HoCL Manual - 1.0a

J. Sérot

Contents

Chapter 1

Core Abstract Syntax

This chapter gives the abstract syntax of a simplified version of the HoCL language. This syntax will be used in chapters 2 and **??** to formalize the typing and static semantics of the language. The main omissions, compared to the "full" HoCL language concern the annotations on IO ports, the description of opaque actors, list patterns and constructors, and mutually recursive definitions.

program ::=	`program` *typedecl** *valdecl** *nodedecl*$^+$ *graphdecl*$^+$	
typedecl ::=	`type` id	
nodedecl ::=	`node` id *param** *io** *io** *nodeimpl*	
param ::=	id : *tyexpr* [= *pexp*]	value only for toplevel graph parameters
io ::=	id : *tyexpr*	
nodeimpl ::=	| `actor` | `graph` *graphdefn*	
graphdecl ::=	`graph` id *param** *io** *io** *graphdefn*	name, params, ins, outs, defn
graphdefn ::=	| `gsd` *gwire** *gnode** | `gfd` *valdecl**	structural definition functional definition
gwire ::=	`wire` id : *tyexpr*	name, type
gnode ::=	`node` id : id *pexp** id* id*	name, model, params, ins, outs
valdecl ::=	`val` *recflag* *pattern* = *expr*	

pattern ::=	| var | ($pattern^*_{,}$) | () | _	 $(pat_1, \ldots, pat_n)$ unit ignore
expr ::=	| var | int | bool | () | ($expr^+_{,}$) | *expr* *pexp*$^+$ | $expr_1$ $expr_2$ | `fun` *pattern* $\rightarrow$ *expr* | `let` [`rec`] *pattern* = *expr* `in` *expr* | `if` *expr* `then` *expr* `else` *expr*	for denoting graph constructs tuples f < params > f arg
pexp ::=	| var | int | bool | *pexp* binop *pexp*	for denoting parameter values builtin binary opn (+,*,...)
tyexpr ::=	id	nullary type ctor

Chapter 2

Typing

This section gives the formal typing rules for the so-called *core* HoCL language defined in Chap. 1.

The type language is fairly standard. A type τ is either :

- a type variable α
- a constructed type $\chi\ \tau_1 \ldots \tau_n$,
- a functional type $\tau_1 \rightarrow \tau_2$,
- a product type $\tau_1 \times \ldots \times \tau_n$,

Typing occurs in the context of a *typing environment* consisting of :

- a type environment TE, recording type constructors,
- a variable environment VE, mapping identifiers to types[1]

The initial type environment TE_0 records the type of the *builtin* type constructors :

$$\mathrm{TE}_0 = \{\mathrm{int} \mapsto \mathsf{Int}, \mathrm{bool} \mapsto \mathsf{Bool}, \ldots\}$$

The initial variable environment VE_0 contains the types of the builtin primitives.

$$\mathrm{VE}_0 = \{+ : \mathsf{Int} \times \mathsf{Int} \rightarrow \mathsf{Int}, = : \mathsf{Int} \times \mathsf{Int} \rightarrow \mathsf{Bool}, \ldots\}$$

2.1 Notations

Both type and variable *environments* are viewed as partial maps (from identifiers to types and from type constructors to types resp.). If E is an environment, the domain of E is denoted by $\mathrm{dom}(E)$. The empty environment is written $\varnothing$. $[x \mapsto y]$ denotes the singleton environment mapping x to y. We note $E(x)$ the result of applying the underlying map to x (for ex. if E is $[x \mapsto y]$ then $E(x) = y$) and $E[x \mapsto y]$ the environment that maps x to y and behaves like E otherwise. $E \oplus E'$ denotes the environment obtained by adding the mappings of E' to those of E. If E and E' are not disjoints, then the mappings of E are "shadowed" by those of E'. Given two types τ and τ', we will note $\tau \cong \tau'$ if τ and τ' are equal modulo unification[2]. Finally, given two typing environments VE and VE$'$, we will note

[1] More precisely, to *type schemes* $\sigma = \forall\alpha.\ \tau$; but, for simplicity, we do not distinguish types from type schemes in this presentation, *i.e.* the instanciation of a type scheme into a type and the generalisation of a (polymorphic) type into a type scheme are left implicit in the rules given above. The corresponding definitions are completely standard.

[2] If τ and τ' are monomorphic, this is structural equality.

$\mathrm{VE} \subset \mathrm{VE}'$ *iff* $\forall x \in \mathrm{dom(VE)} \cap \mathrm{dom(VE')}$, $\mathrm{VE}(x) \cong \mathrm{VE}'(x)$, *i.e.* *iff* for each symbol occuring both in VE and VE′, the related types are equals modulo unification[3].

For convenience and readability, we will adhere to the following naming conventions throughout this chapter :

Meta-variable	**Meaning**
TE	Type environment
VE	Variable environment
ty	Type expression
τ	Type or type scheme
α	Type variable
χ	Type constructor
id	Identifier
pat	Pattern
expr	Graph expression
pexp	Param expression

Syntactical terminal symbols are written in **bold**. Non terminals in *italic*. Types values are written in serif.

2.2 Typing rules

2.2.1 Programs

$$\boxed{\mathrm{TE}, \mathrm{VE} \vdash \mathrm{Program} \Rightarrow \mathrm{VE}_v, \mathrm{VE}_n, \mathrm{VE}_g}$$

$$\frac{\begin{array}{c}\mathrm{TE}_0 \vdash \mathit{tydecls} \Rightarrow \mathrm{TE} \\ \mathrm{TE}_0 \oplus \mathrm{TE}, \mathrm{VE}_0 \vdash \mathit{valdecls} \Rightarrow \mathrm{VE}_v \\ \mathrm{TE}_0 \oplus \mathrm{TE}, \mathrm{VE}_0 \oplus \mathrm{VE}_v \vdash \mathit{nodedecls} \Rightarrow \mathrm{VE}_n \\ \mathrm{TE}_0 \oplus \mathrm{TE}, \mathrm{VE}_0 \oplus \mathrm{VE}_v \oplus \mathrm{VE}_n \vdash \mathit{graphdecls} \Rightarrow \mathrm{VE}_g\end{array}}{\mathrm{TE}_0, \mathrm{VE}_0 \vdash \textbf{program}\ \mathit{tydecls}\ \mathit{valdecls}\ \mathit{nodedecls}\ \mathit{graphdecls} \Rightarrow \mathrm{VE}_v, \mathrm{VE}_n, \mathrm{VE}_g} \quad (\textsc{Program})$$

Value, node and graph declarations are typed in en type environment augmented by the The result of the typing phase is a triplet of variable environments respectively recording the types of declared values, node models and graphs.

2.2.2 Type declarations

$$\boxed{\mathrm{TE}\ \vdash \mathrm{TyDecls} \Rightarrow \mathrm{TE}'}$$

$$\frac{\forall i.\ 1 \leq i \leq n, \quad \mathrm{TE}\ \vdash \mathit{tydecl}_i \Rightarrow \mathrm{TE}_i}{\mathrm{TE} \vdash \mathit{tydecl}_1\ \ldots\ \mathit{tydecl}_n \Rightarrow \bigoplus_{i=1}^{n} \mathrm{TE}_i} \quad (\textsc{TyDecls})$$

[3]This relation will be used to check that the types *inferred* when typing graph or node definitions match the types *declared* in their interface).

$$\boxed{\mathrm{TE}\ \vdash \mathrm{TyDecl} \Rightarrow \mathrm{TE}'}$$

$$\frac{}{\mathrm{TE}\ \vdash \textbf{type}\ \mathrm{id} \Rightarrow [\mathrm{id} \mapsto \mathsf{Id}]} \quad (\textsc{TyDecl})$$

The current version only supports *opaque* type declarations. In the previous rule, Id is the type constructor corresponding to the identifier id.

2.2.3 Value declarations

$$\boxed{\mathrm{TE}, \mathrm{VE}\ \vdash \mathrm{ValDecls} \Rightarrow \mathrm{VE}'}$$

$$\frac{\forall i.\ 1 \leq i \leq n, \quad \mathrm{TE}, \mathrm{VE}_{i-1}\ \vdash \mathit{valdecl}_i \Rightarrow \mathrm{VE}_i, \quad \mathrm{VE}_0 = \mathrm{VE}}{\mathrm{TE}, \mathrm{VE}\ \vdash \mathit{valdecl}_1\ \ldots\ \mathit{valdecl}_n \Rightarrow \mathrm{VE}_n} \quad (\textsc{ValDecls})$$

$$\boxed{\mathrm{TE}, \mathrm{VE}\ \vdash \mathrm{ValDecl} \Rightarrow \mathrm{VE}'}$$

$$\frac{\mathrm{TE}, \mathrm{VE} \vdash \mathit{expr} \Rightarrow \tau \qquad \vdash_p \mathit{pat}, \tau \Rightarrow \mathrm{VE}'}{\mathrm{TE}, \mathrm{VE} \vdash \textbf{val}\ \mathit{pat} = \mathit{expr}\ \Rightarrow \mathrm{VE}'} \quad (\textsc{ValDecl})$$

$$\frac{\mathrm{TE}, \mathrm{VE} \oplus \mathrm{VE}' \vdash \mathit{expr} \Rightarrow \tau \qquad \vdash_p \mathit{pat}, \tau \Rightarrow \mathrm{VE}'}{\mathrm{TE}, \mathrm{VE} \vdash \textbf{val rec}\ \mathit{pat} = \mathit{expr}\ \Rightarrow \mathrm{VE}'} \quad (\textsc{ValRecDecl})$$

Patterns

$$\boxed{\vdash_p \mathrm{Pattern}, \tau \Rightarrow \mathrm{VE}}$$

$$\frac{}{\vdash_p \mathrm{id}, \tau \Rightarrow [\mathrm{id} \mapsto \tau]} \quad (\textsc{PatVar})$$

$$\frac{\forall i.\ 1 \leq i \leq n, \quad \vdash_p \mathit{pat}\ _i, \tau_i \Rightarrow \mathrm{VE}_i}{\vdash_p \texttt{(}\mathit{pat}\ _1, \ldots \mathit{pat}\ _n\texttt{)}, \tau_1 \times \ldots \times \tau_n \Rightarrow \bigoplus_{i=1}^{n} \mathrm{VE}_i} \quad (\textsc{PatTuple})$$

$$\frac{}{\vdash_p (), \mathsf{Unit} \Rightarrow \varnothing} \quad (\textsc{PatUnit})$$

$$\frac{}{\vdash_p \_, \tau \Rightarrow \varnothing} \quad (\textsc{PatIgnore})$$

where

$$\vdash_p \mathit{pat}, \tau \Rightarrow \mathrm{VE}$$

means that declaring *pat* with type τ creates the variable environment VE.

Expressions

$$\boxed{\mathrm{TE}, \mathrm{VE}\ \vdash \mathrm{Expr} \Rightarrow \tau}$$

$$\frac{\mathrm{VE}(id) = \tau}{\mathrm{TE}, \mathrm{VE} \vdash \mathrm{id} \Rightarrow \tau} \quad (\text{EVar})$$

$$\frac{}{\mathrm{TE}, \mathrm{VE} \vdash \mathrm{int/bool} \Rightarrow \mathsf{Int/Bool}} \quad (\text{EConst})$$

$$\frac{}{\mathrm{TE}, \mathrm{VE} \vdash () \Rightarrow \mathsf{Unit}} \quad (\text{EUnit})$$

$$\frac{\forall i.\ 1 \leq i \leq n, \quad \mathrm{TE}, \mathrm{VE} \vdash\ \ expr_i \Rightarrow \tau_i}{\mathrm{TE}, \mathrm{VE} \vdash (expr_1,\ \ldots\ expr_n) \Rightarrow \tau_1 \times \ldots \times \tau_n} \quad (\text{ETuple})$$

$$\frac{\mathrm{TE}, \mathrm{VE} \vdash expr \Rightarrow \tau \to \tau' \to \tau'' \qquad \mathrm{TE}, \mathrm{VE} \vdash pexps \Rightarrow \tau}{\mathrm{TE}, \mathrm{VE} \vdash expr\ pexps \Rightarrow \tau' \to \tau''} \quad (\text{EPApp})$$

$$\frac{\mathrm{TE}, \mathrm{VE} \vdash expr_1 \Rightarrow \tau \to \tau' \qquad \mathrm{TE}, \mathrm{VE} \vdash expr_2 \Rightarrow \tau}{\mathrm{TE}, \mathrm{VE} \vdash expr_1\ \ expr_2 \Rightarrow \tau'} \quad (\text{EApp})$$

$$\frac{\vdash_p pat\ , \tau \Rightarrow \mathrm{VE}' \qquad \mathrm{TE}, \mathrm{VE} \oplus \mathrm{VE}' \vdash expr \Rightarrow \tau'}{\mathrm{TE}, \mathrm{VE} \vdash \mathbf{fun}\ pat\ \to expr \Rightarrow \tau \to \tau'} \quad (\text{EFun})$$

$$\frac{\vdash_p pat\ , \tau' \Rightarrow \mathrm{VE}' \qquad \mathrm{TE}, \mathrm{VE} \vdash expr_2 \Rightarrow \tau' \qquad \mathrm{VE} \oplus \mathrm{VE}' \vdash expr_1 \Rightarrow \tau}{\mathrm{TE}, \mathrm{VE} \vdash \mathbf{let}\ pat = expr_2\ \mathbf{in}\ expr_1 \Rightarrow \tau} \quad (\text{ELet})$$

$$\frac{\vdash_p pat\ , \tau' \Rightarrow \mathrm{VE}' \qquad \mathrm{TE}, \mathrm{VE} \oplus \mathrm{VE}' \vdash expr_2 \Rightarrow \tau' \qquad \mathrm{VE} \oplus \mathrm{VE}' \vdash expr_1 \Rightarrow \tau}{\mathrm{TE}, \mathrm{VE} \vdash \mathbf{let\ rec}\ pat = expr_2\ \mathbf{in}\ expr_1 \Rightarrow \tau} \quad (\text{ELetRec})$$

$$\frac{\mathrm{TE}, \mathrm{VE} \vdash expr \Rightarrow \mathsf{Bool} \qquad \mathrm{TE}, \mathrm{VE} \vdash expr_1 \Rightarrow \tau \qquad \mathrm{TE}, \mathrm{VE} \vdash expr_2 \Rightarrow \tau}{\mathrm{VE} \vdash \mathbf{if}\ \ expr\ \mathbf{then}\ expr_1\ \mathbf{else}\ expr_2 \Rightarrow \tau} \quad (\text{EIf})$$

Param expressions

$$\boxed{\mathrm{TE}, \mathrm{VE}\ \vdash \mathrm{PExps} \Rightarrow \tau}$$

$$\frac{\forall i.\ 1 \leq i \leq n, \quad \mathrm{TE}, \mathrm{VE} \vdash\ \ pexp_i \Rightarrow \tau_i}{\mathrm{TE}, \mathrm{VE} \vdash pexp_1,\ \ldots\ pexp_n \Rightarrow \tau_1 \times \ldots \times \tau_n} \quad (\text{PExps})$$

$$\boxed{\mathrm{TE}, \mathrm{VE}\ \vdash \mathrm{PExp} \Rightarrow \tau}$$

$$\frac{\mathrm{VE}(id) = \tau}{\mathrm{TE}, \mathrm{VE} \vdash \mathrm{id} \Rightarrow \tau} \quad (\textsc{PVar})$$

$$\frac{}{\mathrm{TE}, \mathrm{VE} \vdash \mathrm{int/bool} \Rightarrow \mathsf{Int/Bool}} \quad (\textsc{PConst})$$

$$\frac{\begin{array}{c}\mathrm{VE}(op) = \tau \times \tau' \to \tau'' \\ \mathrm{TE}, \mathrm{VE} \vdash \mathit{pexp}_1 \Rightarrow \tau \\ \mathrm{TE}, \mathrm{VE} \vdash \mathit{pexp}_2 \Rightarrow \tau'\end{array}}{\mathrm{TE}, \mathrm{VE} \vdash \mathit{pexp}_1 \text{ binop } \mathit{pexp}_2 \Rightarrow \tau''} \quad (\textsc{PBinop})$$

2.2.4 Node declarations

$$\boxed{\mathrm{TE}, \mathrm{VE} \ \vdash \mathrm{NodeDecls} \Rightarrow \mathrm{VE}'}$$

$$\frac{\forall i.\ 1 \le i \le n, \quad \mathrm{TE}, \mathrm{VE}_{i-1} \ \vdash \mathit{nodedecl}_i \Rightarrow \mathrm{VE}_i, \quad \mathrm{VE}_0 = \mathrm{VE}}{\mathrm{TE}, \mathrm{VE} \ \vdash \mathit{nodedecl}_1 \ \ldots \ \mathit{nodedecl}_n \Rightarrow \bigoplus_{i=1}^{n} \mathrm{VE}_i} \quad (\textsc{NodeDecls})$$

$$\boxed{\mathrm{TE}, \mathrm{VE} \ \vdash \mathrm{NodeDecl} \Rightarrow \mathrm{VE}'}$$

$$\frac{\begin{array}{c}\mathit{params} \neq \emptyset \\ \mathrm{TE} \vdash \mathit{params} \Rightarrow \tau_p, \mathrm{VE}_p \\ \mathrm{TE} \vdash \mathit{ins} \Rightarrow \tau_i, \mathrm{VE}_i \\ \mathrm{TE} \vdash \mathit{outs} \Rightarrow \tau_o, \mathrm{VE}_o \\ \mathrm{TE}, \mathrm{VE} \oplus \mathrm{VE}_p \oplus \mathrm{VE}_i \vdash \mathit{nodeimpl} \Rightarrow \mathrm{VE}'_o \\ \mathrm{VE}'_o \subset \mathrm{VE}_o\end{array}}{\mathrm{TE}, \mathrm{VE} \ \vdash \textbf{node} \text{ id } \mathit{params\ ins\ outs\ nodeimpl} \Rightarrow [\mathrm{id} \mapsto \tau_p \to \tau_i \to \tau_o]} \quad (\textsc{NodeDecl1})$$

$$\frac{\begin{array}{c}\mathit{params} = \emptyset \\ \mathrm{TE} \vdash \mathit{ins} \Rightarrow \tau_i, \mathrm{VE}_i \\ \mathrm{TE} \vdash \mathit{outs} \Rightarrow \tau_o, \mathrm{VE}_o \\ \mathrm{TE}, \mathrm{VE} \oplus \mathrm{VE}_i \vdash \mathit{nodeimpl} \Rightarrow \mathrm{VE}'_o \\ \mathrm{VE}'_o \subset \mathrm{VE}_o\end{array}}{\mathrm{TE}, \mathrm{VE} \ \vdash \textbf{node} \text{ id } \mathit{params\ ins\ outs\ nodeimpl} \Rightarrow [\mathrm{id} \mapsto \tau_i \to \tau_o]} \quad (\textsc{NodeDecl2})$$

The type assigned to a node only depends on its interface (parameters, inputs and outputs). A node `n` declared as

$$\texttt{node}\ n\ \texttt{param}(p_1 : t_1, \ldots, p_k : t_k)\ \texttt{in}\ (i_1 : t'_1, \ldots, i_m : t'_m)\ \texttt{out}\ (o_1 : t''_1, \ldots, o_n : t''_n)$$

will be assigned type

$$\mathsf{param}\ t_1 \times \ldots \times \mathsf{param}\ t_k \to \mathsf{wire}\ t'_1 \times \ldots \times \mathsf{wire}\ t'_m \to \mathsf{wire}\ t''_1 \times \ldots \times \mathsf{wire}\ t''_n$$

where param and wire are predefined type constructors, used to distinguish values denoting bound to node parameters and node IOs respectively.

Whereas a node n declared as

$$\texttt{node}\ n\ \texttt{in}\ (i_1 : t'_1, \ldots, i_m : t'_m)\ \texttt{out}\ (o_1 : t''_1, \ldots, o_n : t''_n)$$

will be assigned type

$$\mathsf{wire}\ t'_1 \times \ldots \times \mathsf{wire}\ t'_m \rightarrow \mathsf{wire}\ t''_1 \times \ldots \times \mathsf{wire}\ t''_n$$

2.2.5 Node or graph parameters

$$\boxed{\mathrm{TE}, \mathrm{VE}\ \vdash \mathrm{Params} \Rightarrow \tau, \mathrm{VE}'}$$

$$\frac{\forall i.\ 1 \leq i \leq n, \quad \mathrm{TE}, \mathrm{VE}\ \vdash \mathit{param}_i \Rightarrow \tau_i, \mathrm{VE}_i}{\mathrm{TE}, \mathrm{VE}\ \vdash \mathit{param}_1\ \ldots\ \mathit{param}_n \Rightarrow \tau_1 \times \ldots \times \tau_n,\ \bigoplus_{i=1}^{n} \mathrm{VE}_i} \quad (\textsc{Params})$$

$$\boxed{\mathrm{TE}, \mathrm{VE}\ \vdash \mathrm{Param} \Rightarrow \tau, \mathrm{VE}'}$$

$$\frac{\begin{array}{c}\mathrm{TE}\ \vdash \mathit{ty} \Rightarrow \tau \\ \tau' = \mathsf{param}\ \tau\end{array}}{\mathrm{TE}, \mathrm{VE}\ \vdash \mathrm{id}\ :\ \mathit{ty} \Rightarrow \tau',\ [\mathrm{id} \mapsto \tau']} \quad (\textsc{Param})$$

$$\frac{\begin{array}{c}\mathrm{TE}\ \vdash \mathit{ty} \Rightarrow \tau \\ \mathrm{TE}, \mathrm{VE}\ \vdash \mathit{pexp} \Rightarrow \tau' \\ \tau \cong \tau' \\ \tau'' = \mathsf{param}\ \tau\end{array}}{\mathrm{TE}, \mathrm{VE}\ \vdash \mathrm{id}\ :\ \mathit{ty}\ =\ \mathit{pexp} \Rightarrow \tau'',\ [\mathrm{id} \mapsto \tau'']} \quad (\textsc{ParamWithValue})$$

2.2.6 Node or graph IOs

$$\boxed{\mathrm{TE}\ \vdash \mathrm{Ios} \Rightarrow \tau, \mathrm{VE}'}$$

$$\frac{\forall i.\ 1 \leq i \leq n, \quad \mathrm{TE}\ \vdash \mathit{io}_i \Rightarrow \tau_i, \mathrm{VE}_i}{\mathrm{TE}\ \vdash \mathit{io}_1\ \ldots\ \mathit{io}_n \Rightarrow \tau_1 \times \ldots \times \tau_n,\ \bigoplus_{i=1}^{n} \mathrm{VE}_i} \quad (\textsc{Ios})$$

$$\boxed{\mathrm{TE}\ \vdash \mathrm{Io} \Rightarrow \tau, \mathrm{VE}'}$$

$$\frac{\begin{array}{c}\mathrm{TE} \vdash ty \Rightarrow \tau \\ \tau' = \mathsf{wire}\ \tau\end{array}}{\mathrm{TE} \vdash \mathrm{id}\ :\ ty \Rightarrow \tau',\ [\mathrm{id} \mapsto \tau']} \quad (\textsc{Io})$$

$$\boxed{\mathrm{TE}, \mathrm{VE} \vdash \mathrm{NodeImpl} \Rightarrow \mathrm{VE}'}$$

2.2.7 Node implementation

$$\frac{}{\mathrm{TE}, \mathrm{VE} \vdash \mathbf{actor} \Rightarrow \varnothing} \quad (\textsc{ActorImpl})$$

$$\frac{\mathrm{TE}, \mathrm{VE} \vdash \mathit{graphdefn} \Rightarrow \mathrm{VE}'}{\mathrm{TE}, \mathrm{VE} \vdash \mathbf{graph}\ \mathit{graphdefn} \Rightarrow \mathrm{VE}'} \quad (\textsc{GraphImpl})$$

2.2.8 Graph definitions

$$\boxed{\mathrm{TE}, \mathrm{VE} \vdash \mathrm{GraphDefn} \Rightarrow \mathrm{VE}'}$$

$$\frac{\forall i.\ 1 \le i \le n,\quad \mathrm{TE}, \mathrm{VE}_{i-1} \vdash \mathit{valdecl}_i \Rightarrow \mathrm{VE}_i,\quad \mathrm{VE}_0 = \mathrm{VE}}{\mathrm{TE}, \mathrm{VE} \vdash \mathbf{gfd}\ \mathit{valdecl}_1\ \ldots\ \mathit{valdecl}_n \Rightarrow \bigoplus_{i=1}^{n} \mathrm{VE}_i} \quad (\textsc{GraphFunDefn})$$

$$\frac{\begin{array}{c}\mathrm{TE} \vdash \mathit{wires} \Rightarrow \mathrm{VE}_w \\ \mathrm{TE}, \mathrm{VE} \oplus \mathrm{VE}_w \vdash \mathit{nodes} \Rightarrow \mathrm{VE}_o\end{array}}{\mathrm{TE}, \mathrm{VE} \vdash \mathbf{gsd}\ \mathit{wires}\ \mathit{nodes} \Rightarrow \mathrm{VE}_o} \quad (\textsc{GraphStructDefn})$$

Graph structural definitions

$$\boxed{\mathrm{TE} \vdash \mathrm{Wires} \Rightarrow \mathrm{VE}'}$$

$$\frac{\forall i.\ 1 \le i \le n,\quad \mathrm{TE} \vdash \mathit{wire}_i \Rightarrow \mathrm{VE}_i}{\mathrm{TE} \vdash \mathit{wire}_1\ \ldots\ \mathit{wire}_n \Rightarrow \bigoplus_{i=1}^{n} \mathrm{VE}_i} \quad (\textsc{Wires})$$

$$\boxed{\mathrm{TE} \vdash \mathrm{Wire} \Rightarrow \mathrm{VE}'}$$

$$\frac{\begin{array}{c}\mathrm{TE} \vdash ty \Rightarrow \tau \\ \tau' = \mathsf{wire}\ \tau\end{array}}{\mathrm{TE} \vdash \mathbf{wire}\ \mathrm{id}\ :\ ty \Rightarrow [\mathrm{id} \mapsto \tau']} \quad (\textsc{Wire})$$

$$\boxed{\mathrm{TE}, \mathrm{VE} \vdash \mathrm{Nodes} \Rightarrow \mathrm{VE}'}$$

$$\frac{\forall i.\ 1 \leq i \leq n, \quad \mathrm{TE}, \mathrm{VE} \vdash \mathit{node}_i \Rightarrow \mathrm{VE}_i}{\mathrm{TE}, \mathrm{VE} \vdash \mathit{node}_1 \ \ldots \ \mathit{node}_n \Rightarrow \bigoplus_{i=1}^{n} \mathrm{VE}_i} \quad (\textsc{Nodes})$$

$$\boxed{\mathrm{TE}, \mathrm{VE} \vdash \mathrm{Node} \Rightarrow \mathrm{VE}'}$$

$$\frac{\begin{array}{c} \mathit{pvals} \neq \emptyset \\ \mathrm{VE}(m) = \tau \to \tau' \to \tau'' \\ \mathrm{TE} \vdash \mathit{pvals} \Rightarrow \tau_p \\ \mathrm{TE} \vdash \mathit{ins} \Rightarrow \tau_i, \mathrm{VE}_i \\ \mathrm{TE} \vdash \mathit{outs} \Rightarrow \tau_o, \mathrm{VE}_o \\ \tau_p \cong \tau \qquad \tau_i \cong \tau' \qquad \tau_o \cong \tau'' \end{array}}{\mathrm{TE}, \mathrm{VE} \vdash \textbf{node} \ \mathrm{id} \ : \ \mathrm{m} \ \mathit{pvals} \ \mathit{ins} \ \mathit{outs} \Rightarrow \mathrm{VE}_o} \quad (\textsc{Node1})$$

$$\frac{\begin{array}{c} \mathit{pvals} = \emptyset \\ \mathrm{VE}(m) = \tau \to \tau' \\ \mathrm{TE} \vdash \mathit{ins} \Rightarrow \tau_i, \mathrm{VE}_i \\ \mathrm{TE} \vdash \mathit{outs} \Rightarrow \tau_o, \mathrm{VE}_o \\ \tau_i \cong \tau \qquad \tau_o \cong \tau' \end{array}}{\mathrm{TE}, \mathrm{VE} \vdash \textbf{node} \ \mathrm{id} \ : \ \mathrm{m} \ \mathit{pvals} \ \mathit{ins} \ \mathit{outs} \Rightarrow \mathrm{VE}_o} \quad (\textsc{Node2})$$

2.2.9 Graph declarations

$$\boxed{\mathrm{TE}, \mathrm{VE} \vdash \mathrm{GraphDecls} \Rightarrow \mathrm{VE}'}$$

$$\frac{\forall i.\ 1 \leq i \leq n, \quad \mathrm{TE}, \mathrm{VE}_{i-1} \vdash \mathit{graphdecl}_i \Rightarrow \mathrm{VE}_i, \quad \mathrm{VE}_0 = \mathrm{VE}}{\mathrm{TE}, \mathrm{VE} \vdash \mathit{graphdecl}_1 \ \ldots \ \mathit{graphdecl}_n \Rightarrow \bigoplus_{i=1}^{n} \mathrm{VE}_i} \quad (\textsc{GraphDecls})$$

$$\boxed{\mathrm{TE}, \mathrm{VE} \vdash \mathrm{GraphDecl} \Rightarrow \mathrm{VE}'}$$

$$\frac{\begin{array}{c} \mathit{params} \neq \emptyset \\ \mathrm{TE} \vdash \mathit{params} \Rightarrow \tau_p, \mathrm{VE}_p \\ \mathrm{TE} \vdash \mathit{ins} \Rightarrow \tau_i, \mathrm{VE}_i \\ \mathrm{TE} \vdash \mathit{outs} \Rightarrow \tau_o, \mathrm{VE}_o \\ \mathrm{TE}, \mathrm{VE} \oplus \mathrm{VE}_p \oplus \mathrm{VE}_i \vdash \mathit{graphdefn} \Rightarrow \mathrm{VE}'_o \\ \mathrm{VE}'_o \subset \mathrm{VE}_o \end{array}}{\mathrm{TE}, \mathrm{VE} \vdash \textbf{graph} \ \mathrm{id} \ \mathit{params} \ \mathit{ins} \ \mathit{outs} \ \mathit{graphdefn} \Rightarrow [\mathrm{id} \mapsto \tau_p \to \tau_i \to \tau_o]} \quad (\textsc{GraphDecl1})$$

$$\frac{\begin{array}{c} \mathit{params} = \emptyset \\ \mathrm{TE} \vdash \mathit{ins} \Rightarrow \tau_i, \mathrm{VE}_i \\ \mathrm{TE} \vdash \mathit{outs} \Rightarrow \tau_o, \mathrm{VE}_o \\ \mathrm{TE}, \mathrm{VE} \oplus \oplus \mathrm{VE}_i \vdash \mathit{graphdefn} \Rightarrow \mathrm{VE}'_o \\ \mathrm{VE}'_o \subset \mathrm{VE}_o \end{array}}{\mathrm{TE}, \mathrm{VE} \ \vdash \textbf{graph} \ \mathrm{id} \ \mathit{params} \ \mathit{ins} \ \mathit{outs} \ \mathit{graphdefn} \Rightarrow [\mathrm{id} \mapsto \tau_i \rightarrow \tau_o]} \quad (\textsc{GraphDecl2})$$

2.2.10 Type expressions

$$\boxed{\mathrm{TE} \ \vdash \mathit{ty} \Rightarrow \tau}$$

$$\frac{\mathrm{TE}(\mathrm{id}) = \tau}{\mathrm{TE} \vdash \mathrm{id} \Rightarrow \tau} \quad (\textsc{TyCon})$$

Type expressions, at the syntax level, are limited to type names.