) \wedge t == \Gamma.channels\ x) \\ NewChans\ \{x : t, xs\}\ \Gamma \Leftrightarrow & (NewChans\ \{x : t\}\ \Gamma) \wedge (NewChans\ xs\ \Gamma) \end{aligned}$$

2.50 NewDefs

Function that verifies if the names defined in a Z paragraph are not into a set passed as argument. It returns *true* if the defined names are new; otherwise, returns *false*.

$$
\begin{array}{ll}
\mathit{NewDefs} : \mathit{ZPar} \nrightarrow \mathbb{P}\,\mathit{NAME} \nrightarrow \mathit{Bool} & \\
\mathit{NewDefs}(\texttt{ZED}\ [\mathit{ln}]\ \texttt{END})\ \mathit{ns} & \Leftrightarrow (\mathrm{dom}(\mathit{ExtractGivenTypes}\ \mathit{ln}) \cap \mathit{ns}) = \emptyset \\
\mathit{NewDefs}(\texttt{AX}\ d \mid p\ \texttt{END})\ \mathit{ns} & \Leftrightarrow (\mathrm{dom}(\mathit{Decs}\ d) \cap \mathit{ns}) = \emptyset \\
\mathit{NewDefs}(\texttt{SCH}\ n\ \mathit{st}\ \texttt{END})\ \mathit{ns} & \Leftrightarrow n \notin \mathit{ns} \\
\mathit{NewDefs}(\texttt{GENAX}\ [\mathit{ln}]\ d \mid p\ \texttt{END})\ \mathit{ns} & \Leftrightarrow (\mathrm{dom}(\mathit{Decs}\ d) \cap \mathit{ns}) = \emptyset \\
\mathit{NewDefs}(\texttt{GENSCH}\ n\ [\mathit{ln}]\ \mathit{st}\ \texttt{END})\ \mathit{ns} & \Leftrightarrow n \notin \mathit{ns} \\
\mathit{NewDefs}(\texttt{ZED}\ n == e\ \texttt{END})\ \mathit{ns} & \Leftrightarrow n \notin \mathit{ns} \\
\mathit{NewDefs}(\texttt{ZED}\ n\ [\mathit{ln}] == e\ \texttt{END})\ \mathit{ns} & \Leftrightarrow n \notin \mathit{ns} \\
\mathit{NewDefs}(\texttt{ZED}\ \mathit{fts}\ \texttt{END})\ \mathit{ns} & \Leftrightarrow (\mathrm{dom}(\mathit{ExtractFreeTypes}\ \mathit{fts}) \cap \mathit{ns}) = \emptyset \\
\mathit{NewDefs}(\texttt{ZED}\ \vdash ?\ p\ \texttt{END})\ \mathit{ns} & \Leftrightarrow \mathit{true} \\
\mathit{NewDefs}(\texttt{ZED}\ [\mathit{ln}]\ \vdash ?\ p\ \texttt{END})\ \mathit{ns} & \Leftrightarrow \mathit{true} \\
\mathit{NewDefs}(\texttt{ZED}\ \mathit{ot}\ \texttt{END})\ \mathit{ns} & \Leftrightarrow \mathit{true}
\end{array}
$$

2.51 NoConflicts

Function that verifies if there are redeclarations of names with different types.

$$
\begin{array}{l}
\mathit{NoConflicts} : \mathbb{P}(\mathit{NAME} \times T) \rightarrow \mathit{Bool} \\
\mathit{NoConflicts}\ \mathit{set} \Leftrightarrow \sharp(\mathrm{dom}\ \ \mathit{set}) == \sharp \mathit{set}
\end{array}
$$

2.52 NoRep

Function that verifies if there are not repeated name within a list of names. For this, it transforms the list into a set, and verifies if the number of elements of the generated set is equals than the number of elements of the list.

$$
\begin{array}{l}
\mathit{NoRep} : \mathit{SeqName} \rightarrow \mathit{Bool} \\
\mathit{NoRep}\ \mathit{ln} \Leftrightarrow \sharp(\mathit{Set}\ \ \mathit{ln}) == \sharp \mathit{ln}
\end{array}
$$

2.53 NotInto

Function that verifies if none of the names of a list passed as argument is within a set also passed as argument.

$$
\begin{array}{l}
\mathit{NotInto} : \mathit{SeqName} \rightarrow \mathbb{P}\,\mathit{NAME} \rightarrow \mathit{Bool} \\
\mathit{NotInto}\ \ n\ \ \mathit{set} \Leftrightarrow (n \notin \mathit{set}) \\
\mathit{NotInto}\ \ (n, \mathit{ln})\ \ \mathit{set} \Leftrightarrow (n \notin \mathit{set}) \wedge (\mathit{NotInto}\ \ \mathit{ln}\ \ \mathit{set})
\end{array}
$$

2.54 NotRedeclare

Function that verifies if there are not redeclarations into two sets passed as arguments. It returns *true* if there are not redeclarations, and *false* otherwise.

$$
\begin{array}{l}
\mathit{NotRedeclare} : \mathbb{P}\,\mathit{NAME} \rightarrow \mathbb{P}\,\mathit{NAME} \rightarrow \mathit{Bool} \\
\mathit{NotRedeclare}\ \ s_1\ \ s_2 \Leftrightarrow \sharp(s_1 \cup s_2) == (\sharp s_1 + \sharp s_2)
\end{array}
$$

2.55 NumTypes

Function that returns the number of simple types within a cartesian product type.

$$\begin{array}{l} \mathit{NumTypes} : T \rightarrow \mathit{Int} \rightarrow \mathit{Int} \\ \mathit{NumTypes}\ t\ n = n + 1 \\ \mathit{NumTypes}\ (t \times tl)\ n = (\mathit{NumTypes}\ tl\ (n + 1)) \end{array}$$

2.56 NumTypesChan

Function that returns the number of simple types within a compound types.

$$\begin{array}{l} \mathit{NumTypesChan} : T \rightarrow \mathit{Int} \\ \mathit{NumTypesChan}\ t = \mathit{NumTypes}\ t\ 0 \end{array}$$

2.57 RefGenParam

Function that receives a expression that represents a type and a set of names of generic parameters, and verifies if some generic parameter is being referred by the expression.

$$\begin{array}{l} \mathit{RefGenParam} : \mathit{Expr} \rightarrow \mathbb{P}\,\mathit{NAME} \rightarrow \mathit{Bool} \\ \mathit{RefGenParam}\ t\ ns \Leftrightarrow t \in ns \\ \mathit{RefGenParam}\ (\mathbb{P}\,t)\ ns \Leftrightarrow (\mathit{RefGenParam}\ t\ ns) \\ \mathit{RefGenParam}\ (t \times ts)\ ns \Leftrightarrow (\mathit{RefGenParam}\ t\ ns) \vee (\mathit{RefGenParam}\ ts\ ns) \\ \mathit{RefGenParam}\ e\ ns \Leftrightarrow \mathit{false} \end{array}$$

2.58 RefState

Function that receives the declarative part of a schema and the environment, and verifies if there is some reference to the process state into the declarative part.

$$\begin{array}{l} \mathit{RefState} : \mathit{Dec} \rightarrow\!\!\!\!\!+ \mathit{TEnv} \rightarrow\!\!\!\!\!+ \mathit{Bool} \\ \mathit{RefState}\ n\ \Gamma \Leftrightarrow n == \Gamma.\mathit{state} \\ \mathit{RefState}\ (\Delta n)\ \Gamma \Leftrightarrow n == \Gamma.\mathit{state} \\ \mathit{RefState}\ (\Xi n)\ \Gamma \Leftrightarrow n == \Gamma.\mathit{state} \\ \mathit{RefState}\ (d;\ ds)\ \Gamma \Leftrightarrow (\mathit{RefState}\ d\ \Gamma) \vee (\mathit{RefState}\ ds\ \Gamma) \\ \mathit{RefState}\ (n : t)\ \Gamma \Leftrightarrow \mathit{false} \\ \mathit{RefState}\ (n, ns : t)\ \Gamma \Leftrightarrow \mathit{false} \end{array}$$

2.59 ReplaceChanType

Function that returns the type of a channel, replacing the generic names of the type by the new types. It receives as arguments the generic name, the type that will replace the generic type, and the channel type.

$$\begin{array}{ll} \multicolumn{2}{l}{\mathit{ReplaceChanType} : \mathit{NAME} \rightarrow\!\!\!\!\!+ T \rightarrow\!\!\!\!\!+ T \rightarrow\!\!\!\!\!+ T} \\ \mathit{ReplaceChanType}\ n\ nt\ \mathbb{P}\,t & = \mathbb{P}(\mathit{ReplaceChanType}\ n\ nt\ t) \\ \mathit{ReplaceChanType}\ n\ nt\ (t \times ts) & = (\mathit{ReplaceChanType}\ n\ nt\ t)\ \times \\ & \quad (\mathit{ReplaceChanType}\ n\ nt\ ts) \\ \mathit{ReplaceChanType}\ n\ nt\ t & = \mathbf{if}\ (t == n) \\ & \quad \mathbf{then}\ nt \\ & \quad \mathbf{else}\ t \end{array}$$

2.60 Set

Function that return the set of names from a list of names.

$$Set : SeqName \rightarrow \mathbb{P}\ NAME$$

$$Set\ n = \{n\}$$
$$Set\ n, ln = \{n\} \cup Set\ ln$$

2.61 VarsC

Function that extracts the input variables of a communication, and returns these variables map to their types, in accordance with the channel type that communicates these variables.

$$VarsC : Comm \nrightarrow TEnv \nrightarrow TEnv$$

$$VarsC(c\ cpl)\ \Gamma = (localVars = (Extract\ cpl\ (\Gamma.channels\ c)))$$

2.62 VarsDec

Function that receives a declaration, and returns the environment updated with the declared variables - and their variations (x', $x?$, $x!$) - associated with their corresponding types.

$$VarsDec : Dec \nrightarrow TEnv$$

$$\begin{aligned} VarsDec\ (d;\ ds) &= (VarsDec\ d) \oplus (VarsDec\ ds) \\ VarsDec\ (x, xs : T) &= (VarsDec\ (x : T)) \oplus (VarsDec\ (xs : T)) \\ VarsDec\ (x : T) &= (localVars = (ExtractVars\ x\ T)) \end{aligned}$$

2.63 VarsParDec

Function that extracts the variables, and their corresponding types, of a parametrised command declaration. It returns an environment that contains the variables and their types, associated with the process name passed as argument.

$$VarsParDec : ParDec \nrightarrow TEnv$$

$$VarsParDec\ (sym\ d) = VarsDec\ d$$
$$\textbf{where}:\ sym = \ \textbf{val}\ \mid\ \textbf{res}\ \mid\ \textbf{valres},$$

2.64 Verify

Function that verifies if the expressions, within the list of expressions passed as argument, are well typed and have the same type of the corresponding variables, within in the list of names passed as argument.

$$Verify : SeqName \nrightarrow SeqExp \nrightarrow TEnv \nrightarrow Bool$$

$$Verify\ n\ e\ \Gamma \Leftrightarrow \Gamma \rhd e : \textbf{Expression}(\Gamma.localVars\ n)$$
$$Verify\ (n, ln)\ (e, le)\ \Gamma \Leftrightarrow (Verify\ n\ e\ \Gamma) \wedge (Verify\ ln\ le\ \Gamma)$$

2.65 override

Function that unifies the fields of two environment passed as arguments, and return only one environment as result.

$$\_ \oplus \_ : TEnv \nrightarrow TEnv \nrightarrow TEnv$$

$$
\begin{array}{l}
\Gamma \oplus \Gamma' = \\
\quad (channels = \Gamma.channels \ \cup \ \Gamma'.channels, \\
\quad genericChannels = \Gamma.genericChannels \ \cup \ \Gamma'.genericChannels, \\
\quad chansets = \Gamma.chansets \ \cup \ \Gamma'.chansets, \\
\quad processes = \Gamma.processes \ \cup \ \Gamma'.processes, \\
\quad definedProcs = \Gamma.definedProcs \ \cup \ \Gamma'.definedProcs, \\
\quad parProcesses = \Gamma.parProcesses \ \cup \ \Gamma'.parProcesses, \\
\quad indexProcesses = \Gamma.indexProcesses \ \cup \ \Gamma'.indexProcesses, \\
\quad genProcesses = \Gamma.genProcesses \ \cup \ \Gamma'.genProcesses, \\
\quad zDefs = \Gamma.zDefs \ \cup \ \Gamma'.zDefs, \\
\quad defNames = \Gamma.defNames \ \cup \ \mathrm{dom}(\Gamma'.channels) \ \cup \ \mathrm{dom}(\Gamma'.chansets) \cup \\
\qquad\qquad\qquad \cup \, \Gamma'.definedProcs \ \cup \ \mathrm{dom}(\Gamma'.zDefs), \\
\quad localVars = \Gamma.localVars \ \cup \ \Gamma'.localVars, \\
\quad params = \Gamma.params \ \cup \ \Gamma'.params, \\
\quad namesets = \Gamma.namesets \ \cup \ \Gamma'.namesets, \\
\quad actions = \Gamma.actions \ \cup \ \Gamma'.actions, \\
\quad definedActs = \Gamma.definedActs \ \cup \ \Gamma'.definesActs, \\
\quad parActions = \Gamma.parActions \ \cup \ \Gamma'.parActions, \\
\quad localZDefs = \Gamma.localZDefs \ \cup \ \Gamma'.localZDefs, \\
\quad localDefNames = \Gamma.localDefNames \ \cup \ \Gamma'.localVars \ \cup \ \Gamma'.namesets \cup \\
\qquad\qquad\qquad\qquad \cup \, \Gamma'.definedActs \ \cup \ \mathrm{dom}(\Gamma'.localZDefs) \ \cup \ \mathrm{dom}(\Gamma'.params), \\
\quad usedChansProc = \Gamma.usedChansProc \ \cup \ \Gamma'.usedChansProc)
\end{array}
$$