Formal Semantics of H-VHDL

Vincent Iampietro

Formal semantics of $\mathcal{H}$-VHDL

VINCENT IAMPIETRO

May 9, 2022

This document serves as a reference for the definition of the syntax and semantics of a subset of VHDL named $\mathcal{H}$-VHDL. $\mathcal{H}$-VHDL aims at the definition of synthesizable designs with a synchronous behavior (i.e. the execution of the designs is synchronized with a clock signal).

1 Abstract syntax of $\mathcal{H}$-VHDL

e	::=	$name$	read a signal, a local variable or a generic constant value
	|	cst	constant
	|	$bop(e_1, e_2)$	binary operation
	|	$uop(e)$	unary operation
	|	`(`e^+`)`	aggregate expression
$name$	::=	id	read a signal, local variable, or generic constant value
	|	id`(`e`)`	read value of an array signal or local variable at index e
cst	::=	n | b	natural or Boolean
bop	::=	`and` | `or`	Boolean operators
	|	`add` | `sub`	natural number arithmetic
	|	`eq` | `ne` | `gt` | `ge` | `lt` | `le`	comparisons
uop	::=	`not`	Boolean negation

Table 1: Expressions

τ	::=	`bool`	boolean
	|	`nat (`e_1`,` e_2`)`	natural range e_1 to e_2
	|	`array (`τ`,` e_1`,` e_2`)`	array of τ with index range e_1 to e_2

Table 2: Type indication

ss	::=	$name \Leftarrow e$	assignment to a signal
	|	$name := e$	assignment to a local variable
	|	`if(`e`){`ss_1`} else {`ss_2`}`	conditional
	|	`for(`id, e_1, e_2`){`ss`}`	range loop
	|	`falling{`ss`}`	falling edge block
	|	`rising{`ss`}`	rising edge block
	|	`rst {`ss_1`} else {`ss_2`}`	reset conditional
	|	ss_1;ss_2	sequence
	|	`null`	no operation

Table 3: Sequential statements

cs	$::= ps$	process statement
	$\mid comp$	component (or design) instantiation statement
	$\mid cs_1 \parallel cs_2$	parallel composition
	$\mid$ `null`	no operation
ps	$::=$ `ps`$(id,$	process identifier
	$\quad vars = \{(id, \tau)^*\},$	local variable declarations
	$\quad body = ss)$	statement body
$comp$	$::=$ `comp`$(id_c,$	component instance identifier
	$\quad id_e,$	instantiated design identifier
	$\quad g = \{(id \Rightarrow e)^*\},$	generic constant map
	$\quad i = \{(name \Rightarrow e)^*\},$	input port map
	$\quad o = \{((id \Rightarrow (name \mid$ `open`$))\mid(id(e) \Rightarrow name))^*\})$	output port map

Table 4: Concurrent statements

$design$	$::= \{gens = \{(id, \tau, e)^*\},$	generic constants
	$\quad ports = \{((\texttt{in}\mid\texttt{out}), id, \tau)^*\},$	input and output ports
	$\quad sigs = \{(id, \tau)^*\},$	internal signals
	$\quad beh = cs\}$	design behavior

Table 5: Design

2 Semantic domains

Table 6: The t (type) and v (value) semantic types.

t	::= `bool`	Boolean type
	| `nat`(n_1, n_2)	natural range n_1 to n_2
	| `array`(t, n_1, n_2)	array of t with index range n_1 to n_2
v	::= b	Boolean
	| n	natural number (limited to `NATMAX`)
	| a	array of values
a	::= (v^+)	

`NATMAX` denotes the maximum value for a natural number. The `NATMAX` value depends on the implementation of the VHDL language; `NATMAX` must at least be equal to $2^{31} - 1$.

Definition 1 (Elaborated design)**.** *An elaborated design $\Delta \in ElDesign$ is a record $<G, I, O, S, P, C>$ where:*

- $G \in id \nrightarrow (t \times v)$ *is the function yielding the type and the value of generic constants.*
- $I \in id \nrightarrow t$ *is the function yielding the type of input ports.*
- $O \in id \nrightarrow t$ *is the function yielding the type of output ports.*
- $S \in id \nrightarrow t$ *is the function yielding the type of declared signals.*
- $P \in id \nrightarrow (id \nrightarrow (t \times v))$ *is the function associating process identifiers to their local environment.*
- $C \in id \nrightarrow ElDesign$ *is the function mapping component instance identifiers to their own elaborated design version.*

We assume that there is no overlapping between the identifiers of the sub-environments of an elaborate design (i.e, an identifier belongs to at most one sub-environment), and also between the identifiers of the sub-environments and the identifiers of local environments. When there is no ambiguity, we write $\Delta(x)$ to denote the value returned for identifier x, where x is looked up in the appropriate field of Δ. We write $x \in \Delta$ to state that identifier x is defined in the domain of one of Δ's field. We note $\Delta(x) \leftarrow v$ the overriding of the value associated to identifier x with value v in the appropriate field of Δ, $\Delta \cup (x, v)$ to note the addition of the mapping from identifier x to value v in the appropriate field of Δ, that assuming $x \notin \Delta$. We write $x \in \mathcal{F}(\Delta)$, where $\mathcal{F}$ is a field of Δ, when more precision is needed regarding the lookup of identifier x in the record Δ.

Definition 2 (Design state)**.** *A design state $\sigma \in \Sigma$ is a couple $(\mathcal{S}, \mathcal{C})$ where:*

- $\mathcal{S} \in id \nrightarrow v$ *is the signal store, i.e. the function yielding the current values of ports and declared signals.*
- $\mathcal{C} \in id \nrightarrow \Sigma$ *is the design instance store, i.e. the function yielding the current state of design instances.*

When there is no ambiguity regarding which store a given identifier belongs to, we use $\sigma(id)$ as a shorthand notation for $\mathcal{S}(\sigma)(id)$ or $\mathcal{C}(\sigma)(id)$. Similarly, we write $id \in \sigma$ as a shorthand notation for $id \in \mathsf{dom}(\mathcal{S}(\sigma))$ or $id \in \mathsf{dom}(\mathcal{C}(\sigma))$.

3 Elaboration rules

3.1 Implicit default value

The following predicate states that a semantic type is well-formed.

$$\frac{}{\mathcal{WF}(\texttt{bool})}\ \textsc{WFBool} \qquad \frac{}{\mathcal{WF}(\texttt{nat}(l,u))}\ \textsc{WFNat}\ l \le u \le \texttt{NATMAX} \qquad \frac{\mathcal{WF}(t)}{\mathcal{WF}(\texttt{array}(t,l,u))}\ \textsc{WFArr}\ l \le u \le \texttt{NATMAX}$$

Table 7: Well-formed semantic type

According to the VHDL LRM, at the declaration of a port, a signal or a variable, these items must receive an implicit default value depending on their types [1, p.61, 64, 173]. The dv relation determines the default value for a given type.

$$\frac{}{\texttt{bool} \xrightarrow{dv} \texttt{false}}\ \textsc{DefaultVBool} \qquad \frac{\mathcal{WF}(\texttt{nat}(l,u))}{\texttt{nat}(l,u) \xrightarrow{dv} l}\ \textsc{DefaultVCNat} \qquad \frac{\mathcal{WF}(\texttt{array}(t,l,u)) \quad t \xrightarrow{dv} v}{\texttt{array}(t,l,u) \xrightarrow{dv} \texttt{create\_array}(size, v)}\ \textsc{DefaultVCArr}\ size = (u-l)+1$$

Table 8: Type default value

The $\texttt{create\_array}(size, v)$ expression yields an array of size $size$ where each element is initialized with the value v.

3.2 Typing relation

The typing relation $\in_c$ checks that a given value conforms to a given type.

$$\frac{}{b \in_c \texttt{bool}}\ \textsc{IsBool}\ b \in \mathbb{B} \qquad \frac{\mathcal{WF}(\texttt{nat}(l,u))}{n \in_c \texttt{nat}(l,u)}\ \textsc{IsCNat}\ n \in [l,u] \qquad \frac{\mathcal{WF}(\texttt{array}(t,l,u)) \quad v_i \in_c t}{(v_1,\ldots,v_n) \in_c \texttt{array}(t,l,u)}\ \textsc{IsArray}\ \begin{array}{l} i = 1,\ldots,n \\ n = (u-l)+1 \end{array}$$

3.3 Static expressions

Static expressions are either locally static or globally static; the LRM defines locally static and globally static expressions as follows.

3.3.1 Locally static expressions

The SE_l relation, defined by the following rules, states that an expression is locally static.

$$\frac{}{SE_l(n)}\ \textsc{LSENat}\ n \in \mathbb{N} \qquad \frac{}{SE_l(b)}\ \textsc{LSEBool}\ b \in \mathbb{B} \qquad \frac{SE_l(e)}{SE_l(uop(e))}\ \textsc{LSEUOp} \qquad \frac{SE_l(e_1) \quad SE_l(e_2)}{SE_l(bop(e_1,e_2))}\ \textsc{LSEBOp}$$

3.3.2 Globally static expressions

The SE_g relation, defined by the following rules, checks that an expression is globally static in the context of an elaborated design Δ.

$$\frac{SE_l(e)}{\Delta \vdash SE_g(e)}\ \textsc{GSELocal} \qquad \frac{}{\Delta \vdash SE_g(id)}\ \textsc{GSEGen}\ id \in G(\Delta) \qquad \frac{\Delta \vdash SE_g(e_i)}{\Delta \vdash SE_g((e_1, \ldots, e_n))}\ \textsc{GSEAggregate}\ i = 1, \ldots, n$$

3.4 Type indication elaboration

The *econstr* relation checks that a constraint is well-formed and evaluates the constraint bounds.

$$\textsc{EConstr}\ \frac{\begin{array}{lll}\Delta \vdash SE_g(e_1) & \Delta \vdash e_1 \xrightarrow{e} n_1 & n_1 \in_c \texttt{nat}(0, \texttt{NATMAX}) \\ \Delta \vdash SE_g(e_2) & \Delta \vdash e_2 \xrightarrow{e} n_2 & n_2 \in_c \texttt{nat}(0, \texttt{NATMAX})\end{array}}{\Delta \vdash (e_1, e_2) \xrightarrow{econstr} (n_1, n_2)}\ n_1 \leq n_2$$

The *et* relation checks the well-formedness of a type indication τ, and transforms it into a semantic type t (as defined in Table 6).

$$\textsc{ETypeBool}\ \frac{}{\Delta \vdash \texttt{bool} \xrightarrow{et} \texttt{bool}} \qquad \textsc{ETypeNat}\ \frac{\Delta \vdash (e_1, e_2) \xrightarrow{econstr} (n_1, n_2)}{\Delta \vdash \texttt{nat}(e_1, e_2) \xrightarrow{et} \texttt{nat}(n_1, n_2)} \qquad \textsc{ETypeArray}\ \frac{\Delta \vdash \tau \xrightarrow{et} t \quad \Delta \vdash (e_1, e_2) \xrightarrow{econstr} (n_1, n_2)}{\Delta \vdash \texttt{array}(\tau, e_1, e_2) \xrightarrow{et} \texttt{array}(t, n_1, n_2)}$$

The $econstr_g$ relation checks that a *generic* constraint (i.e, a constraint appearing in a type indication associated with a generic constant declaration) is well-formed and evaluates the constraint bounds.

$$\textsc{EConstrG}\ \frac{\begin{array}{lll}SE_l(e_1) & e_1 \xrightarrow{e} n_1 & n_1 \in_c \texttt{nat}(0, \texttt{NATMAX}) \\ SE_l(e_2) & e_2 \xrightarrow{e} n_2 & n_2 \in_c \texttt{nat}(0, \texttt{NATMAX})\end{array}}{(e_1, e_2) \xrightarrow{econstr_g} (n_1, n_2)}\ n_1 \leq n_2$$

The et_g relation is specially defined to check the well-formedness of a type indication associated with a generic constant declaration.

$$\frac{}{\texttt{bool} \xrightarrow{et_g} \texttt{bool}}\ \textsc{ETypeGBool} \qquad \frac{(e_1, e_2) \xrightarrow{econstr_g} (n_1, n_2)}{\texttt{nat}(e_1, e_2) \xrightarrow{et_g} \texttt{nat}(n_1, n_2)}\ \textsc{ETypeGNat}$$

3.5 Design elaboration

Given design store $\mathcal{D} \in id \rightharpoonup design$ and a dimensioning function $\mathcal{M}_g \in id \rightharpoonup v$, the elaboration phase generates an elaborated design $\Delta \in ElDesign$ along with a *default* state $\sigma_e \in \Sigma$ out of a $\mathcal{H}$-VHDL design $d \in design$. The elaboration performs static type-checking over the declarative (gens, ports and sigs) and behavioral (behavior) parts of a design, but also checks the well-formedness of generic and port maps in design instantiation statements, and finally checks that there is no multiply-driven signal.

$$\frac{\Delta_\emptyset, \mathcal{M}_g \vdash d.gens \xrightarrow{egens} \Delta \qquad \Delta, \sigma_\emptyset \vdash d.ports \xrightarrow{eports} \Delta', \sigma \qquad \Delta', \sigma \vdash d.sigs \xrightarrow{esigs} \Delta'', \sigma' \qquad \mathcal{D}, \Delta'', \sigma' \vdash d.beh \xrightarrow{ebeh} \Delta''', \sigma'' \qquad \texttt{NoMDInCs}(\Delta''', d.beh)}{\mathcal{D}, \mathcal{M}_g \vdash d \xrightarrow{elab} \Delta''', \sigma''}\ \textsc{DesignElab}$$

3.6 Generic clause elaboration

$$\frac{\tau \xrightarrow{et_g} t \qquad e \xrightarrow{e} v \qquad SE_l(e) \qquad \mathcal{M}(id) \in_c t \qquad v \in_c t \qquad \mathcal{M}(id) = \lfloor v \rfloor}{\Delta, \mathcal{M} \vdash (id, \tau, e) \xrightarrow{egens} \Delta \cup (id, (t, \mathcal{M}(id)))}\ id \notin \Delta\ \textsc{GenElabDimen}$$

$$\frac{\tau \xrightarrow{et_g} t \qquad e \xrightarrow{e} v \qquad SE_l(e) \qquad v \in_c t}{\Delta, \mathcal{M} \vdash (id, \tau, e) \xrightarrow{egens} \Delta \cup (id, (t, v))}\ \begin{matrix} id \notin \Delta \\ id \notin \mathcal{M} \end{matrix}\ \textsc{GenElabDefault}$$

$$\frac{\Delta, \mathcal{M} \vdash (id, \tau, e) \xrightarrow{egens} \Delta' \qquad \Delta', \mathcal{M} \vdash gens \xrightarrow{egens} \Delta''}{\Delta, \mathcal{M} \vdash (id, \tau, e), gens \xrightarrow{egens} \Delta''}\ \textsc{GenElabComp}$$

3.7 Port clause elaboration

$$\frac{\Delta \vdash \tau \xrightarrow{et} t \qquad t \xrightarrow{dv} v}{\Delta, \sigma \vdash (\texttt{in}, id, \tau) \xrightarrow{eports} I(\Delta) \cup (id, t),\ \sigma \cup (id, v)}\ \begin{matrix} id \notin \Delta \\ id \notin \sigma \end{matrix}\ \textsc{InPortElab}$$

OutPortElab

$$\frac{\Delta \vdash \tau \xrightarrow{et} t \qquad t \xrightarrow{dv} v}{\Delta, \sigma \vdash (\texttt{out}, id, \tau) \xrightarrow{eports} O(\Delta) \cup (id, t),\ \sigma \cup (id, v)} \quad \begin{array}{l} id \notin \Delta \\ id \notin \sigma \end{array}$$

PortElabComp

$$\frac{\Delta, \sigma \vdash pdecl \xrightarrow{eports} \Delta', \sigma' \qquad \Delta', \sigma' \vdash ports \xrightarrow{eports} \Delta'', \sigma''}{\Delta, \sigma \vdash pdecl, ports \xrightarrow{eports} \Delta'', \sigma''}$$

where $pdecl ::= \big((\texttt{in}|\texttt{out}), id, \tau\big)$.

3.8 Architecture declarative part elaboration

SigElab

$$\frac{\Delta \vdash \tau \xrightarrow{et} t \qquad t \xrightarrow{dv} v}{\Delta, \sigma \vdash (id, \tau) \xrightarrow{esigs} S(\Delta) \cup (id, t),\ \sigma \cup (id, v)} \quad \begin{array}{l} id \notin \Delta \\ id \notin \sigma \end{array}$$

SigElabComp

$$\frac{\Delta, \sigma \vdash (id, \tau) \xrightarrow{esigs} \Delta', \sigma' \qquad \Delta', \sigma' \vdash sigs \xrightarrow{esigs} \Delta'', \sigma''}{\Delta, \sigma \vdash (id, \tau), sigs \xrightarrow{esigs} \Delta'', \sigma''}$$

3.9 Behavior elaboration

CsParElab

$$\frac{\mathcal{D}, \Delta, \sigma \vdash cs \xrightarrow{ebeh} \Delta', \sigma' \qquad \mathcal{D}, \Delta', \sigma' \vdash cs' \xrightarrow{ebeh} \Delta'', \sigma''}{\mathcal{D}, \Delta, \sigma \vdash cs \mid\mid cs' \xrightarrow{ebeh} \Delta'', \sigma''}$$

CsNullElab

$$\frac{}{\mathcal{D}, \Delta, \sigma \vdash \texttt{null} \xrightarrow{ebeh} \Delta, \sigma}$$

PsElab

$$\frac{\Delta, \Lambda_\emptyset \vdash vars \xrightarrow{evars} \Lambda \qquad \Delta, \mathcal{S}, \Lambda \vdash \texttt{valid}_{ss}(ss)}{\mathcal{D}, \Delta, \sigma \vdash \texttt{ps}(id, vars, ss) \xrightarrow{ebeh} \Delta \cup (id, \Lambda), \sigma} \quad \begin{array}{l} id \notin \Delta \\ \sigma = (\mathcal{S}, \mathcal{C}) \end{array}$$

CompElab

$$\frac{\begin{array}{cc} \mathcal{M}_\emptyset \vdash g \xrightarrow{emapg} \mathcal{M} & \Delta, \Delta_c, \mathcal{S} \vdash \texttt{valid}_{ipm}(i) \\ \mathcal{D}, \mathcal{M} \vdash d \xrightarrow{elab} \Delta_c, \sigma_c & \Delta, \Delta_c \vdash \texttt{valid}_{opm}(o) \end{array}}{\mathcal{D}, \Delta, \sigma \vdash \texttt{comp}(id_c, id_e, g, i, o) \xrightarrow{ebeh} \Delta \cup (id_c, \Delta_c),\ \sigma \cup (id_c, \sigma_c)} \quad \begin{array}{l} id_c \notin \Delta,\ id_c \notin \sigma \\ \sigma = (\mathcal{S}, \mathcal{C}) \\ \mathcal{D}(id_e) = \lfloor d \rfloor \end{array}$$

3.9.1 Process declarative part elaboration

VarElab

$$\frac{\Delta \vdash \tau \xrightarrow{et} t \qquad t \xrightarrow{dv} v}{\Delta, \Lambda \vdash (id, \tau) \xrightarrow{evars} \Lambda \cup (id, (t, v))} \quad \begin{array}{l} id \notin \Lambda \\ id \notin \Delta \end{array}$$

VarElabComp

$$\frac{\Delta, \Lambda \vdash (id, \tau) \xrightarrow{evars} \Lambda' \qquad \Delta, \Lambda' \vdash vars \xrightarrow{evars} \Lambda''}{\Delta, \Lambda \vdash (id, \tau), vars \xrightarrow{evars} \Lambda''}$$

3.9.2 Sequential statement validity

ValidSig

$$\frac{\Delta, \mathcal{S}, \Lambda \vdash e \xrightarrow{e} v \qquad v \in_c t}{\Delta, \mathcal{S}, \Lambda \vdash \texttt{valid}_{ss}(id \Leftarrow e)} \quad \begin{array}{l} id \in S(\Delta) \cup O(\Delta) \\ \Delta(id) = \lfloor t \rfloor \end{array}$$

ValidIdxSig

$$\frac{\begin{array}{ll} \Delta, \mathcal{S}, \Lambda \vdash e \xrightarrow{e} v & v \in_c t \\ \Delta, \mathcal{S}, \Lambda \vdash e_i \xrightarrow{e} v_i & v_i \in_c \texttt{nat}(n, m) \end{array}}{\Delta, \mathcal{S}, \Lambda \vdash \texttt{valid}_{ss}(id(e_i) \Leftarrow e)} \quad \begin{array}{l} id \in S(\Delta) \cup O(\Delta) \\ \Delta(id) = \lfloor \texttt{array}(t, n, m) \rfloor \end{array}$$

ValidVar

$$\frac{\Delta, \mathcal{S}, \Lambda \vdash e \xrightarrow{e} v \qquad v \in_c t}{\Delta, \mathcal{S}, \Lambda \vdash \texttt{valid}_{ss}(id := e)} \quad \begin{array}{l} id \in \Lambda \\ \Lambda(id) = \lfloor (t, val) \rfloor \end{array}$$

ValidIdxVar

$$\frac{\begin{array}{ll} \Delta, \mathcal{S}, \Lambda \vdash e \xrightarrow{e} v & v \in_c t \\ \Delta, \mathcal{S}, \Lambda \vdash e_i \xrightarrow{e} v_i & v_i \in_c \texttt{nat}(n, m) \end{array}}{\Delta, \Lambda \vdash \texttt{valid}_{ss}(id(e_i) := e)} \quad \begin{array}{l} id \in \Lambda \\ \Lambda(id) = \lfloor (\texttt{array}(t, n, m), a) \rfloor \end{array}$$

ValidCond

$$\frac{\begin{array}{lll} & & \Delta, \mathcal{S}, \Lambda \vdash \texttt{valid}_{ss}(ss_1) \\ \Delta, \mathcal{S}, \Lambda \vdash e \xrightarrow{e} v & v \in_c \texttt{bool} & \Delta, \mathcal{S}, \Lambda \vdash \texttt{valid}_{ss}(ss_2) \end{array}}{\Delta, \mathcal{S}, \Lambda \vdash \texttt{valid}_{ss}(\texttt{if}(e)\{ss_1\}\texttt{else}\{ss_2\})}$$

ValidLoop

$$\frac{\begin{array}{lll} \Delta, \mathcal{S}, \Lambda \vdash e_1 \xrightarrow{e} v_1 & v_1 \in_c \texttt{nat}(0, \texttt{NATMAX}) & \\ \Delta, \mathcal{S}, \Lambda \vdash e_2 \xrightarrow{e} v_2 & v_2 \in_c \texttt{nat}(0, \texttt{NATMAX}) & \Delta, \mathcal{S}, \Lambda(id) \leftarrow (\texttt{nat}(0, \texttt{NATMAX}), v_1) \vdash \texttt{valid}_{ss}(ss) \end{array}}{\Delta, \mathcal{S}, \Lambda \vdash \texttt{valid}_{ss}(\texttt{for}(id, e_1, e_2)\{ss\})}$$

ValidRising

$$\frac{\Delta, \mathcal{S}, \Lambda \vdash \texttt{valid}_{ss}(ss)}{\Delta, \mathcal{S}, \Lambda \vdash \texttt{valid}_{ss}(\texttt{rising}\{ss\})}$$

ValidFalling

$$\frac{\Delta, \mathcal{S}, \Lambda \vdash \texttt{valid}_{ss}(ss)}{\Delta, \mathcal{S}, \Lambda \vdash \texttt{valid}_{ss}(\texttt{falling}\{ss\})}$$

ValidNull

$$\frac{}{\Delta, \mathcal{S}, \Lambda \vdash \texttt{valid}_{ss}(\texttt{null})}$$

ValidRst

$$\frac{\Delta, \mathcal{S}, \Lambda \vdash \texttt{valid}_{ss}(ss_1) \qquad \Delta, \mathcal{S}, \Lambda \vdash \texttt{valid}_{ss}(ss_2)}{\Delta, \mathcal{S}, \Lambda \vdash \texttt{valid}_{ss}(\texttt{rst}\{ss_1\}\{ss_2\})}$$

ValidSeq

$$\frac{\Delta, \mathcal{S}, \Lambda \vdash \texttt{valid}_{ss}(ss_1) \qquad \Delta, \mathcal{S}, \Lambda \vdash \texttt{valid}_{ss}(ss_2)}{\Delta, \mathcal{S}, \Lambda \vdash \texttt{valid}_{ss}(ss_1; ss_2)}$$

3.9.3 Map validity

Dimensioning function construction

AssocGELab

$$\frac{SE_l(e) \quad e \xrightarrow{e} v}{\mathcal{M} \vdash (id \Rightarrow e) \xrightarrow{emapg} \mathcal{M} \cup (id, v)} \; id \notin \mathcal{M}$$

GMELab

$$\frac{\mathcal{M} \vdash (id \Rightarrow e) \xrightarrow{emapg} \mathcal{M}' \quad \mathcal{M}' \vdash g \xrightarrow{emapg} \mathcal{M}''}{\mathcal{M} \vdash (id \Rightarrow e), g \xrightarrow{emapg} \mathcal{M}''}$$

Input port map validity

ListIPMSimple

$$\frac{\Delta, \mathcal{S} \vdash e \xrightarrow{e} v \quad v \in_c t}{\Delta, \Delta_c, \mathcal{S}, \mathcal{L} \vdash (id \Rightarrow e) \xrightarrow{list_{ipm}} \mathcal{L} \cup \{id\}} \; \begin{array}{l} id \notin \mathcal{L},\, id \in I(\Delta_c) \\ \nexists v_i \text{ s.t. } (id, v_i) \in \mathcal{L} \\ \Delta_c(id) = \lfloor t \rfloor \end{array}$$

ListIPMPartial

$$\frac{\begin{array}{cc} & e_i \xrightarrow{e} v_i \quad v_i \in_c \texttt{nat}(n,m) \\ SE_l(e_i) \quad \Delta, \mathcal{S} \vdash e \xrightarrow{e} v & v \in_c t \end{array}}{\Delta, \Delta_c, \mathcal{S}, \mathcal{L} \vdash (id(e_i) \Rightarrow e) \xrightarrow{list_{ipm}} \mathcal{L} \cup \{(id, v_i)\}} \; \begin{array}{l} id \notin \mathcal{L},\, (id, v_i) \notin \mathcal{L} \\ id \in I(\Delta_c) \\ \Delta_c(id) = \lfloor \texttt{array}(t, n, m) \rfloor \end{array}$$

ListIPMCons

$$\frac{\Delta, \Delta_c, \mathcal{S}, \mathcal{L} \vdash (name \Rightarrow e) \xrightarrow{list_{ipm}} \mathcal{L}' \quad \Delta, \Delta_c, \mathcal{S}, \mathcal{L}' \vdash i \xrightarrow{list_{ipm}} \mathcal{L}''}{\Delta, \Delta_c, \mathcal{S}, \mathcal{L} \vdash (name \Rightarrow e),\ i \xrightarrow{list_{ipm}} \mathcal{L}''}$$

where $\mathcal{L} \subseteq id \cup (id \times \mathbb{N})$.

$$\texttt{check}_{pm}(Ports, \mathcal{L}) \equiv \forall id_f \in \texttt{dom}(Ports),\ id_f \in \mathcal{L} \ \vee \ \exists t \in type, n, m \in \mathbb{N}, (Ports(id_f) = \texttt{array}(t, n, m) \wedge \forall i \in [n, m],\ (id_f, i) \in \mathcal{L})$$

where $Ports \in id \nrightarrow type$, and $\mathcal{L} \subseteq id \cup (id \times \mathbb{N})$.

ValidIPM

$$\frac{\Delta, \Delta_c, \mathcal{S}, \mathcal{L}_\emptyset \vdash i \xrightarrow{list_{ipm}} \mathcal{L} \quad \texttt{check}_{pm}(I(\Delta_c), \mathcal{L})}{\Delta, \Delta_c, \mathcal{S} \vdash \texttt{valid}_{ipm}(i)}$$

Output port map validity

$$\frac{}{\Delta, \Delta_c, \mathcal{L} \vdash (id_f \Rightarrow id_a) \xrightarrow{list_{opm}} \mathcal{L} \cup \{id_f\}} \text{ListOPMSimpleToSimple}$$
$id_f \notin \mathcal{L}$
$id_f \in O(\Delta_c)$
$id_a \in S(\Delta) \cup O(\Delta)$
$\Delta_c(id_f) = \Delta(id_a) = \lfloor t \rfloor$

$$\frac{SE_l(e_i) \quad e_i \xrightarrow{e} v_i \quad v_i \in_c \texttt{nat}(n,m)}{\Delta, \Delta_c, \mathcal{L} \vdash (id_f \Rightarrow id_a(e_i)) \xrightarrow{list_{opm}} \mathcal{L} \cup \{id_f\}} \text{ListOPMSimpleToPartial}$$
$id_f \notin \mathcal{L}$
$id_f \in O(\Delta_c)$
$id_a \in S(\Delta) \cup O(\Delta)$
$\Delta_c(id_f) = \lfloor t \rfloor$
$\Delta(id_a) = \lfloor \texttt{array}(t, n, m) \rfloor$

$$\frac{}{\Delta, \Delta_c, \mathcal{L} \vdash (id_f \Rightarrow \texttt{open}) \xrightarrow{list_{opm}} \mathcal{L} \cup \{id_f\}} \text{ListOPMSimpleToOpen}$$
$id_f \notin \mathcal{L}$
$id_f \in O(\Delta_c)$

$$\frac{SE_l(e_i) \quad e_i \xrightarrow{e} v_i \quad v_i \in_c \texttt{nat}(n,m)}{\Delta, \Delta_c, \mathcal{L} \vdash (id_f(e_i) \Rightarrow id_a) \xrightarrow{list_{opm}} \mathcal{L} \cup \{(id_f, v_i)\}} \text{ListOPMPartialToSimple}$$
$id_f, (id_f, v_i) \notin \mathcal{L}$
$id_f \in O(\Delta_c)$
$id_a \in S(\Delta) \cup O(\Delta)$
$\Delta_c(id_f) = \lfloor \texttt{array}(t, n, m) \rfloor$
$\Delta(id_a) = \lfloor t \rfloor$

$$\frac{\begin{array}{ll} SE_l(e'_i) & e'_i \xrightarrow{e} v'_i \quad v'_i \in_c \texttt{nat}(n',m') \\ SE_l(e_i) & e_i \xrightarrow{e} v_i \quad v_i \in_c \texttt{nat}(n,m) \end{array}}{\Delta, \Delta_c, \mathcal{L} \vdash (id_f(e_i) \Rightarrow id_a(e'_i)) \xrightarrow{list_{opm}} \mathcal{L} \cup \{(id_f, v_i)\}} \text{ListOPMPartialToPartial}$$
$id_f, (id_f, v_i) \notin \mathcal{L}$
$id_f \in O(\Delta_c)$
$id_a \in S(\Delta) \cup O(\Delta)$
$\Delta_c(id_f) = \lfloor \texttt{array}(t, n, m) \rfloor$
$\Delta(id_a) = \lfloor \texttt{array}(t, n', m') \rfloor$

$$\frac{\Delta, \Delta_c, \mathcal{L} \vdash assoc_{po} \xrightarrow{list_{opm}} \mathcal{L}' \qquad \Delta, \Delta_c, \mathcal{L}' \vdash o \xrightarrow{list_{opm}} \mathcal{L}''}{\Delta, \Delta_c, \mathcal{L} \vdash assoc_{po},\ o \xrightarrow{list_{opm}} \mathcal{L}''} \text{ListOPMCons}$$

where $\mathcal{L} \subseteq id \cup (id \times \mathbb{N})$ and $assoc_{po} ::= (id \Rightarrow (name | \texttt{open})) | (id(e) \Rightarrow name)$.

$$\frac{\Delta, \Delta_c, \mathcal{L}_\emptyset, \mathcal{L}_{ids\emptyset} \vdash o \xrightarrow{list_{opm}} \mathcal{L}, \mathcal{L}_{ids}}{\Delta, \Delta_c \vdash \texttt{valid}_{opm}(o)} \text{ValidOPM}$$

3.9.4 Detection of multiply-driven signals

Assignment in sequential statement

The `is_assgd_in_ss` function returns `true` if the signal identifier id_s is assigned in the sequential statement ss, i.e. if id_s appears in the left part of a signal assignment statement.

$$
\begin{array}{rcl}
\texttt{is\_assgd\_in\_ss}(id_s \Leftarrow e, id_s) \\
\texttt{is\_assgd\_in\_ss}(id_s(e_1) \Leftarrow e_2, id_s) & = & \texttt{true} \\
\texttt{is\_assgd\_in\_ss}(\texttt{for}(id, e_1, e_2)\{ss\}, id_s) \\
\texttt{is\_assgd\_in\_ss}(\texttt{falling}\{ss\}, id_s) & = & \texttt{is\_assgd\_in\_ss}(ss, id_s) \\
\texttt{is\_assgd\_in\_ss}(\texttt{rising}\{ss\}, id_s) \\
\texttt{is\_assgd\_in\_ss}(\texttt{if}(e)\{ss_1\}\ \texttt{else}\ \{ss_2\}, id_s) \\
\texttt{is\_assgd\_in\_ss}(\texttt{rst}\ \{ss_1\}\ \texttt{else}\ \{ss_2\}, id_s) & = & \texttt{is\_assgd\_in\_ss}(ss_1, id_s) \\
\texttt{is\_assgd\_in\_ss}(ss_1; ss_2, id_s) & & \textit{or}\ \texttt{is\_assgd\_in\_ss}(ss_2, id_s) \\
otherwise & = & \texttt{false}
\end{array}
$$

Assignment in output port map

The `is_assgd_in_omap` function returns `true` if the signal identifier id_s is an actual part (i.e. the right part of an association) in the output port map o.

$$
\texttt{is\_assgd\_in\_omap}(o, id_s) = \begin{cases} \texttt{true} \text{ if } \exists name \text{ s.t. } (name \Rightarrow id_s) \in o \\ \texttt{false } otherwise \end{cases}
$$

Assignment in concurrent statement

$$
\begin{array}{rcl}
\texttt{is\_assgd\_in\_cs}(\texttt{ps}(id, vars, body), id_s) & = & \texttt{is\_assgd\_in\_ss}(body, id_s) \\
\texttt{is\_assgd\_in\_cs}(\texttt{comp}(id_c, id_e, g, i, o), id_s) & = & \texttt{is\_assgd\_in\_omap}(o, id_s) \\
\texttt{is\_assgd\_in\_cs}(cs_1 \mid\mid cs_2, id_s) & = & \texttt{is\_assgd\_in\_cs}(cs_1, id_s) \\
 & & \textit{or}\ \texttt{is\_assgd\_in\_cs}(cs_2, id_s) \\
\texttt{is\_assgd\_in\_cs}(\texttt{null}, id_s) & = & \texttt{false}
\end{array}
$$

Multiply-driven signal in output port map

$$
\texttt{is\_md\_in\_omap}(o, id_s) = \begin{cases} \texttt{is\_assgd\_in\_omap}(o \setminus \{(name \Rightarrow id_s)\}, id_s) \text{ if } \exists name \text{ s.t. } (name \Rightarrow id_s) \in o \\ \texttt{false otherwise} \end{cases}
$$

Multiply-driven signal in concurrent statement

$$
\begin{array}{rcl}
\texttt{is\_md\_in\_cs}(\texttt{comp}(id_c, id_e, g, i, o), id_s) & = & \texttt{is\_md\_in\_omap}(o, id_s) \\
\texttt{is\_md\_in\_cs}(cs_1 \mid\mid cs_2, id_s) & = & \texttt{is\_md\_in\_cs}(cs_1, id_s) \\
 & & \textit{or}\ \texttt{is\_md\_in\_cs}(cs_2, id_s) \\
 & & \textit{or}\ (\texttt{is\_assgd\_in\_cs}(cs_1, id_s) \\
 & & \quad \textit{and}\ \texttt{is\_assgd\_in\_cs}(cs_2, id_s)) \\
otherwise & = & \texttt{false}
\end{array}
$$

No multiply-driven signal check

$$
\texttt{NoMDInCs}(\Delta, cs) \equiv \forall id_s \in O(\Delta) \cup S(\Delta), \texttt{is\_md\_in\_cs}(cs, id_s) = \texttt{false}.
$$

4 Simulation rules

4.1 Evaluation of expressions

$$\textsc{Sig}\ \frac{\mathcal{S}(id) = \lfloor v \rfloor}{\Delta, \mathcal{S}, \Lambda \vdash id \xrightarrow{e} v}\ id \in S(\Delta) \cup I(\Delta) \qquad \textsc{Var}\ \frac{\Lambda(id) = \lfloor (t, v) \rfloor}{\Delta, \mathcal{S}, \Lambda \vdash id \xrightarrow{e} v} \qquad \textsc{Gen}\ \frac{G(\Delta)(id) = \lfloor (t, v) \rfloor}{\Delta, \mathcal{S}, \Lambda \vdash id \xrightarrow{e} v} \qquad \textsc{Out}\ \frac{\mathcal{S}(id) = \lfloor v \rfloor}{\Delta, \mathcal{S} \vdash id \xrightarrow{e_o} v}\ id \in O(\Delta)$$

$$\textsc{IdxOut}\ \frac{e_i \xrightarrow{e} n_i \qquad a[i] = \lfloor v \rfloor}{\Delta, \mathcal{S} \vdash id(e_i) \xrightarrow{e_o} v}\ \begin{array}{l} id \in O(\Delta) \\ \Delta(id) = \lfloor \texttt{array}(t, n, m) \rfloor \\ \mathcal{S}(id) = \lfloor a \rfloor \\ i = n_i - n \end{array} \qquad \textsc{IdxSig}\ \frac{\Delta, \mathcal{S}, \Lambda \vdash e_i \xrightarrow{e} n_i \qquad a[i] = \lfloor v \rfloor}{\Delta, \mathcal{S}, \Lambda \vdash id(e_i) \xrightarrow{e} v}\ \begin{array}{l} id \in S(\Delta) \cup I(\Delta) \\ \Delta(id) = \lfloor \texttt{array}(t, n, m) \rfloor \\ \mathcal{S}(id) = \lfloor a \rfloor \\ i = n_i - n \end{array}$$

$$\textsc{IdxVar}\ \frac{\Delta, \mathcal{S}, \Lambda \vdash e_i \xrightarrow{e} n_i \qquad a[i] = \lfloor v \rfloor}{\Delta, \mathcal{S}, \Lambda \vdash id(e_i) \xrightarrow{e} v}\ \begin{array}{l} id \in \Lambda \\ \Lambda(id) = \lfloor (\texttt{array}(t, n, m), a) \rfloor \\ i = n_i - n \end{array} \qquad \textsc{Cst}\ \frac{\texttt{vcst}(cst) = \lfloor v \rfloor}{\Delta, \mathcal{S}, \Lambda \vdash cst \xrightarrow{e} v}$$

$$\textsc{BOp}\ \frac{\Delta, \mathcal{S}, \Lambda \vdash e_1 \xrightarrow{e} v_1 \qquad \Delta, \mathcal{S}, \Lambda \vdash e_2 \xrightarrow{e} v_2 \qquad \texttt{vbop}(bop, v_1, v_2) = \lfloor v \rfloor}{\Delta, \mathcal{S}, \Lambda \vdash bop(e_1, e_2) \xrightarrow{e} v}$$

$$\textsc{Not}\ \frac{\Delta, \mathcal{S}, \Lambda \vdash e \xrightarrow{e} b}{\Delta, \mathcal{S}, \Lambda \vdash \texttt{not}(e) \xrightarrow{e} \neg b} \qquad \textsc{Aggreg}\ \frac{\Delta, \mathcal{S}, \Lambda \vdash e_i \xrightarrow{e} v_i}{\Delta, \mathcal{S}, \Lambda \vdash (e_1, \ldots, e_n) \xrightarrow{e} (v_1, \ldots, v_n)}\ i = 1, \ldots, n$$

Evaluation of constants

$$\begin{aligned} \texttt{vcst}(b) &= \lfloor b \rfloor \\ \texttt{vcst}(n) &= \begin{cases} \lfloor n \rfloor \text{ if } n \leq \texttt{NATMAX} \\ \emptyset \text{ otherwise} \end{cases} \end{aligned}$$

Evaluation of binary operators

$$
\begin{array}{rcl}
\texttt{vbop}(\texttt{and}, b_1, b_2) & = & \lfloor b_1 \ \&\& \ b_2 \rfloor \\
\texttt{vbop}(\texttt{or}, b_1, b_2) & = & \lfloor b_1 \ || \ b_2 \rfloor \\
\texttt{vbop}(\texttt{add}, n_1, n_2) & = & \begin{cases} \lfloor n_1 + n_2 \rfloor \text{ if } n_1 + n_2 \leq \texttt{NATMAX} \\ \emptyset \text{ otherwise} \end{cases} \\
\texttt{vbop}(\texttt{sub}, n_1, n_2) & = & \begin{cases} \lfloor n_1 - n_2 \rfloor \text{ if } n_1 \geq n_2 \\ \emptyset \text{ otherwise} \end{cases} \\
\texttt{vbop}(\texttt{eq}, b_1, b_2) & = & \lfloor (b_1 =_{\mathbb{B}} b_2) \rfloor \\
\texttt{vbop}(\texttt{eq}, n_1, n_2) & = & \lfloor (n_1 =_{\mathbb{N}} n_2) \rfloor \\
\texttt{vbop}(\texttt{eq}, a_1, a_2) & = & \lfloor (a_1 =_{a} a_2) \rfloor \\
\texttt{vbop}(\texttt{neq}, v_1, v_2) & = & \begin{cases} \lfloor \neg b \rfloor \text{ if } \texttt{vbop}(\texttt{eq}, v_1, v_2) = \lfloor b \rfloor \\ \emptyset \text{ otherwise} \end{cases} \\
\texttt{vbop}(\texttt{gt}, n_1, n_2) & = & \lfloor n_1 > n_2 \rfloor \\
\texttt{vbop}(\texttt{ge}, n_1, n_2) & = & \lfloor n_1 \geq n_2 \rfloor \\
\texttt{vbop}(\texttt{lt}, n_1, n_2) & = & \lfloor n_1 < n_2 \rfloor \\
\texttt{vbop}(\texttt{le}, n_1, n_2) & = & \lfloor n_1 \leq n_2 \rfloor
\end{array}
$$

The function $=_a \in a \to a \to \mathbb{B}$ is the equality between two arrays, where $a ::= (v^+)$ as defined in the *value* set presented in Table 6. It yields `true` if the two compared arrays are of the same size and have all their elements positionally equal; the result is `false` otherwise.

4.2 Evaluation of sequential statements

SigAssign

$$\frac{\Delta, \mathcal{S}_r, \Lambda \vdash e \xrightarrow{e} v \qquad v \in_c t}{\Delta, \mathcal{S}_r, \mathcal{S}_w, \Lambda \vdash id \Leftarrow e \xrightarrow{ss} \mathcal{S}_w(id) \leftarrow v, \Lambda} \begin{array}{l} id \in S(\Delta) \cup O(\Delta) \\ \Delta(id) = \lfloor t \rfloor \end{array}$$

IdxSigAssign

$$\frac{\begin{array}{ll} \Delta, \mathcal{S}_r, \Lambda \vdash e_i \xrightarrow{e} n_i & v \in_c t \\ \Delta, \mathcal{S}_r, \Lambda \vdash e \xrightarrow{e} v & n_i \in_c \texttt{nat}(n, m) \end{array}}{\Delta, \mathcal{S}_r, \mathcal{S}_w, \Lambda \vdash id(e_i) \Leftarrow e \xrightarrow{ss} \mathcal{S}_w(id) \leftarrow a', \Lambda} \begin{array}{l} id \in S(\Delta) \cup O(\Delta) \\ \Delta(id) = \lfloor \texttt{array}(t, n, m) \rfloor \\ \mathcal{S}_w(id) = \lfloor a \rfloor \\ i = n_i - n \\ a[i] \leftarrow v = \lfloor a' \rfloor \end{array}$$

VarAssign

$$\frac{\Delta, \mathcal{S}_r, \Lambda \vdash e \xrightarrow{e} v \qquad v \in_c t}{\Delta, \mathcal{S}_r, \mathcal{S}_w, \Lambda \vdash id := e \xrightarrow{ss} \mathcal{S}_w, \Lambda(id) \leftarrow (t, v)} \ \Lambda(id) = \lfloor (t, val) \rfloor$$

IdxVarAssign

$$\frac{\begin{array}{ll} \Delta, \mathcal{S}_r, \Lambda \vdash e_i \xrightarrow{e} n_i & n_i \in_c \texttt{nat}(n, m) \\ \Delta, \mathcal{S}_r, \Lambda \vdash e \xrightarrow{e} v & v \in_c t \end{array}}{\Delta, \mathcal{S}_r, \mathcal{S}_w, \Lambda \vdash id(e_i) := e \xrightarrow{ss} \mathcal{S}_w, \Lambda(id) \leftarrow (t, a')} \begin{array}{l} \Lambda(id) = \lfloor (\texttt{array}(t, n, m), a) \rfloor \\ i = n_i - n \\ a[i] \leftarrow v = \lfloor a' \rfloor \end{array}$$

IfElse⊤

$$\frac{\Delta, \mathcal{S}_r, \Lambda \vdash e \xrightarrow{e} \texttt{true} \qquad \Delta, \mathcal{S}_r, \mathcal{S}_w, \Lambda \vdash ss_1 \xrightarrow{ss} \mathcal{S}'_w, \Lambda'}{\Delta, \mathcal{S}_r, \mathcal{S}_w, \Lambda \vdash \ \texttt{if}(e) \ \{ss_1\} \ \texttt{else} \ \{ss_2\} \xrightarrow{ss} \mathcal{S}'_w, \Lambda'}$$

IfElse⊥

$$\frac{\Delta, \mathcal{S}_r, \Lambda \vdash e \xrightarrow{e} \texttt{false} \qquad \Delta, \mathcal{S}_r, \mathcal{S}_w, \Lambda \vdash ss_2 \xrightarrow{ss} \mathcal{S}'_w, \Lambda'}{\Delta, \mathcal{S}_r, \mathcal{S}_w, \Lambda \vdash \ \texttt{if} \ (e) \ \{ss_1\} \ \texttt{else} \ \{ss_2\} \xrightarrow{ss} \mathcal{S}'_w, \Lambda'}$$

LOOP⊥

$$\frac{\begin{array}{l}\Delta, \mathcal{S}_r, \Lambda \vdash e_2 > 0 \xrightarrow{e} \texttt{true} \\ \quad \Delta, \mathcal{S}_r, \Lambda \vdash e_1 \xrightarrow{e} v_1 \qquad \Delta, \mathcal{S}_r, \mathcal{S}_w, \Lambda(id) \leftarrow \big(\texttt{nat}(0, \texttt{NATMAX}), v_1\big) \vdash ss \xrightarrow{ss} \mathcal{S}'_w, \Lambda' \\ v_1 \in_c \texttt{nat}(0, \texttt{NATMAX}) \qquad \Delta, \mathcal{S}_r, \mathcal{S}'_w, \Lambda' \vdash \texttt{ for } (id, e_1 + 1, e_2 - 1)\ \{ss\} \xrightarrow{ss} \mathcal{S}''_w, \Lambda''\end{array}}{\Delta, \mathcal{S}_r, \mathcal{S}_w, \Lambda \vdash \texttt{ for } (id, e_1, e_2)\ \{ss\} \xrightarrow{ss} \mathcal{S}''_w, \Lambda''}$$

LOOP⊤

$$\frac{\Delta, \mathcal{S}_r, \Lambda \vdash e_2 = 0 \xrightarrow{e} \texttt{true}}{\Delta, \mathcal{S}_r, \mathcal{S}_w, \Lambda \vdash \texttt{ for } (id, e_1, e_2)\ \{ss\} \xrightarrow{ss} \mathcal{S}_w, \Lambda \setminus \{id\}}$$

RISINGEDGEDEFAULT

$$\frac{}{\Delta, \mathcal{S}_r, \mathcal{S}_w, \Lambda \vdash \texttt{rising } \{ss\} \xrightarrow{ss_f} \mathcal{S}_w, \Lambda} \begin{array}{l} f \neq \uparrow \\ f \in \{\downarrow, i, c\}\end{array}$$

FALLINGEDGEDEFAULT

$$\frac{}{\Delta, \mathcal{S}_r, \mathcal{S}_w, \Lambda \vdash \texttt{falling } \{ss\} \xrightarrow{ss_f} \mathcal{S}_w, \Lambda} \begin{array}{l} f \neq \downarrow \\ f \in \{\uparrow, i, c\}\end{array}$$

RISINGEDGEEXEC

$$\frac{\Delta, \mathcal{S}_r, \mathcal{S}_w, \Lambda \vdash ss \xrightarrow{ss_\uparrow} \mathcal{S}'_w, \Lambda'}{\Delta, \mathcal{S}_r, \mathcal{S}_w, \Lambda \vdash \texttt{rising } \{ss\} \xrightarrow{ss_\uparrow} \mathcal{S}'_w, \Lambda'}$$

FALLINGEDGEEXEC

$$\frac{\Delta, \mathcal{S}_r, \mathcal{S}_w, \Lambda \vdash ss \xrightarrow{ss_\downarrow} \mathcal{S}'_w, \Lambda'}{\Delta, \mathcal{S}_r, \mathcal{S}_w, \Lambda \vdash \texttt{falling } \{ss\} \xrightarrow{ss_\downarrow} \mathcal{S}'_w, \Lambda'}$$

RSTDEFAULT

$$\frac{\Delta, \mathcal{S}_r, \mathcal{S}_w, \Lambda \vdash ss_2 \xrightarrow{ss_f} \mathcal{S}'_w, \Lambda'}{\Delta, \mathcal{S}_r, \mathcal{S}_w, \Lambda \vdash \texttt{rst } \{ss_1\} \texttt{ else } \{ss_2\} \xrightarrow{ss_f} \mathcal{S}'_w, \Lambda'} \begin{array}{l} f \neq i \\ f \in \{\uparrow, \downarrow, c\}\end{array}$$

RSTEXEC

$$\frac{\Delta, \mathcal{S}_r, \mathcal{S}_w, \Lambda \vdash ss_1 \xrightarrow{ss_i} \mathcal{S}'_w, \Lambda'}{\Delta, \mathcal{S}_r, \mathcal{S}_w, \Lambda \vdash \texttt{rst } \{ss_1\} \texttt{ else } \{ss_2\} \xrightarrow{ss_i} \mathcal{S}'_w, \Lambda'}$$

SEQ

$$\frac{\Delta, \mathcal{S}_r, \mathcal{S}_w, \Lambda \vdash ss_1 \xrightarrow{ss} \mathcal{S}'_w, \Lambda' \qquad \Delta, \mathcal{S}_r, \mathcal{S}'_w, \Lambda' \vdash ss_2 \xrightarrow{ss} \mathcal{S}''_w, \Lambda''}{\Delta, \mathcal{S}_r, \mathcal{S}_w, \Lambda \vdash ss_1;\ ss_2 \xrightarrow{ss} \mathcal{S}''_w, \Lambda''}$$

NULL

$$\frac{}{\Delta, \mathcal{S}_r, \mathcal{S}_w, \Lambda \vdash \texttt{null} \xrightarrow{ss} \mathcal{S}_w, \Lambda}$$

4.3 Evaluation of input and output port maps

4.3.1 Evaluation of an input port map

MIPSIMPLE

$$\frac{\Delta, \mathcal{S} \vdash e \xrightarrow{e} v \qquad v \in_c t}{\Delta, \Delta_c, \mathcal{S}, \mathcal{S}_c \vdash (id \Rightarrow e) \xrightarrow{mip} \mathcal{S}_c(id) \leftarrow v} \begin{array}{l} id \in I(\Delta_c) \\ \Delta_c(id) = \lfloor t \rfloor\end{array}$$

MIPPARTIAL

$$\frac{\begin{array}{l}\Delta, \mathcal{S} \vdash e \xrightarrow{e} v \qquad v \in_c t \\ \quad e_i \xrightarrow{e} n_i \qquad n_i \in_c \texttt{nat}(n, m)\end{array}}{\Delta, \Delta_c, \mathcal{S}, \mathcal{S}_c \vdash (id(e_i) \Rightarrow e) \xrightarrow{mip} \mathcal{S}_c(id) \leftarrow a'} \begin{array}{l} id \in I(\Delta_c) \\ \Delta_c(id) = \lfloor \texttt{array}(t, n, m) \rfloor \\ i = n_i - n \\ \mathcal{S}_c(id) = \lfloor a \rfloor \\ a[i] \leftarrow v = \lfloor a' \rfloor\end{array}$$

MIPCOMP

$$\frac{\Delta, \Delta_c, \mathcal{S}, \mathcal{S}_c \vdash (name \Rightarrow e) \xrightarrow{mip} \mathcal{S}'_c \qquad \Delta, \Delta_c, \mathcal{S}, \mathcal{S}'_c \vdash i \xrightarrow{mip} \mathcal{S}''_c}{\Delta, \Delta_c, \mathcal{S}, \mathcal{S}_c \vdash (name \Rightarrow e), i \xrightarrow{mip} \mathcal{S}''_c}$$

4.3.2 Evaluation of an output port map

$$\textsc{MopOpen}\ \frac{}{\Delta, \Delta_c, \mathcal{S}, \mathcal{S}_c \vdash (id \Rightarrow \texttt{open}) \xrightarrow{mop} \mathcal{S}}$$

$$\textsc{MopSimple}\ \frac{\Delta_c, \mathcal{S}_c \vdash name \xrightarrow{e_o} v \qquad v \in_c t}{\Delta, \Delta_c, \mathcal{S}, \mathcal{S}_c \vdash (name \Rightarrow id) \xrightarrow{mop} \mathcal{S}(id) \leftarrow v}\ \begin{array}{l} id \in S(\Delta) \cup O(\Delta) \\ \Delta(id) = \lfloor t \rfloor \end{array}$$

$$\textsc{MopPartial}\ \frac{\begin{array}{ll} e_i \xrightarrow{e} n_i & v \in_c t \\ \Delta_c, \mathcal{S}_c \vdash name \xrightarrow{e_o} v & n_i \in_c \texttt{nat}(n, m) \end{array}}{\Delta, \Delta_c, \mathcal{S}, \mathcal{S}_c \vdash (name \Rightarrow id(e_i)) \xrightarrow{mop} \mathcal{S}(id) \leftarrow a'}\ \begin{array}{l} id \in S(\Delta) \cup O(\Delta) \\ \Delta(id) = \lfloor \texttt{array}(t, n, m) \rfloor \\ i = n_i - n \\ \mathcal{S}(id) = \lfloor a \rfloor \\ a[i] \leftarrow v = \lfloor a' \rfloor \end{array}$$

$$\textsc{MopComp}\ \frac{\Delta, \Delta_c, \mathcal{S}, \mathcal{S}_c \vdash assoc_{po} \xrightarrow{mop} \mathcal{S}' \qquad \Delta, \Delta_c, \mathcal{S}', \mathcal{S}_c \vdash o \xrightarrow{mop} \mathcal{S}''}{\Delta, \Delta_c, \mathcal{S}, \mathcal{S}_c \vdash assoc_{po}, o \xrightarrow{mop} \mathcal{S}''}$$

where $assoc_{po} ::= (id \Rightarrow (name | \texttt{open})) \big| (id(e) \Rightarrow name)$.

4.4 Evaluation of concurrent statements

```
1 Definition merge(σ, σ′, σ″) :=
2     let σ = (S, C) in
3     let σ′ = (S′, C′) in
4     let σ″ = (S″, C″) in
5     let S_m(id) = { S′(id)  if S′(id) ≠ S(id)
                    { S″(id)  if S″(id) ≠ S(id) in
                    { S(id)   otherwise
6     let C_m(id) = { C′(id)  if C′(id) ≠ C(id)
                    { C″(id)  if C″(id) ≠ C(id) in
                    { C(id)   otherwise
7     (S_m, C_m).
```

Listing 1: The `merge` function that fuses together an origin state σ, with two states σ' and σ'' generated by the execution of two $\mathcal{H}$-VHDL concurrent statements.

The `merge` function is correct if the three input design states assume the same domains in their signal store and design instance store, and also if there are no multiply driven signal, i.e. a signal that would have a value that is different from the original signal store $\mathcal{S}$ in both signal stores $\mathcal{S}'$ and $\mathcal{S}''$. In practice, the case where a design instance identifier could be associated with a different state in both design instance store $\mathcal{C}'$ and $\mathcal{C}''$ is not possible. Such a case could only arise if two design instances with the same identifier exist in a design's behavior, and such a design can never be elaborated.

PS

$$\frac{\Delta, \mathcal{S}, \mathcal{S}, \Lambda \vdash ss \xrightarrow{ss_f} \mathcal{S}', \Lambda'}{\mathcal{D}, \Delta, \sigma \vdash \texttt{ps}(id, vars, ss) \xrightarrow{cs_f} (\mathcal{S}', \mathcal{C})} \begin{array}{l}\Delta(id) = \lfloor \Lambda \rfloor \\ \sigma = (\mathcal{S}, \mathcal{C})\end{array}$$

COMP

$$\frac{\begin{array}{c}\Delta, \Delta_c, \mathcal{S}, \mathcal{S}_c \vdash i \xrightarrow{mip} \mathcal{S}'_c \\ \mathcal{D}, \Delta_c, (\mathcal{S}'_c, \mathcal{C}_c) \vdash d.beh \xrightarrow{cs_f} \sigma''_c \\ \Delta, \Delta_c, \mathcal{S}, \mathcal{S}''_c \vdash o \xrightarrow{mop} \mathcal{S}'\end{array}}{\mathcal{D}, \Delta, \sigma \vdash \texttt{comp}(id_c, id_e, g, i, o) \xrightarrow{cs_f} (\mathcal{S}', \mathcal{C}(id_c) \leftarrow \sigma''_c)} \begin{array}{l}\mathcal{D}(id_e) = \lfloor d \rfloor \\ \Delta(id_c) = \lfloor \Delta_c \rfloor \\ \sigma(id_c) = \lfloor \sigma_c \rfloor \\ \sigma = (\mathcal{S}, \mathcal{C}) \\ \sigma_c = (\mathcal{S}_c, \mathcal{C}_c) \\ \sigma''_c = (\mathcal{S}''_c, \mathcal{C}''_c)\end{array}$$

PAR

$$\frac{\mathcal{D}, \Delta, \sigma \vdash cs \xrightarrow{cs_f} \sigma' \quad \mathcal{D}, \Delta, \sigma \vdash cs' \xrightarrow{cs_f} \sigma''}{\mathcal{D}, \Delta, \sigma \vdash cs \mid\mid cs' \xrightarrow{cs_f} \texttt{merge}(\sigma, \sigma', \sigma'')}$$

NULL

$$\frac{}{\Delta, \sigma \vdash \texttt{null} \xrightarrow{cs_f} \sigma}$$

where $f \in \{i, \uparrow, \downarrow, c\}$. The i flag (resp. $\uparrow, \downarrow, c$) stands for the evaluation of concurrent statements during the initialization phase (resp. rising edge, falling edge and stabilization phases).

We choose to remove from the side condition of the rule PAR that was stating the absence of multiply-driven signal, and of twin design instances. The two can be obtain after a successful elaboration phase.

No multiply-driven signal

$$\texttt{nmds}(\mathcal{S}, \mathcal{S}', \mathcal{S}'') \equiv \mathsf{dom}(\mathcal{S}) = \mathsf{dom}(\mathcal{S}') = \mathsf{dom}(S)'' \wedge \forall id_s,\ \mathcal{S}(id_s) = \mathcal{S}'(id_s) \vee \mathcal{S}(id_s) = \mathcal{S}''(id_s)$$

where $\mathcal{S}, \mathcal{S}', \mathcal{S}'' \in id \nrightarrow v$, i.e. three signal stores.

State equality relation

$$\sigma \overset{\Sigma}{=} \sigma' \equiv \big(\forall id_s,\ \mathcal{S}(\sigma)(id_s) = \mathcal{S}(\sigma')(id_s)\big) \wedge \big(\forall id_c,\ \mathcal{C}(\sigma)(id_c) \overset{\Sigma}{=} \mathcal{C}(\sigma')(id_c)\big)$$

No twin design instance

$$\texttt{ntdi}(\mathcal{C}, \mathcal{C}', \mathcal{C}'') \equiv \mathsf{dom}(\mathcal{C}) = \mathsf{dom}(\mathcal{C}') = \mathsf{dom}(C)'' \wedge \forall id_c,\ \mathcal{C}(id_c) \overset{\Sigma}{=} \mathcal{C}'(id_c) \vee \mathcal{C}(id_c) \overset{\Sigma}{=} \mathcal{C}''(id_c)$$

where $\mathcal{C}, \mathcal{C}', \mathcal{C}'' \in id \nrightarrow \Sigma$, i.e. three design instance stores.

4.5 Simulation phases

4.5.1 Stabilization

STABILIZEEND

$$\frac{\mathcal{D}, \Delta, \sigma \vdash cs \xrightarrow{cs_c} \sigma'}{\mathcal{D}, \Delta, \sigma \vdash cs \overset{\rightsquigarrow}{\longrightarrow} \sigma'}\ \sigma \overset{\Sigma}{=} \sigma'$$

STABILIZELOOP

$$\frac{\mathcal{D}, \Delta, \sigma \vdash cs \xrightarrow{cs_c} \sigma' \quad \mathcal{D}, \Delta, \sigma' \vdash cs \overset{\rightsquigarrow}{\longrightarrow} \sigma''}{\mathcal{D}, \Delta, \sigma \vdash cs \overset{\rightsquigarrow}{\longrightarrow} \sigma''}\ \sigma \overset{\Sigma}{\neq} \sigma'$$

4.5.2 Initialization

$$\textsc{Init}\quad \frac{\mathcal{D}, \Delta, \sigma \vdash \text{cs} \xrightarrow{cs_i} \sigma' \qquad \mathcal{D}, \Delta, \sigma' \vdash \text{cs} \xrightarrow{\rightsquigarrow} \sigma_0}{\mathcal{D}, \Delta, \sigma \vdash \text{cs} \xrightarrow{init} \sigma_0}$$

4.5.3 Simulation cycle

Definition 3 (Input port values update)**.** *Given a simulation environment $E_p \in \mathbb{N} \to (id \nrightarrow v)$, let us define the function that update of the value of signals at a given design state $\sigma \in \Sigma$ and clock cycle count $\tau \in \mathbb{N}$. The function is defined as follows:* $\texttt{inj}(\sigma, E_p, \tau) = (\mathcal{S} \overleftarrow{\cup} E_p(\tau), \mathcal{C})$ *where $\sigma = (\mathcal{S}, \mathcal{C})$, and for all sets X and Y, and for all partial function $f, f' \in X \nrightarrow Y$,* $f \overleftarrow{\cup} f'(x) = \begin{cases} f'(x) & \text{if } x \in \texttt{dom}(f') \\ f(x) & \text{otherwise} \end{cases}$.

$$\textsc{Cycle}\quad \frac{\mathcal{D}, \Delta, \texttt{inj}(\sigma, E_p, \tau) \vdash cs \xrightarrow{cs_\uparrow} \sigma_\uparrow \qquad \mathcal{D}, \Delta, \sigma_\uparrow \vdash cs \xrightarrow{\rightsquigarrow} \sigma' \qquad \mathcal{D}, \Delta, \sigma' \vdash cs \xrightarrow{cs_\downarrow} \sigma_\downarrow \qquad \mathcal{D}, \Delta, \sigma_\downarrow \vdash cs \xrightarrow{\rightsquigarrow} \sigma''}{\mathcal{D}, E_p, \Delta, \tau, \sigma \vdash cs \xrightarrow{\uparrow,\downarrow} \sigma', \sigma''}$$

4.5.4 Simulation loop

$$\textsc{SimEnd}\quad \frac{}{\mathcal{D}, E_p, \Delta, 0, \sigma \vdash cs \to [\,]} \qquad \textsc{SimLoop}\quad \frac{\mathcal{D}, E_p, \Delta, \tau, \sigma \vdash cs \xrightarrow{\uparrow,\downarrow} \sigma', \sigma'' \qquad \mathcal{D}, E_p, \Delta, \tau - 1, \sigma'' \vdash cs \to \theta}{\mathcal{D}, E_p, \Delta, \tau, \sigma \vdash cs \to (\sigma' :: \sigma'' :: \theta)}\ \tau > 0$$

4.6 Full simulation

$$\textsc{FullSim}\quad \frac{\mathcal{D}, \mathcal{M}_g \vdash d \xrightarrow{elab} \Delta, \sigma \qquad \mathcal{D}, \Delta, \sigma \vdash d.cs \xrightarrow{init} \sigma_0 \qquad \mathcal{D}, E_p, \Delta, \tau, \sigma_0 \vdash d.beh \to \theta}{\mathcal{D}, \mathcal{M}_g, E_p, \tau \vdash d \xrightarrow{full} (\sigma_0 :: \theta)}\ \forall \tau, \mathsf{dom}(E_p(\tau)) \subseteq \mathsf{dom}(I(\Delta))$$

References

[1] IEEE Computer Society et al. *IEEE standard VHDL language reference manual*. English. OCLC: 49592632. New York, N.Y.: Institute of Electrical and Electronics Engineers, 2000. ISBN: 978-0-7381-1948-9 978-0-7381-1949-6. URL: `http://ieeexplore.ieee.org/lpdocs/epic03/standards.htm` (visited on 09/16/2019).