Type and Clock Checking in *Lustre** (*Draft*)

L2C team, System Software and Software Engineering Laboratory
Department of Computer Science and Technology, Tsinghua University

October 27, 2019

1 Introduction

A type system keeps track of the types of variables and, in general, of the types of all expressions in a program. Type systems are used to determine whether programs are well behaved. A statically type checking process is performed, by a type checker, to prevent unsafe and ill behaved programs from ever running. A program that passes the type checker is said to be well typed; otherwise, it is ill typed.

The type checking in this document includes two parts of statically checking for the *Lustre** language, the source language in the L2C project. One is for the conventional type checking, and the other is for the clock checking specific to a Lustre-like synchronous data-flow language. The abstract syntax of *Lustre**, produced by parsing, is specified in the next section. The type system for the conventional type checking and the clock checking, defined on this abstract syntax, are presented in Section 3 and Section 4 respectively.

2 Abstract Syntax of *Lustre**

2.1 Program

$\langle program \rangle$ ::= { $\langle one\_decl \rangle$ }

$\langle one\_decl \rangle$ ::= $\langle type\_decl \rangle$
| $\langle const\_decl \rangle$
| $\langle node\_decl \rangle$

2.2 Type Declaration

$\langle type\_decl \rangle$ ::= ($IDENT$, $\langle kind \rangle$)

$\langle kind \rangle$ ::= $IDENT$ //type identifiers
| $Short$ (* integer types, signed 16 bits *)
| $UShort$ (* integer types, unsigned 16 bits *)
| Int (* integer types, signed 32 bits *)
| $UInt$ (* integer types, unsigned 32 bits *)
| $Float$ (* floating-point types, 32 bits *)
| $Real$ (* floating-point types, 64 bits *)
| $Bool$ (* bool types *)
| $Char$ (* char types *)
| $Array(\langle kind \rangle, IntConst(INTEGER))$
| $Struct(\langle fieldlist \rangle)$
| $Enum(\langle identlist \rangle)$

$\langle fieldlist \rangle$ ::= $(IDENT, \langle kind \rangle)^*$

$\langle identlist \rangle$ ::= $(IDENT)^*$

2.3 Const Declaration

$\langle const\_decl \rangle$ ::= $(IDENT, \langle kind \rangle, \langle const\_expr \rangle)$

$\langle const\_expr \rangle$ ::= $\langle atom\_expr \rangle$
| $Eunop(\langle unary\_operation \rangle, \langle const\_expr \rangle)$
| $Ebinop(\langle binary\_operation \rangle, \langle const\_expr \rangle, \langle const\_expr \rangle)$
| $ArrayConst(\langle const\_list \rangle)$ (* array constants *)
| $ConstructConst(\langle field\_const\_list \rangle)$ (* struct constants *)
| $EnumConst(IDENT, \langle kind \rangle)$ (* enum constants *)

$\langle atom\_expr \rangle$ ::= $IDENT$ //to define other constants by a constant identifier
| $\langle bool \rangle$ (* bool constants *)
| $ShortConst(INTEGER)$ (* integer constants, signed 16 bits *)
| $UshortConst(INTEGER)$ (* integer constants, unsigned 16 bits *)
| $IntConst(INTEGER)$ (* integer constants, signed 32 bits *)
| $UintConst(INTEGER)$ (* integer constants, unsigned 32 bits *)
| $CharConst(INTEGER)$ (* char constants *)
| $FloatConst(FLOAT)$ (* floating-point constants, 32 bits *)
| $RealConst(REAL)$ (* floating-point constants, 64 bits *)
| $BoolConst(\langle bool \rangle)$ (* bool constants *)

$\langle bool \rangle$ ::= $TRUE \mid FALSE$

$\langle const\_list \rangle$ ::= $\langle const\_expr \rangle^*$

$\langle field\_const \rangle$ ::= $(IDENT, \langle const\_expr \rangle)$

$\langle field\_const\_list \rangle$ ::= $\langle field\_const \rangle^*$

2.4 Node/Function Declaration

$\langle node\_decl \rangle$ ::= $(\langle funcType \rangle, \langle modifier \rangle, IDENT, \langle params \rangle, \langle returns \rangle, \langle body \rangle)$
//node or function declareation

$\langle funcType \rangle$ ::= node | function

$\langle modifier \rangle$ ::= main | notMain

$\langle params \rangle$ ::= $\langle var\_decl \rangle^*$

$\langle returns \rangle$ ::= $\langle var\_decl \rangle^*$

$\langle var\_decl \rangle$ ::= $VDecl(IDENT, \langle kind \rangle)$
| $WhenVDecl(IDENT, \langle kind \rangle, \langle ck \rangle)$

$\langle ck \rangle$ ::= $(TRUE, IDENT)$ | $(FALSE, IDENT)$ (* $IDENT$ is a bool identifier *)
| $(IDENT, IDENT)$ (* right $IDENT$: an enum var, left $IDENT$: a value *) [1]

$\langle body \rangle$ ::= $(\langle locals \rangle, \langle equation\_list \rangle)$

$\langle locals \rangle$ ::= $\langle var\_decl \rangle^*$

$\langle equation\_list \rangle$ ::= $\langle equation \rangle^*$

$\langle equation \rangle$ ::= $Equation(\langle lhs \rangle, \langle expr \rangle)$

$\langle lhs \rangle$ ::= $\langle lhs\_id \rangle^*$

$\langle lhs\_id \rangle$ ::= $IDENT$

$\langle expr \rangle$ ::= $Econst(\langle atom\_expr \rangle)$
| $Evar(IDENT)$
| $ListExpr(\langle expr\_list \rangle)$ (* list expression *)
| $ApplyExpr(\langle operator \rangle, \langle expr\_list \rangle)$
(* operator application *)
| $Econstruct(\langle field\_expr\_list \rangle)$
(* construct a struct, e.g. label1 : 3, label2 : false *)
| $Earrayacc(\langle expr \rangle, \langle expr \rangle)$
(* expr[i], access to (i+1)th member of an array `expr` *)
| $Earraydef(\langle expr \rangle, INTEGER)$
(* expr ˆ i, an array of size `i` with every element `expr` *)
| $Earraydiff(\langle expr\_list \rangle)$
(* build an array with elements in the list expression, e.g. [1,2] *)
| $Earrayproj(\langle expr \rangle, \langle expr\_list \rangle, \langle expr \rangle)$
(* dynamic projection,
e.g. (2ˆ 3ˆ 4.[7] default 100ˆ 3), value is 100ˆ 3 *)
| $Earrayslice(\langle expr \rangle, \langle expr \rangle, \langle expr \rangle)$
(* a[i..j] is the sliced array [a_i, a_i+1, ... , a_j] *)

[1] A text in red color shows some candidate consideration in the future.

$\mid$ $Emix(\langle expr\rangle, \langle label\_index\_list\rangle, \langle expr\rangle)$
(* construct a new array or struct,
e.g. label1:2ˆ 3 with label1.[0] = 7, value is label1:[7,2] *)
$\mid$ $Eunop(\langle unary\_operation\rangle, \langle expr\rangle)$ (* unary operation *)
$\mid$ $Ebinop(\langle binary\_operation\rangle, \langle expr\rangle, \langle expr\rangle)$ (* binary operation *)
$\mid$ $Efield(\langle expr\rangle, IDENT)$
(* access to a member of a struct *)
$\mid$ $Epre(\langle expr\rangle)$
(* pre : shift flows on the last instant backward,
producing an undefined value at the first instant *)
$\mid$ $Efby(\langle expr\rangle, INTEGER, \langle expr\rangle)$
(* fby : fby(b; n; a) = a $\rightarrow$ pre fby(b; n-1; a) *)
$\mid$ $Earrow(\langle expr\rangle, \langle expr\rangle)$
(* $\rightarrow$: set the initial values of flows *)
$\mid$ $Ewhen(\langle expr\rangle, \langle ck\rangle)$
(* x when h: if h does not hold, then no value; otherwise x *)
$\mid$ $Ecurrent(\langle expr\rangle)$ (* current expressions *)
$\mid$ $Emerge(IDENT, \langle expr\rangle, \langle expr\rangle)$ (* merge expressions *)
$\mid$ $Emerge(IDENT, \langle merge\_case\_list\rangle)$
(* merge expressions *)
$\mid$ $Eif(\langle expr\rangle, \langle expr\rangle, \langle expr\rangle)$ (* conditional expressions *)
$\mid$ $Ecase(\langle expr\rangle, \langle pattern\_list\rangle)$ (* case expressions *)
$\mid$ $Eboolred(INTEGER, INTEGER, \langle expr\rangle)$
(* boolred expressions *)
$\mid$ $Ediese(\langle expr\rangle)$
(* #(a1, ..., an) $\rightarrow$ boolred(0,1,n)[a1, ..., an] *)
$\mid$ $EnorS(\langle expr\rangle)$
(* nor(a1, ..., an) boolred(0,0,n)[a1, ..., an] *)

$\langle expr\_list\rangle ::= \langle expr\rangle^*$

$\langle field\_expr\_list\rangle ::= (IDENT, \langle expr\rangle)^*$

$\langle label\_index\_list\rangle ::= (IDENT \mid \langle expr\rangle)^*$

$\langle pattern\_list\rangle ::= (\langle pattern\rangle, \langle expr\rangle)^*$

$\langle operator\rangle ::= \langle prefix\rangle \mid Iterator(\langle iterator\_operation\rangle, \langle prefix\rangle, INTEGER)$

$\langle prefix\rangle ::= IDENT$
$\mid \langle prefix\_op\rangle$

$\langle prefix\_op\rangle ::= \langle unary\_operation\rangle$
$\mid \langle binary\_operation\rangle$

$\langle pattern\rangle ::= PatternID(IDENT, \langle kind\rangle)$ (* pattern identifier *)
$\mid PatternChar(INTEGER)$ (* pattern char *)
$\mid PatternInt(INTEGER)$ (* pattern integer *)
$\mid PatternBool(\langle bool\rangle)$ (* pattern bool *)
$\mid PatternAny$ (* pattern any *)

$\langle \mathit{merge\_case\_list} \rangle ::= (\langle \mathit{merge\_head} \rangle, \langle \mathit{expr} \rangle)^*$ //merge case list

$\langle \mathit{merge\_head} \rangle ::=$ *IDENT* //enum identifiers
| *TRUE* //merge bool
| *FALSE* //merge bool

$\langle \mathit{iterator\_operation} \rangle ::=$ *Omap* (* higher-order operator *map* *)
| *Ored* (* higher-order operator *red* *)
| *Ofill* (* higher-order operator *fill* *)
| *Ofillred* (* higher-order operator *fillred* *)

$\langle \mathit{unary\_operation} \rangle ::=$ *Oshort* (* convert to short(signed 16 bits) ([short]) *)
| *Oint* (* convert to int(signed 32 bits) ([int]) *)
| *Ofloat* (* convert to float(32 bits) ([float]) *)
| *Oreal* (* convert to real(64 bits) ([real]) *)
| *Onot* (* boolean negation ([not]) *)
| *Opos* (* positive (unary [+]) *)
| *Oneg* (* opposite (unary [-]) *)

$\langle \mathit{binary\_operation} \rangle ::=$ *Oadd* (* addition (binary [+]) *)
| *Osub* (* subtraction (binary [-]) *)
| *Omul* (* multiplication (binary [*]) *)
| *Odivreal* (* division real ([/]) *)
| *Odivint* (* division integer ([div]) *)
| *Omod* (* remainder ([mod]) *)
| *Oand* (* and ([and]) *)
| *Oor* (* or ([or]) *)
| *Oxor* (* xor ([xor]) *)
| *Oeq* (* comparison ([=]) *)
| *One* (* comparison ([<>']) *)
| *Olt* (* comparison ([<]) *)
| *Ogt* (* comparison ([>]) *)
| *Ole* (* comparison ([<=]) *)
| *Oge* (* comparison ([>=]) *)

3 Type System for type checking

3.1 Type expressions

3.1.1 Basic types

A basic type expression can be defined by the syntax shown as follows.

<type>	::=	*<scalar_type>*	scalar types
	$\mid$	*<array_type>*	array types
	$\mid$	*<struct_type>*	struct types
	$\mid$	*<enum_type>*	enum types
	$\mid$	*void*	any type
<scalar_type>	::=	*ushort*	integer types, unsigned 16 bits
	$\mid$	*short*	integer types, signed 16 bits
	$\mid$	*int*	integer types, signed 32 bits
	$\mid$	*uint*	integer types, unsigned 32 bits
	$\mid$	*float*	floating-point types, 32 bits
	$\mid$	*real*	floating-point types, 64 bits
	$\mid$	*bool*	bool types
	$\mid$	*char*	char types
<array_type>	::=	*array*(<*type*>, *int*)	array type
<struct_type>	::=	*struct*(<*field_type*> {, <*field_type*> })	struct type
<field_type>	::=	(<ident>, <*type*>)	
<enum_type>	::=	*enum*(<ident> {, <ident> })	enum type

where the <ident> shares the definition in Section 2.5.

3.1.2 Compound types

We introduce two kinds of compound types.
A product type expression $A_1 \times A_2 \times ... \times A_n$ is the type of tuples of values with the first component of type expression A_1, the second component of type expression A_2, ..., and the nth component of type expression A_n. It is possible that $n = 0$, where it means a *nil* list of type. We often use $A_1 :: A_2 :: ... :: A_n$, instead of $A_1 \times A_2 \times ... \times A_n$, in the case for a list of expressions (not a list of parameters).
A node/function type expression $A_1 \times A_2 \times ... \times A_n \rightarrow B_1 \times B_2 \times ... \times B_m$ is the type expression of nodes or functions with the input parameter list of type $A_1 \times A_2 \times ... \times A_n$, and the output parameter list of type $B_1 \times B_2 \times ... \times B_m$.

3.2 Typing environment

A typing environment associates type expressions to variables and has the form

$$\mathcal{E} ::= [\ x_1 : A_1,\ A_2 : A_2,\ ...,\ x_n : A_n\ ]$$

where $x_i \neq x_j$ for all i and j , satisfying $i \neq j$ and $(1 \leq i, j \leq n)$.

We use $\mathcal{C}$, $\mathcal{T}$ and $\mathcal{F}$ to denote a global const block typing environment, a global type block typing environment and a local function/node typing environment respectively. In some cases, we use $\mathcal{F}_{id}$ to denote a particular local typing environment specific to the context of a node/function identified by *id*.

3.3 Judgements

- $\mathcal{E} \vdash e : A$, implies that, under the the well-formed typing environment $\mathcal{E}$, the expression e is well-typed and has the type A. Here, $\mathcal{E}$ can be ϕ, $\mathcal{C}$, $\mathcal{T}$ or $\mathcal{F}$.
- $\mathcal{E} \vdash D$, implies that, under the the well-formed typing environment $\mathcal{E}$, the program component D is well-typed. Here, $\mathcal{E}$ can be ϕ, $\mathcal{C}$, $\mathcal{T}$ or $\mathcal{F}$.
- $\mathcal{E} \vdash \diamond$, means that $\mathcal{E}$ is a well-formed typing environment. Here, $\mathcal{E}$ can be ϕ, $\mathcal{C}$, $\mathcal{T}$ or $\mathcal{F}$. We also use some abbreviations to denote the well-formedness for two or more typing environments. For example, $\mathcal{C}, \mathcal{T} \vdash \diamond$, means that both $\mathcal{C}$ and $\mathcal{T}$ are well-formed typing environments respectively.
- $\mathcal{C}, \mathcal{T} \vdash e : A$, implies that, under the well-formed typing environments $\mathcal{C}$ and $\mathcal{T}$, the expression e is well-typed and has the type A.
- $\mathcal{C}, \mathcal{T} \vdash D$, implies that under the well-formed typing environments $\mathcal{C}$ and $\mathcal{T}$, the program component D is well-typed.
- $\mathcal{C}, \mathcal{T}, \mathcal{F} \vdash e : A$, implies that, under the well-formed typing environments $\mathcal{C}$, $\mathcal{T}$ and $\mathcal{F}$, the expression e is well-typed and has the type A.
- $\mathcal{C}, \mathcal{T}, \mathcal{F} \vdash D$, implies that under the well-formed typing environments $\mathcal{C}$, $\mathcal{T}$ and $\mathcal{F}$, the program component D is well-typed.

3.4 Rules used for the conventional type checking based on the abstract syntax of *Lustre**

3.4.1 Common

$$\frac{}{\phi \vdash \diamond}\ (\textsc{Env}\ \phi) \qquad\qquad \frac{\mathcal{E} \vdash \diamond \qquad x : A \in \mathcal{E}}{\mathcal{E} \vdash x : A}\ (\textsc{Env TVal})$$

$$\frac{\mathcal{E}' \vdash \diamond \qquad x \notin dom(\mathcal{E}') \qquad \mathcal{E} = \mathcal{E}' \cup \{x : A\}}{\mathcal{E} \vdash \diamond}\ (\textsc{Env WellTY})$$

$$\frac{\mathcal{E}' \vdash e : A \qquad y \notin dom(\mathcal{E}') \qquad \mathcal{E} = \mathcal{E}' \cup \{y : A'\}}{\mathcal{E} \vdash e : A}\ (\textsc{Env Ext})$$

$$\frac{\mathcal{C} \vdash \diamond \qquad \mathcal{T} \vdash \diamond}{\mathcal{C}, \mathcal{T} \vdash \diamond} \text{ (EnvCT Com)} \qquad \frac{\mathcal{C} \vdash \diamond \qquad \mathcal{T} \vdash \diamond \qquad \mathcal{F} \vdash \diamond}{\mathcal{C}, \mathcal{T}, \mathcal{F} \vdash \diamond} \text{ (EnvCTF Com)}$$

$$\frac{\mathcal{C} \vdash e : A \qquad \mathcal{T} \vdash \diamond}{\mathcal{C}, \mathcal{T} \vdash e : A} \text{ (EnvC Ext1)} \qquad \frac{\mathcal{C} \vdash D \qquad \mathcal{T} \vdash \diamond}{\mathcal{C}, \mathcal{T} \vdash D} \text{ (EnvC Ext2)}$$

$$\frac{\mathcal{C}, \mathcal{T} \vdash e : A}{\mathcal{C}, \mathcal{T}, \mathcal{F} \vdash e : A} \text{ (EnvCNT Ext1)} \qquad \frac{\mathcal{C}, \mathcal{T} \vdash D}{\mathcal{C}, \mathcal{T}, \mathcal{F} \vdash D} \text{ (CEnvNT Ext2)}$$

3.4.2 Initialization of Typing Environments (declarations of constants, types and nodes)

$$\frac{\begin{array}{cc} c = ShortConst(i) \rightarrow t = short & c = UshortConst(i) \rightarrow t = ushort \\ c = IntConst(i) \rightarrow t = int & c = UintConst(i) \rightarrow t = uint \\ c = FloatConst(f) \rightarrow t = float & c = RealConst(f) \rightarrow t = real \\ c = BoolConst(b) \rightarrow t = bool & c = CharConst(i) \rightarrow t = char \end{array}}{\phi \vdash c : t} \ \phi \text{ CScalar}$$

$$\frac{\begin{array}{c} c = ArrayConst(c\_list) \\ c\_list = c_1 :: c_2 :: ... :: c_n \qquad \phi \vdash c_1 : t \quad \phi \vdash c_2 : t \quad ... \quad \phi \vdash c_n : t \end{array}}{\phi \vdash c : array(t, n)} \ \phi \text{ CArray}$$

$$\frac{\begin{array}{c} c = ConstructConst(f\_c\_list) \\ f\_c\_list = (id_1, c_1) :: (id_2, c_2) :: ... :: (id_n, c_n) \\ \phi \vdash c_1 : t_1 \quad \phi \vdash c_2 : t_2 \quad ... \quad \phi \vdash c_n : t_n \end{array}}{\phi \vdash c : struct((id_1, t_1) :: (id_2, t_2) :: ... :: (id_n, t_n))} \ \phi \text{ CStruct}$$

$$\frac{c = EnumConst(id, t) \qquad \mathcal{C}, \mathcal{T} \vdash t : enum(enum\_ids) \qquad id \in enum\_ids}{\mathcal{C}, \mathcal{T} \vdash c : t} \ \phi \text{ CEnum}$$

$$\frac{c = ID(x) \qquad \mathcal{C}, \mathcal{T} \vdash x : t}{\mathcal{C}, \mathcal{T} \vdash c : t} \text{ CIdent}$$

$$\frac{\begin{array}{c} ty \in \{Short, UShort, Int, UInt, Float, Real, Bool, Char\} \\ ty = Short \rightarrow t = short \\ ty = UShort \rightarrow t = ushort \qquad ty = TInt \rightarrow t = int \\ ty = Uint \rightarrow t = uint \qquad ty = Float \rightarrow t = float \qquad ty = Real \rightarrow t = real \\ ty = Bool \rightarrow t = bool \qquad ty = Char \rightarrow t = char \end{array}}{\phi \vdash ty : t} \ \phi \ \textsc{Scalar}$$

$$\frac{ty = Array(elemt, IntConst(n)) \qquad \mathcal{C}, \mathcal{T} \vdash elemt : t}{\mathcal{C}, \mathcal{T} \vdash ty : array(t, n)} \ \phi \ \textsc{Array}$$

$$\frac{\begin{array}{c} ty = Struct(f\_list) \\ f\_list = (id_1, ty_1) :: (id_2, ty_2) :: ... :: (id_n, ty_n), where\ n > 0 \\ undup(id_1, id_2, ..., id_n), where\ undup\ means\ unduplicated \\ \mathcal{C}, \mathcal{T} \vdash ty_1 : t_1 \quad \phi \vdash ty_2 : t_2 \quad ... \quad \phi \vdash ty_n : t_n \\ t = struct((id_1, t_1) :: (id_2, t_2) :: ... :: (id_n, t_n)) \end{array}}{\mathcal{C}, \mathcal{T} \vdash ty : t} \ \phi \ \textsc{Struct}$$

$$\frac{\begin{array}{c} ty = Enum(enum\_ids) \qquad enum\_ids = id_1 :: id_2 :: ... :: id_n, where\ n > 0 \\ undup(id_1, id_2, ..., id_n), where\ undup\ means\ unduplicated \\ t = enum(enum\_ids) \end{array}}{\phi \vdash ty : t} \ \phi \ \textsc{Enum}$$

$$\frac{\begin{array}{c} c\_decl = (x, ty, c) \\ \mathcal{C}', \mathcal{T} \vdash \diamond \qquad \mathcal{C}', \mathcal{T} \vdash ty : t \qquad \mathcal{C}' \vdash c : t \qquad x \notin dom(\mathcal{C}') \qquad \mathcal{C} = \mathcal{C}' \cup \{x : t\} \end{array}}{\mathcal{C}, \mathcal{T} \vdash c\_decl} \ \textsc{ECInit}$$

$$\frac{\begin{array}{c} t\_decl = (x, ty) \\ \mathcal{C}, \mathcal{T}' \vdash ty : t \qquad \mathcal{C}, \mathcal{T}' \vdash \diamond \qquad x \notin dom(\mathcal{T}') \qquad \mathcal{T} = \mathcal{T}' \cup \{x : t\} \end{array}}{\mathcal{C}, \mathcal{T} \vdash t\_decl} \ \textsc{ETInit Scalar}$$

$$
\frac{\begin{array}{c}
nd = (funcType, modifier, id, parameters, returns, body)\\
body = (locals, equation\_list)\\
\mathcal{C}, \mathcal{T}, \mathcal{F}'_{id} \vdash \diamond \qquad decls = parameters{+}{+}returns{+}{+}locals\\
undup(all\ id's\ in\ decls), where\ undup\ means\ unduplicated\\
VDecl(x, ty) \in decls \vee WhenVDecl(x, ty, ck) \in decls\\
WhenVDecl(x, ty, ck) \in decls \rightarrow \mathcal{C}, \mathcal{T}, \mathcal{F}'_{id} \vdash ck : bool \vee \exists\ eids(\mathcal{C}, \mathcal{T}, \mathcal{F}'_{id} \vdash ck : enum(eids))\\
WhenVDecl(x, ty, ck) \in parameters{+}{+}returns \rightarrow \exists y, eid, t', ck'.\\
((ck = (TRUE, y) \vee ck = (FALSE, y) \vee ck = (eid, y))\\
\wedge(VDecl(y, t') \in parameters \vee WhenVDecl(y, t', ck') \in parameters\\
\wedge\ y\ is\ different\ to\ x\ and\ any\ other\ variables\ which\ is\ on\ the\ clock\ definition\ chain\ down\ to\ x))\\
WhenVDecl(x, ty, ck) \in locals \rightarrow \exists y, eid, t', ck'.\\
((ck = (TRUE, y) \vee ck = (FALSE, y) \vee ck = (eid, y))\\
\wedge(VDecl(y, t') \in parameters{+}{+}locals \vee WhenVDecl(y, t', ck') \in parameters{+}{+}locals\\
\wedge\ y\ is\ different\ to\ x\ and\ any\ other\ variables\ which\ is\ on\ the\ clock\ definition\ chain\ down\ to\ x))\\
x \notin dom(\mathcal{C}) \cup dom(\mathcal{F}'_{id}) \qquad \mathcal{C}, \mathcal{T}, \mathcal{F}'_{id} \vdash ty : t \qquad \mathcal{F}_{id} = \mathcal{F}'_{id} \cup \{x : t\}
\end{array}}{\mathcal{C}, \mathcal{T}, \mathcal{F}_{id} \vdash \diamond}\ \textsc{ECTFInit}
$$

$$
\frac{\begin{array}{c}
nd = (funcType, modifier, id, parameters, returns, body) \qquad \mathcal{C}, \mathcal{T}, \mathcal{F}_{id} \vdash \diamond\\
parameters = d_1 :: d_2 :: ... :: d_n \qquad returns = r_1 :: r_2 :: ... :: r_m\\
\forall i : 1 \leq i \leq n.\ (d_i = VDecl(x_i, pt_i)\ \vee\ d_i = WhenVDecl(x_i, pt_i, ck_i))\\
\forall i : 1 \leq i \leq m.\ (r_i = VDecl(y_i, rt_i)\ \vee\ r_i = WhenVDecl(y_i, rt_i, ck'_i))\\
\forall i : 1 \leq i \leq n.\ \mathcal{C}, \mathcal{T} \vdash pt_i : t_i \qquad \forall i : 1 \leq i \leq m.\ \mathcal{C}, \mathcal{T} \vdash rt_i : t'_i
\end{array}}{\mathcal{C}, \mathcal{T} \vdash nd : t_1 \times t_2 \times ... \times t_n \rightarrow t'_1 \times t'_2 \times ... \times t'_m}\ \textsc{Node Def}
$$

3.4.3 Temporal and Clock Expressions

$$\frac{\begin{array}{c} e = Epre(expr) \qquad expr = e_1 :: e_2 :: ... :: e_n \\ \mathcal{C},\mathcal{T},\mathcal{F} \vdash \diamond \qquad \mathcal{C},\mathcal{T},\mathcal{F} \vdash e_1 : t_1 \quad \mathcal{C},\mathcal{T},\mathcal{F} \vdash e_2 : t_2 \quad ... \quad \mathcal{C},\mathcal{T},\mathcal{F} \vdash e_n : t_n \end{array}}{\mathcal{C},\mathcal{T},\mathcal{F} \vdash e : t_1 :: t_2 :: ... :: t_n} \ \textsc{Epre}$$

$$\frac{\begin{array}{c} e = Efby(expr, IntConst(k), expr') \\ expr = e_1 :: e_2 :: ... :: e_n \qquad expr' = e'_1 :: e'_2 :: ... :: e'_n \\ \mathcal{C},\mathcal{T},\mathcal{F} \vdash \diamond \qquad \mathcal{C},\mathcal{T},\mathcal{F} \vdash e_1 : t_1 \quad \mathcal{C},\mathcal{T},\mathcal{F} \vdash e_2 : t_2 \quad ... \quad \mathcal{C},\mathcal{T},\mathcal{F} \vdash e_n : t_n \\ \mathcal{C},\mathcal{T},\mathcal{F} \vdash e'_1 : t_1 \quad \mathcal{C},\mathcal{T},\mathcal{F} \vdash e'_2 : t_2 \quad ... \quad \mathcal{C},\mathcal{T},\mathcal{F} \vdash e'_n : t_n \end{array}}{\mathcal{C},\mathcal{T},\mathcal{F} \vdash e : t_1 :: t_2 :: ... :: t_n} \ \textsc{Efby}$$

$$\frac{\begin{array}{c} e = Earrow(expr, expr') \\ expr = e_1 :: e_2 :: ... :: e_n \qquad expr' = e'_1 :: e'_2 :: ... :: e'_n \\ \mathcal{C},\mathcal{T},\mathcal{F} \vdash \diamond \qquad \mathcal{C},\mathcal{T},\mathcal{F} \vdash e_1 : t_1 \quad \mathcal{C},\mathcal{T},\mathcal{F} \vdash e_2 : t_2 \quad ... \quad \mathcal{C},\mathcal{T},\mathcal{F} \vdash e_n : t_n \\ \mathcal{C},\mathcal{T},\mathcal{F} \vdash e'_1 : t_1 \quad \mathcal{C},\mathcal{T},\mathcal{F} \vdash e'_2 : t_2 \quad ... \quad \mathcal{C},\mathcal{T},\mathcal{F} \vdash e'_n : t_n \end{array}}{\mathcal{C},\mathcal{T},\mathcal{F} \vdash e : t_1 :: t_2 :: ... :: t_n} \ \textsc{Earrow}$$

$$\frac{\begin{array}{c} e = Ewhen(expr, ck) \qquad expr = e_1 :: e_2 :: ... :: e_n \\ \mathcal{C},\mathcal{T},\mathcal{F} \vdash \diamond \qquad \mathcal{C},\mathcal{T},\mathcal{F} \vdash ck : bool \vee \exists\ eids(\mathcal{C},\mathcal{T},\mathcal{F} \vdash ck : enum(eids)) \\ \mathcal{C},\mathcal{T},\mathcal{F} \vdash e_1 : t_1 \quad \mathcal{C},\mathcal{T},\mathcal{F} \vdash e_2 : t_2 \quad ... \quad \mathcal{C},\mathcal{T},\mathcal{F} \vdash e_n : t_n \end{array}}{\mathcal{C},\mathcal{T},\mathcal{F} \vdash e : t_1 :: t_2 :: ... :: t_n} \ \textsc{Ewhen}$$

$$\frac{\begin{array}{c} e = Emerge(id, expr, expr') \qquad \mathcal{C},\mathcal{T},\mathcal{F} \vdash \diamond \\ \mathcal{C},\mathcal{T},\mathcal{F} \vdash id : bool \qquad \mathcal{C},\mathcal{T},\mathcal{F} \vdash expr : t \qquad \mathcal{C},\mathcal{T},\mathcal{F} \vdash expr' : t \end{array}}{\mathcal{C},\mathcal{T},\mathcal{F} \vdash e : t} \ \textsc{Emerge}$$

$$\frac{\begin{array}{c} e = Emerge(id, merge\_case\_list) \\ merge\_case\_list = (TRUE, e_1) :: (FALSE, e_2) \\ \mathcal{C},\mathcal{T},\mathcal{F} \vdash \diamond \qquad \mathcal{C},\mathcal{T},\mathcal{F} \vdash id : bool \qquad \mathcal{C},\mathcal{T},\mathcal{F} \vdash e_1 : t \qquad \mathcal{C},\mathcal{T},\mathcal{F} \vdash e_2 : t \end{array}}{\mathcal{C},\mathcal{T},\mathcal{F} \vdash e : t} \ \textsc{Emerge Bool}$$

$$\frac{\begin{array}{c} e = Emerge(id, merge\_case\_list) \qquad ident\_list = id_1 :: id_2 :: ... :: id_n \\ merge\_case\_list = (id_1, e_1) :: (id_2, e_2) :: ... :: (id_n, e_n) \\ \mathcal{C},\mathcal{T},\mathcal{F} \vdash \diamond \qquad \mathcal{C},\mathcal{T},\mathcal{F} \vdash id : enum(ident\_list) \\ \mathcal{C},\mathcal{T},\mathcal{F} \vdash e_1 : t \quad \mathcal{C},\mathcal{T},\mathcal{F} \vdash e_2 : t \quad ... \quad \mathcal{C},\mathcal{T},\mathcal{F} \vdash e_n : t \end{array}}{\mathcal{C},\mathcal{T},\mathcal{F} \vdash e : t} \ \textsc{Emerge Enum}$$

$$\frac{\begin{array}{c} e = Ecurrent(expr) \qquad expr = e_1 :: e_2 :: ... :: e_n \\ \mathcal{C},\mathcal{T},\mathcal{F} \vdash \diamond \qquad \mathcal{C},\mathcal{T},\mathcal{F} \vdash e_1 : t_1 \quad \mathcal{C},\mathcal{T},\mathcal{F} \vdash e_2 : t_2 \quad ... \quad \mathcal{C},\mathcal{T},\mathcal{F} \vdash e_n : t_n \end{array}}{\mathcal{C},\mathcal{T},\mathcal{F} \vdash e : t_1 :: t_2 :: ... :: t_n} \ \textsc{Ecurrent}$$

3.4.4 Call Expressions

$$\frac{\begin{array}{c} e = ApplyExpr(id, args\_list) \\ nd = (funcType, modifier, id, parameters, returns, body) \\ args\_list = e_1 :: e_2 :: ... :: e_n \\ \mathcal{C}, \mathcal{T}, \mathcal{F} \vdash \diamond \qquad \mathcal{C}, \mathcal{T} \vdash nd : t_1 \times t_2 \times ... \times t_n \rightarrow t'_1 \times t'_2 \times ... \times t'_m \\ \forall i : 1 \leq i \leq n.\ \mathcal{C}, \mathcal{T} \vdash d_i : t_i \\ \forall i : 1 \leq i \leq m.\ \mathcal{C}, \mathcal{T} \vdash r_i : t'_i \qquad \forall i : 1 \leq i \leq n.\ (\mathcal{C}, \mathcal{T}, \mathcal{F} \vdash e_i : t_i) \end{array}}{\mathcal{C}, \mathcal{T}, \mathcal{F} \vdash e : t'_1 :: t'_2 :: ... :: t'_m} \text{ CALL}$$

3.4.5 Higher-Order Operations

$$\frac{\begin{array}{c} e = ApplyExpr(iter, expr\_list) \qquad iter = Iterator(Omap, id, IntConst(k)) \\ nd = (funcType, modifier, id, parameters, returns, body) \\ expr\_list = expr_1 :: expr_2 :: ... :: expr_n \\ \mathcal{C}, \mathcal{T}, \mathcal{F} \vdash \diamond \qquad \mathcal{C}, \mathcal{T} \vdash nd : t_1 \times t_2 \times ... \times t_n \rightarrow t'_1 \times t'_2 \times ... \times t'_m \\ \forall i : 1 \leq i \leq n.\ (\mathcal{C}, \mathcal{T}, \mathcal{F} \vdash expr_i : array(t_i, k)) \\ \forall i : 1 \leq i \leq m.\ (rt_i = array(t'_i, k)) \end{array}}{\mathcal{C}, \mathcal{T}, \mathcal{F} \vdash e : rt_1 \times rt_2 \times ... \times rt_m} \text{ MAP USR-OP}$$

$$\frac{\begin{array}{c} e = ApplyExpr(iter, expr) \qquad iter = Iterator(Ofill, id, IntConst(k)) \\ nd = (funcType, modifier, id, parameters, returns, body) \\ \mathcal{C}, \mathcal{T}, \mathcal{F} \vdash \diamond \qquad \mathcal{C}, \mathcal{T} \vdash nd : t \rightarrow t \times t_1 \times t_2 \times ... \times t_m, where\ m \geq 1 \\ \mathcal{C}, \mathcal{T}, \mathcal{F} \vdash expr : t \qquad \forall i : 1 \leq i \leq m.\ (rt_i = array(t_i, k)) \end{array}}{\mathcal{C}, \mathcal{T}, \mathcal{F} \vdash e : t \times rt_1 \times rt_2 \times ... \times rt_m} \text{ FILL USR-OP}$$

$$\frac{\begin{array}{c} e = ApplyExpr(iter, expr\_list) \qquad iter = Iterator(Ored, id, IntConst(k)) \\ nd = (funcType, modifier, id, parameters, returns, body) \\ \mathcal{C}, \mathcal{T}, \mathcal{F} \vdash \diamond \qquad \mathcal{C}, \mathcal{T} \vdash nd : t \times t_1 \times t_2 \times ... \times t_n \rightarrow t, where\ n \geq 1 \\ \forall i : 1 \leq i \leq n.\ (at_i = array(t_i, k)) \\ \mathcal{C}, \mathcal{T}, \mathcal{F} \vdash expr\_list : t \times at_1 \times at_2 \times ... \times at_n \end{array}}{\mathcal{C}, \mathcal{T}, \mathcal{F} \vdash e : t} \text{ RED USR-OP}$$

$$\frac{\begin{array}{c} e = ApplyExpr(iter, expr\_list) \qquad iter = Iterator(Ofillred, id, IntConst(k)) \\ nd = (funcType, modifier, id, parameters, returns, body) \qquad \mathcal{C}, \mathcal{T}, \mathcal{F} \vdash \diamond \\ \mathcal{C}, \mathcal{T} \vdash nd : t \times t_1 \times t_2 \times ... \times t_n \rightarrow t \times t'_1 \times t'_2 \times ... \times t'_m, where\ n, m \geq 1 \\ \forall i : 1 \leq i \leq n.\ (at_i = array(t_i, k)) \\ \mathcal{C}, \mathcal{T}, \mathcal{F} \vdash expr\_list : t \times at_1 \times at_2 \times ... \times at_n \\ \forall i : 1 \leq i \leq m.\ (rt_i = array(t'_i, k)) \end{array}}{\mathcal{C}, \mathcal{T}, \mathcal{F} \vdash e : t \times rt_1 \times rt_2 \times ... \times rt_m} \text{ FILLRED USR-OP}$$

$$\frac{\begin{array}{c} e = ApplyExpr(iter, expr) \qquad iter = Iterator(Omap, un\_op, IntConst(k)) \\ un\_op \in \{Oshort, Oint, ..., Oneg\} \\ \mathcal{C}, \mathcal{T}, \mathcal{F} \vdash \diamond \qquad \mathcal{C}, \mathcal{T}, \mathcal{F} \vdash expr : array(t, k) \end{array}}{\mathcal{C}, \mathcal{T}, \mathcal{F} \vdash e : array(t, k)} \textsc{ Map PrefixUnOp}$$

$$\frac{\begin{array}{c} e = ApplyExpr(iter, expr_1 :: expr_2) \\ iter = Iterator(Omap, bin\_op, IntConst(k)) \\ bin\_op \in \{Oadd, Osub, ..., Oge\} \qquad \mathcal{C}, \mathcal{T}, \mathcal{F} \vdash \diamond \\ \mathcal{C}, \mathcal{T}, \mathcal{F} \vdash expr_1 : array(t, k) \qquad \mathcal{C}, \mathcal{T}, \mathcal{F} \vdash expr_2 : array(t, k) \end{array}}{\mathcal{C}, \mathcal{T}, \mathcal{F} \vdash e : array(t, k)} \textsc{ Map PrefixBinOp}$$

$$\frac{\begin{array}{c} e = ApplyExpr(iter, expr_1 :: expr_2) \\ iter = Iterator(Ored, bin\_op, IntConst(k)) \\ bin\_op \in \{Oadd, Osub, ..., Oge\} \\ \mathcal{C}, \mathcal{T}, \mathcal{F} \vdash \diamond \qquad \mathcal{C}, \mathcal{T}, \mathcal{F} \vdash expr_1 : t \qquad \mathcal{C}, \mathcal{T}, \mathcal{F} \vdash expr_2 : array(t, k) \end{array}}{\mathcal{C}, \mathcal{T}, \mathcal{F} \vdash e : t} \textsc{ Red PrefixBinOp}$$

$$\frac{\begin{array}{c} e = ApplyExpr(iter, expr_1 :: expr_2) \\ iter = Iterator(Ofillred, bin\_op, IntConst(k)) \\ bin\_op \in \{Oadd, Osub, ..., Oge\} \\ \mathcal{C}, \mathcal{T}, \mathcal{F} \vdash \diamond \qquad \mathcal{C}, \mathcal{T}, \mathcal{F} \vdash expr_1 : t \qquad \mathcal{C}, \mathcal{T}, \mathcal{F} \vdash expr_2 : array(t, k) \end{array}}{\mathcal{C}, \mathcal{T}, \mathcal{F} \vdash e : t \times array(t, k)} \textsc{ Fillred PrefixBinOp}$$

3.4.6 Array and Struct Expressions

$$\frac{\begin{array}{c} e = Econstruct(field\_expr\_list) \\ field\_expr\_list = (id_1, e_1) :: (id_2, e_2) :: ... :: (id_n, e_n) \\ \mathcal{C}, \mathcal{T}, \mathcal{F} \vdash \diamond \qquad \mathcal{C}, \mathcal{T}, \mathcal{F} \vdash e_1 : t_1 \quad \mathcal{C}, \mathcal{T}, \mathcal{F} \vdash e_2 : t_2 \quad ... \quad \mathcal{C}, \mathcal{T}, \mathcal{F} \vdash e_n : t_n \end{array}}{\mathcal{C}, \mathcal{T}, \mathcal{F} \vdash e : struct((id_1, t_1) :: (id_2, t_2) :: ... :: (id_n, t_n))} \textsc{ Construct}$$

$$\frac{\begin{array}{c} e = Earrayacc(expr, index) \\ \mathcal{C},\mathcal{T},\mathcal{F} \vdash \diamond \qquad \mathcal{C},\mathcal{T},\mathcal{F} \vdash expr : array(t,n) \qquad \mathcal{C},\mathcal{T},\mathcal{F} \vdash index : int \end{array}}{\mathcal{C},\mathcal{T},\mathcal{F} \vdash e : t}\ \textsc{Array Acc}$$

$$\frac{e = Earraydef(expr, IntConst(n)) \qquad \mathcal{C},\mathcal{T},\mathcal{F} \vdash \diamond \qquad \mathcal{C},\mathcal{T},\mathcal{F} \vdash expr : t}{\mathcal{C},\mathcal{T},\mathcal{F} \vdash e : array(t,n)}\ \textsc{Array Def}$$

$$\frac{\begin{array}{c} e = Earraydiff(expr\_list) \qquad expr\_list = e_1 :: e_2 :: ... :: e_n \\ \mathcal{C},\mathcal{T},\mathcal{F} \vdash \diamond \qquad \forall i : 1 \leq i \leq n.(\mathcal{C},\mathcal{T},\mathcal{F} \vdash e_i : t) \end{array}}{\mathcal{C},\mathcal{T},\mathcal{F} \vdash e : array(t,n)}\ \textsc{Array Diff}$$

$$\frac{\begin{array}{c} e = Earrayproj(expr_1, expr\_list, expr_2) \\ expr\_list = index_1 :: index_2 :: ... :: index_k \\ \mathcal{C},\mathcal{T},\mathcal{F} \vdash \diamond \qquad \mathcal{C},\mathcal{T},\mathcal{F} \vdash expr_1 : array(t_1,n_1) \qquad \mathcal{C},\mathcal{T},\mathcal{F} \vdash t_1 : array(t_2,n_2) \\ ... \qquad \mathcal{C},\mathcal{T},\mathcal{F} \vdash t_{k-2} : array(t_{k-1},n_{k-1}) \qquad \mathcal{C},\mathcal{T},\mathcal{F} \vdash t_{k-1} : array(t,n_k) \\ \mathcal{C},\mathcal{T},\mathcal{F} \vdash expr_2 : t \qquad \forall i : 1 \leq i \leq k.(\mathcal{C},\mathcal{T},\mathcal{F} \vdash index_i : int) \end{array}}{\mathcal{C},\mathcal{T},\mathcal{F} \vdash e : t}\ \textsc{Array Proj}$$

$$\frac{\begin{array}{c} e = Earrayslice(expr, expr_1, expr_2) \\ \mathcal{C},\mathcal{T},\mathcal{F} \vdash \diamond \qquad \mathcal{C},\mathcal{T},\mathcal{F} \vdash expr : array(t,n) \\ \mathcal{C},\mathcal{T},\mathcal{F} \vdash expr_1 : int \wedge expr_1 = Econst(IntConst(i)) \\ \mathcal{C},\mathcal{T},\mathcal{F} \vdash expr_2 : int \wedge expr_2 = Econst(IntConst(j)) \qquad 0 \leq i < j < n \end{array}}{\mathcal{C},\mathcal{T},\mathcal{F} \vdash e : array(t, j-i+1)}\ \textsc{Array Slice}$$

$$\frac{\begin{array}{c} e = Emix(expr, label\_index\_list, expr_1) \\ label\_index\_list = li_1 :: li_2 :: ... :: li_n \qquad \mathcal{C},\mathcal{T},\mathcal{F} \vdash \diamond \qquad \mathcal{C},\mathcal{T},\mathcal{F} \vdash expr : t \\ t = array(t_1,k_1) \rightarrow \mathcal{C},\mathcal{T} \vdash li_1 : int \quad t = struct(f\_list_1) \rightarrow (li_1, t_1) \in f\_list_1 \\ t_1 = array(t_2,k_2) \rightarrow \mathcal{C},\mathcal{T} \vdash li_2 : int \quad t_1 = struct(f\_list_2) \rightarrow (li_2, t_2) \in f\_list_2 \\ ... \\ t_{n-1} = array(t_n,k_n) \rightarrow \mathcal{C},\mathcal{T} \vdash li_n : int \quad t_{n-1} = struct(f\_list_n) \rightarrow (li_n, t_n) \in f\_list_n \\ \mathcal{C},\mathcal{T},\mathcal{F} \vdash expr_1 : t_n \end{array}}{\mathcal{C},\mathcal{T},\mathcal{F} \vdash e : t_n}\ \textsc{Mix}$$

$$\frac{\begin{array}{c} e = Efield(expr, ident) \\ \mathcal{C},\mathcal{T},\mathcal{F} \vdash \diamond \qquad \mathcal{C},\mathcal{T},\mathcal{F} \vdash expr : struct(f\_list) \qquad (ident, t) \in f\_list \end{array}}{\mathcal{C},\mathcal{T},\mathcal{F} \vdash e : t}\ \textsc{Field}$$

3.4.7 Other Expressions or Sub-Expressions

$$\frac{e = Econst(atom\_c) \qquad \mathcal{C},\mathcal{T},\mathcal{F} \vdash \diamond \qquad \mathcal{C},\mathcal{T} \vdash atom\_c : t}{\mathcal{C},\mathcal{T},\mathcal{F} \vdash e : t}\ \textsc{Const}$$

$$\frac{e = Evar(x) \qquad \mathcal{C},\mathcal{T},\mathcal{F} \vdash \diamond \qquad \mathcal{C},\mathcal{T},\mathcal{F} \vdash x : t}{\mathcal{C},\mathcal{T},\mathcal{F} \vdash e : t}\ \textsc{Var}$$

$$\frac{\begin{array}{c} e = ListExpr(e_1 :: e_2 :: ... :: e_n) \\ \mathcal{C},\mathcal{T},\mathcal{F} \vdash \diamond \qquad \mathcal{C},\mathcal{T},\mathcal{F} \vdash e_1 : t_1 \quad \mathcal{C},\mathcal{T},\mathcal{F} \vdash e_2 : t_2 \quad ... \quad \mathcal{C},\mathcal{T},\mathcal{F} \vdash e_n : t_n \end{array}}{\mathcal{C},\mathcal{T},\mathcal{F} \vdash e : t_1 :: t_2 :: ... :: t_n}\ \textsc{ListExpr}$$

$$\frac{\begin{array}{c} e = Eunop(un\_op, expr) \\ un\_op \in \{Oshort, Oint, ..., Oneg\} \qquad \mathcal{C},\mathcal{T},\mathcal{F} \vdash \diamond \qquad \mathcal{C},\mathcal{T},\mathcal{F} \vdash expr : t \end{array}}{\mathcal{C},\mathcal{T},\mathcal{F} \vdash e : t}\ \textsc{UnOp}$$

$$\frac{\begin{array}{c} e = Ebinop(bin\_op, expr_1, expr_2) \qquad bin\_op \in \{Oadd, Osub, ..., Oge\} \\ \mathcal{C},\mathcal{T},\mathcal{F} \vdash \diamond \qquad \mathcal{C},\mathcal{T},\mathcal{F} \vdash expr_1 : t \qquad \mathcal{C},\mathcal{T},\mathcal{F} \vdash expr_2 : t \end{array}}{\mathcal{C},\mathcal{T},\mathcal{F} \vdash e : t}\ \textsc{BinOp}$$

$$\frac{\begin{array}{c} e = Eif(expr, expr', expr'') \qquad expr' = e'_1 :: e'_2 :: ... :: e'_n \\ expr'' = e''_1 :: e''_2 :: ... :: e''_n \qquad \mathcal{C},\mathcal{T},\mathcal{F} \vdash \diamond \qquad \mathcal{C},\mathcal{T},\mathcal{F} \vdash expr : bool \\ \mathcal{C},\mathcal{T},\mathcal{F} \vdash e'_1 : t_1 \quad \mathcal{C},\mathcal{T},\mathcal{F} \vdash e'_2 : t_2 \quad ... \quad \mathcal{C},\mathcal{T},\mathcal{F} \vdash e'_n : t_n \\ \mathcal{C},\mathcal{T},\mathcal{F} \vdash e''_1 : t_1 \quad \mathcal{C},\mathcal{T},\mathcal{F} \vdash e''_2 : t_2 \quad ... \quad \mathcal{C},\mathcal{T},\mathcal{F} \vdash e''_n : t_n \end{array}}{\mathcal{C},\mathcal{T},\mathcal{F} \vdash e : t_1 :: t_2 :: ... :: t_n}\ \textsc{If}$$

$$\frac{\begin{array}{c} e = Ecase(expr, p\_list) \\ p\_list = (p_1, e_1) :: (p_2, e_2) :: ... :: (p_n, e_n) \qquad \mathcal{C},\mathcal{T},\mathcal{F} \vdash \diamond \\ \mathcal{C},\mathcal{T},\mathcal{F} \vdash expr : t' \qquad \forall i : 1 \leq i \leq n.\ (\mathcal{C},\mathcal{T},\mathcal{F} \vdash p_i : t' \vee \mathcal{C},\mathcal{T},\mathcal{F} \vdash p_i : void) \\ \forall i : 1 \leq i \leq n.\ (\mathcal{C},\mathcal{T},\mathcal{F} \vdash e_i : t) \end{array}}{\mathcal{C},\mathcal{T},\mathcal{F} \vdash e : t}\ \textsc{Case}$$

$$\frac{\begin{array}{c} (p = PatternID(ident, enum\_type) \vee p = PatternChar(i) \vee \\ p = PatternInt(j) \vee p = PatternBool(b) \vee p = PatternAny) \\ enum\_type = Tenum(id, enum\_ids) \\ ident \in enum\_ids \qquad \mathcal{C},\mathcal{T},\mathcal{F} \vdash \diamond \qquad \mathcal{C},\mathcal{T} \vdash enum\_type : enum\_t \\ p = PatternID(ident, enum\_type) \rightarrow t = enum\_t \\ p = PatternChar(i) \rightarrow t = char \qquad p = PatternInt(j) \rightarrow t = int \\ p = PatternBool(b) \rightarrow t = bool \qquad p = PatternAny \rightarrow t = void \end{array}}{\mathcal{C},\mathcal{T},\mathcal{F} \vdash p : t}\ \textsc{Pattern}$$

$$\frac{\begin{array}{c} e = \mathit{Eboolred}(\mathit{IntConst}(i), \mathit{IntConst}(j), \mathit{IntConst}(k), \mathit{expr}) \\ \mathit{expr} = e_1 :: e_2 :: ... :: e_k \\ 0 \leq i \leq j \leq k \wedge k > 0 \qquad \mathcal{C}, \mathcal{T}, \mathcal{F} \vdash \diamond \qquad \forall l : 1 \leq l \leq k.\ (\mathcal{C}, \mathcal{T}, \mathcal{F} \vdash e_l : \mathit{bool}) \end{array}}{\mathcal{C}, \mathcal{T}, \mathcal{F} \vdash e : \mathit{bool}} \text{ BOOLRED}$$

$$\frac{\begin{array}{c} e = \mathit{Ediese}(\mathit{expr}) \\ \mathit{expr} = e_1 :: e_2 :: ... :: e_n \qquad \mathcal{C}, \mathcal{T}, \mathcal{F} \vdash \diamond \qquad \forall l : 1 \leq l \leq n.\ (\mathcal{C}, \mathcal{T}, \mathcal{F} \vdash e_l : \mathit{bool}) \end{array}}{\mathcal{C}, \mathcal{T}, \mathcal{F} \vdash e : \mathit{bool}} \text{ DIESE}$$

$$\frac{\begin{array}{c} e = \mathit{EnorS}(\mathit{expr}) \\ \mathit{expr} = e_1 :: e_2 :: ... :: e_n \qquad \mathcal{C}, \mathcal{T}, \mathcal{F} \vdash \diamond \qquad \forall l : 1 \leq l \leq n.\ (\mathcal{C}, \mathcal{T}, \mathcal{F} \vdash e_l : \mathit{bool}) \end{array}}{\mathcal{C}, \mathcal{T}, \mathcal{F} \vdash e : \mathit{bool}} \text{ NOR}$$

$$\frac{\mathit{ck} = (\mathit{TRUE}, \mathit{id}) \vee \mathit{ck} = (\mathit{FALSE}, \mathit{id}) \qquad \mathcal{C}, \mathcal{T}, \mathcal{F} \vdash \diamond \qquad \mathcal{C}, \mathcal{T}, \mathcal{F} \vdash \mathit{id} : \mathit{bool}}{\mathcal{C}, \mathcal{T}, \mathcal{F} \vdash \mathit{ck} : \mathit{bool}} \text{ CLOCKBOOL}$$

$$\frac{\begin{array}{c} \mathit{ck} = (\mathit{eid}, \mathit{id}) \\ \mathcal{C}, \mathcal{T}, \mathcal{F} \vdash \diamond \qquad \mathcal{C}, \mathcal{T}, \mathcal{F} \vdash \mathit{id} : \mathit{enum}(\mathit{enum\_ids}) \qquad \mathit{eid} \in \mathit{enum\_ids} \end{array}}{\mathcal{C}, \mathcal{T}, \mathcal{F} \vdash \mathit{ck} : \mathit{enum}(\mathit{enum\_ids})} \text{ CLOCKENUM}$$

3.4.8 Equations

$$\frac{\begin{array}{c} eq = Equation(x, expr) \\ \mathcal{C}, \mathcal{T}, \mathcal{F}' \vdash \diamond \qquad x \notin dom(\mathcal{F}') \qquad \mathcal{F}' \vdash expr : t \qquad \mathcal{F} = \mathcal{F}' \cup \{x : t\} \end{array}}{\mathcal{C}, \mathcal{T}, \mathcal{F} \vdash eq} \text{ USUAL EQUATION SIG1}$$

$$\frac{\begin{array}{c} eq = Equation(x, expr) \\ \mathcal{C}, \mathcal{T}, \mathcal{F} \vdash \diamond \qquad \mathcal{C}, \mathcal{T}, \mathcal{F} \vdash x : t \qquad \mathcal{C}, \mathcal{T}, \mathcal{F} \vdash expr : t \end{array}}{\mathcal{C}, \mathcal{T}, \mathcal{F} \vdash eq} \text{ USUAL EQUATION SIG2}$$

$$\frac{\begin{array}{c} eq = Equation(lhs, exprs) \qquad lhs = lh :: lhs' \qquad exprs = expr :: exprs' \\ eq' = Equation(lh, expr) \qquad eq'' = Equation(lhs', exprs') \\ \mathcal{C}, \mathcal{T}, \mathcal{F} \vdash \diamond \qquad \mathcal{C}, \mathcal{T}, \mathcal{F} \vdash eq' \qquad \mathcal{C}, \mathcal{T}, \mathcal{F} \vdash eq'' \end{array}}{\mathcal{C}, \mathcal{T}, \mathcal{F} \vdash eq} \text{ USUAL EQUATION COM}$$

$$\frac{\begin{array}{c} eq = Equation(lhs, ApplyExpr(op, args\_list)) \\ \mathcal{C}, \mathcal{T}, \mathcal{F}' \vdash \diamond \qquad \mathcal{C}, \mathcal{T}, \mathcal{F}' \vdash ApplyExpr(op, args\_list) : t_1 \times t_2 \times ... \times t_m \\ lhs = x_1 :: x_2 :: ... :: x_m \\ \forall i : 1 \le i \le m.\ (\mathcal{H} \vdash x_i : t_i) \qquad \mathcal{F} = \mathcal{F}' \cup \{x_i : t_i | x_i \notin dom(\mathcal{F}'), 1 \le i \le m\} \end{array}}{\mathcal{C}, \mathcal{T}, \mathcal{F} \vdash eq} \text{ APPLY EQUATION}$$

3.4.9 Equation List, Node Block, Const Block, Type Block and Programs

$$\frac{eqs = eq :: eqs' \qquad \mathcal{C}, \mathcal{T}, \mathcal{F} \vdash \diamond \qquad \mathcal{C}, \mathcal{T}, \mathcal{F} \vdash eq \qquad \mathcal{C}, \mathcal{T}, \mathcal{F} \vdash eqs'}{\mathcal{C}, \mathcal{T}, \mathcal{F} \vdash eqs} \text{ EQUATIONLIST}$$

$$\frac{\begin{array}{c} nd = (funcType, modifier, id, parameters, returns, body) \\ body = (locals, eq\_list) \\ \mathcal{C}, \mathcal{T}, \mathcal{F}_{id} \vdash \diamond \qquad funcType = node \lor funcType = function \\ modifier = main \lor modifier = notMain \qquad \mathcal{C}, \mathcal{T}, \mathcal{F} \vdash equation\_list \\ \forall x \in ids(parameters).\ x \notin left\_ids(eq\_list), i.e., input\ variables\ should\ not\ be\ redefined \\ \forall x \in ids(returns).\ x \notin right\_ids(eq\_list), i.e., output\ variables\ are\ not\ available\ to\ access \end{array}}{\mathcal{C}, \mathcal{T} \vdash nd} \text{ NODE}$$

$$\frac{\begin{array}{c} nd\_main \in p \qquad nd\_main = (node, main, main\_id, parameters, returns, body) \\ undup(all\ node/function\ id's\ in\ p), where\ undup\ means\ unduplicated\ (the\ same\ below) \\ undup(all\ type\ id's\ in\ p), or\ being\ checked\ in\ the\ formation\ of\ \mathcal{T} \\ undup(all\ const\ id's\ in\ p), or\ being\ checked\ in\ the\ formation\ of\ \mathcal{C} \\ \mathcal{C}, \mathcal{T} \vdash \diamond \qquad \forall\ nd \in p.\ (nd\ is\ a\ node\ declaration\ in\ p \to \mathcal{C}, \mathcal{T} \vdash nd) \\ \forall\ t\_decl \in p.\ (t\_decl\ is\ a\ type\ declaration\ in\ p \to \mathcal{C}, \mathcal{T} \vdash t\_decl) \\ \forall\ c\_decl \in p.\ (c\_decl\ is\ a\ const\ declaration\ in\ p \to \mathcal{C}, \mathcal{T} \vdash c\_decl) \end{array}}{\phi \vdash p} \text{ PROGRAM}$$

4 Type System for clock checking

Clock calculus deals with the correctness and consistency of clock definitions and usage within and among function calls. Clock checker checks the validity of the clock operators that manipulate multiple clocks, based on the rules of clock calculus.

4.1 Clocks

4.1.1 Basic clock expressions

In a *Lustre** program, each node/function has a *base* clock. This is the clock used when the node/function is called. Each of arguments of the node/function may also be associated with a different clock when it is called. The base clock of the *Main* node is the reference clock of the whole program used directly or indirectly by all of the clocks derived in its call tree. A basic clock is expressed as follows.

clock	::=	*base*	a base clock
	|	*clock* on *ck*	a derived clock
ck	::=	*id* | *not id*	*id* is a *bool* variable
	|	*id match enum_id*	*id* is an *enum* variable

The expression *"clock on ck"* performs the sampling of the clock *ck*, i.e. deriving a new clock based on *ck*, using a variable or constant name, or the negation of such an identifier, or a pattern matching. In the first two cases, the identifier must has a *bool* data type. In the latter case, the identifier must has an enum data type, and the pattern part must refer to one of the item of this data type.
The expression *"clock on ck* is used to realize the *When* operator. Here, *clock* has a higher clock rate, and the derived (or sampled) clock from the expression *"clock on ck"* has a lower clock rate because those clock cycles, when c is False or does not match a specific enum item , are skipped.

4.1.2 Compound clock expressions

The compound clock expression $ck_1 :: ck_2 :: ... :: ck_n$ is the clock of tuples of expressions with the first component of clock expression ck_1, the second component of clock expression ck_2, ..., and the nth component of clock expression ck_n. It is possible that $n = 0$, where it means a *nil* list of clock expressions.

4.1.3 Constant clock expression

A constant expression is assigned a special clock expression, *ConstCK*, that is compatible with any other clock expressions. Generally, *ConstCK* is used in the places where, in any time, all the operators in an operator other than a user-defined one must have a uniform clock *CK*, unless it has the clock

ConstCK, and the result expression will has the clock *CK*. In other words, *ConstCK* may be assumed to be the clock of the context it is used in.
In the clock type rules below, we use clock_check_calcclock_listto check the consistency of clocks in a clock list *clock_list*, and return the uniform clock. It first leaves out all *ConstCK* in *clock_list*and return *ConstCK* if the clocks in *clock_list* are all *ConstCK*. Then, it checks all the clocks in the left clock list, and returns the uniform clock if they are all equal with each other, otherwise, it returns a "unmatched clocks" error.

4.2 Clock environments

A local clock environment $\mathcal{H}$ associates clocks to variables and has the form

$$\mathcal{H} ::= [\ x_1 : ck_1,\, x_2 : ck_2,\, ...,\, x_n : ck_n\ ]$$

where $x_i \neq x_j$ for $i \neq j$.

In some cases, we use $\mathcal{H}_{id}$ to denote a particular local clock environment specific to the context of a node/function identified by *id*.

4.3 Judgements

- $\mathcal{H} \vdash e : ck$, implies that,
  under the clock environment $\mathcal{H}$, the expression e is well-clocked and has the clock ck.
- $\mathcal{H} \vdash D$, implies that
  under the clock environment $\mathcal{H}$, the program component D is well-clocked.
- $\mathcal{H} \vdash \diamond$, implies that
  the clock environment $\mathcal{H}$ is well-defined.

4.4 Rules for the clock calculus on the abstract syntax of well-typed *Lustre**

In L2C project, the clock checking is performed immediately after the conventional type checking. So it is supposed that the type system for the clock checking be designed for a well typed program, that is, a program p satisfying $\phi \vdash p$ as is derived by the very last rule in Section 3.4.9. The type system for the clock checking is given by defining the rules for the clock calculus as follows.

4.4.1 Common

$$\frac{}{\phi \vdash \diamond}\ (\textsc{Env}\ \phi) \qquad \frac{\mathcal{H}' \vdash \diamond \qquad x \notin dom(\mathcal{H}') \qquad \mathcal{H} = \mathcal{H}' \cup \{x : x\_ck\}}{\mathcal{H} \vdash \diamond}\ (\textsc{Env WellCK})$$

$$\frac{\mathcal{H} \vdash \diamond \qquad x : x\_ck \in \mathcal{H}}{\mathcal{H} \vdash x : x\_ck}\ (\textsc{Env CkVal})$$

4.4.2 Initialization of Clock Environments

$$\frac{\begin{array}{c} nd = (funcType, modifier, id, parameters, returns, body) \\ body = (locals, eq\_list) \\ \mathcal{H}'_{id} \vdash \diamond \qquad VDecl(x,t) \in parameters{+}{+}returns{+}{+}locals \\ x \notin dom(\mathcal{H}'_{id}) \qquad \mathcal{H}_{id} = \mathcal{H}'_{id} \cup \{x : base\} \end{array}}{\mathcal{H}_{id} \vdash \diamond}\ \textsc{EnvInit noWhen}$$

$$\frac{\begin{array}{c} nd = (funcType, modifier, id, parameters, returns, body) \\ body = (locals, eq\_list) \\ \mathcal{H}'_{id} \vdash \diamond \qquad VDecl(x,t') \in parameters \vee WhenVDecl(x,t',c') \in parameters \\ c = (b,x) \qquad WhenVDecl((y,t,c) \in parameters{+}{+}returns \qquad x \neq y \\ y \notin dom(\mathcal{H}'_{id}) \qquad \mathcal{H}'_{id} \vdash x : x\_ck \qquad \mathcal{H}_{id} = \mathcal{H}'_{id} \cup \{y : x\_ck\ on\ c\} \end{array}}{\mathcal{H}_{id} \vdash \diamond}\ \textsc{EnvInit hasWhenPR}$$

$$\frac{\begin{array}{c} nd = (funcType, modifier, id, parameters, returns, body) \\ body = (locals, eq\_list) \qquad \mathcal{H}'_{id} \vdash \diamond \\ VDecl(x,t') \in parameters{+}{+}locals \vee WhenVDecl(x,t',c') \in parameters{+}{+}locals \\ c = (b,x) \qquad WhenVDecl(y,t,c) \in locals \qquad x \neq y \\ y \notin dom(\mathcal{H}'_{id}) \qquad \mathcal{H}'_{id} \vdash x : x\_ck \qquad \mathcal{H}_{id} = \mathcal{H}'_{id} \cup \{y : x\_ck\ on\ c\} \end{array}}{\mathcal{H}_{id} \vdash \diamond}\ \textsc{EnvInit hasWhenL}$$

4.4.3 Temporal and Clock Expressions

$$\frac{\begin{array}{c} e = Epre(expr) \qquad expr = e_1 :: e_2 :: ... :: e_n \\ \mathcal{H} \vdash \diamond \qquad \mathcal{H} \vdash e_1 : ck_1 \quad \mathcal{H} \vdash e_2 : ck_2 \quad ... \quad \mathcal{H} \vdash e_n : ck_n \\ ck = clock\_check\_calc(ck_1 :: ck_2 :: ... :: ck_n) \end{array}}{\mathcal{H} \vdash e : (ck)^n} \ \textsc{Epre}$$

$$\frac{\begin{array}{c} e = Efby(expr, IntConst(k), expr') \\ expr = e_1 :: e_2 :: ... :: e_n \qquad expr' = e'_1 :: e'_2 :: ... :: e'_n \\ \mathcal{H} \vdash \diamond \qquad \mathcal{H} \vdash e_1 : ck_1 \quad \mathcal{H} \vdash e_2 : ck_2 \quad ... \quad \mathcal{H} \vdash e_n : ck_n \\ \mathcal{H} \vdash e'_1 : ck'_1 \quad \mathcal{H} \vdash e'_2 : ck'_2 \quad ... \quad \mathcal{H} \vdash e'_n : ck'_n \\ c = clock\_check\_calc(ck_1 :: ck_2 :: ... :: ck_n) \\ c' = clock\_check\_calc(ck'_1 :: ck'_2 :: ... :: ck'_n) \\ ck = clock\_check\_calc(c :: c') \end{array}}{\mathcal{H} \vdash e : (ck)^n} \ \textsc{Efby}$$

$$\frac{\begin{array}{c} e = Earrow(expr, expr') \\ expr = e_1 :: e_2 :: ... :: e_n \qquad expr' = e'_1 :: e'_2 :: ... :: e'_n \\ \mathcal{H} \vdash \diamond \qquad \mathcal{H} \vdash e_1 : ck_1 \quad \mathcal{H} \vdash e_2 : ck_2 \quad ... \quad \mathcal{H} \vdash e_n : ck_n \\ \mathcal{H} \vdash e'_1 : ck'_1 \quad \mathcal{H} \vdash e'_2 : ck'_2 \quad ... \quad \mathcal{H} \vdash e'_n : ck'_n \\ c = clock\_check\_calc(ck_1 :: ck_2 :: ... :: ck_n) \\ c' = clock\_check\_calc(ck'_1 :: ck'_2 :: ... :: ck'_n) \\ ck = clock\_check\_calc(c :: c') \end{array}}{\mathcal{H} \vdash e : (ck)^n} \ \textsc{Earrow}$$

$$\frac{\begin{array}{c} e = Ewhen(expr, c) \qquad expr = e_1 :: e_2 :: ... :: e_n \qquad c = (b, x) \\ \mathcal{H} \vdash \diamond \qquad \mathcal{H} \vdash x : ck \qquad \mathcal{H} \vdash e_1 : ck'_1 \quad \mathcal{H} \vdash e_2 : ck'_2 \quad ... \quad \mathcal{H} \vdash e_n : ck'_n \\ ck' = clock\_check\_calc(ck'_1 :: ck'_2 :: ... :: ck'_n) \end{array}}{\mathcal{H} \vdash e : (ck' \ on \ ck)^n} \ \textsc{Ewhen}$$

$$\frac{\begin{array}{c} e = Emerge(id, expr, expr') \qquad \mathcal{H} \vdash \diamond \qquad \mathcal{H} \vdash id : ck \\ \mathcal{H} \vdash expr : ck \ on \ (TRUE, id) \rightarrow (\mathcal{H} \vdash expr' : ck \ on \ (FALSE, id) \vee \mathcal{H} \vdash expr' : ConstCK) \\ \mathcal{H} \vdash expr : ck \ on \ (FALSE, id) \rightarrow (\mathcal{H} \vdash expr' : ck \ on \ (TRUE, id) \vee \mathcal{H} \vdash expr' : ConstCK) \\ \mathcal{H} \vdash expr : ConstCK \rightarrow \\ (\mathcal{H} \vdash expr' : ck \ on \ (TRUE, id) \vee \mathcal{H} \vdash expr' : ck \ on \ (FALSE, id) \vee \mathcal{H} \vdash expr' : ConstCK) \end{array}}{\mathcal{H} \vdash e : ck} \ \textsc{Emerge}$$

$$\frac{\begin{array}{c} e = Emerge(id, merge\_case\_list) \\ merge\_case\_list = (TRUE, e_1) :: (FALSE, e_2) \qquad \mathcal{H} \vdash \diamond \\ \mathcal{H} \vdash id : ck \qquad \mathcal{H} \vdash e_1 : ck \ on \ (TRUE, id) \qquad \mathcal{H} \vdash e_2 : ck \ on \ (FALSE, id) \end{array}}{\mathcal{H} \vdash e : ck} \ \textsc{Emerge Bool}$$

$$\frac{\begin{array}{c} e = Emerge(id, merge\_case\_list) \qquad ident\_list = id_1 :: id_2 :: ... :: id_n \\ merge\_case\_list = (id_1, e_1) :: (id_2, e_2) :: ... :: (id_n, e_n) \\ \mathcal{H} \vdash \diamond \qquad \mathcal{H} \vdash id : ck \\ \mathcal{H} \vdash e_1 : ck \ on \ (id_1, id) \quad \mathcal{H} \vdash e_2 : ck \ on \ (id_2, id) \quad ... \quad \mathcal{H} \vdash e_n : ck \ on \ (id_n, id) \end{array}}{\mathcal{H} \vdash e : ck} \ \textsc{Emerge Enum}$$

$$\frac{\begin{array}{c} e = Ecurrent(expr) \qquad expr = e_1 :: e_2 :: ... :: e_n \\ \mathcal{H} \vdash \diamond \qquad \mathcal{H} \vdash e_1 : ck \ on \ c_1 \quad \mathcal{H} \vdash e_2 : ck \ on \ c_2 \quad ... \quad \mathcal{H} \vdash e_n : ck \ on \ c_n \\ c = clock\_check\_calc(c_1 :: c_2 :: ... :: c_n) \end{array}}{\mathcal{H} \vdash e : (ck)^n} \ \textsc{Ecurrent}$$

4.4.4 Call Expressions

$$
\frac{\begin{array}{c}
e = ApplyExpr(id, args\_list) \\
nd = (funcType, modifier, id, parameters, returns, body) \\
args\_list = e_1 :: e_2 :: ... :: e_n \\
parameters = d_1 :: d_2 :: ... :: d_n \qquad returns = r_1 :: r_2 :: ... :: r_m \\
\forall i : 1 \leq i \leq n.(d_i = VDecl(x_i, pt_i) \ \vee \ d_i = WhenVDecl(x_i, pt_i, c_i)) \\
\forall i : 1 \leq i \leq m.(r_i = VDecl(y_i, rt_i) \ \vee \ r_i = WhenVDecl(y_i, rt_i, c'_i) \qquad \mathcal{H} \vdash \diamond \\
\phi \vdash nd \qquad \mathcal{H}_{id} \vdash \diamond \qquad \mathcal{H} \vdash e_1 : ck_1 \quad \mathcal{H} \vdash e_2 : ck_2 \quad ... \quad \mathcal{H} \vdash e_n : ck_n \\
base\_clock = find\_fastest(ck_1, ck_2, ..., ck_n) \\
\forall i : 1 \leq i \leq n.(ck_i = base\_clock \rightarrow \mathcal{H}_{id} \vdash x_i : base) \\
\forall i : 1 \leq i \leq m.(\mathcal{H}_{id} \vdash y_i : base \rightarrow ck'_i = base\_clock) \\
\forall i, j : 1 \leq i, j \leq n. \forall c, b. \exists x', c'.(i \neq j \wedge c = (b, x_i) \wedge (d_j = WhenVDecl(x_j, pt_j, c) \rightarrow \\
(e_i = Evar(x', pt_i) \wedge c' = (b, x') \wedge ck_j = ck_i \ on \ c') \\
\forall i : 1 \leq i \leq n. \forall j : 1 \leq j \leq m. \forall c, b. \exists x', c'.(c = (b, x_i) \wedge (r_j = WhenVDecl(y_j, rt_j, c) \rightarrow \\
(e_i = Evar(x', pt_i) \wedge c' = (b, x') \wedge ck'_j = ck_i \ on \ c')
\end{array}}{\mathcal{H} \vdash e : ck'_1 :: ck'_2 :: ... :: ck'_m} \textsc{Call}
$$

4.4.5 Higher-Order Operations

$$\frac{\begin{array}{c} e = ApplyExpr(iter, expr\_list) \qquad iter = Iterator(Omap, id, IntConst(k)) \\ nd = (funcType, modifier, id, parameters, returns, body) \\ parameters = d_1 :: d_2 :: ... :: d_n \\ returns = r_1 :: r_2 :: ... :: r_m \qquad expr\_list = e_1 :: e_2 :: ... :: e_n \\ \mathcal{H}_{id} \vdash \diamond \qquad \forall i : 1 \leq i \leq n, \exists x_i, pt_i.(d_i = VDecl(x_i, pt_i) \wedge \mathcal{H}_{id} \vdash x_i : base) \\ \forall i : 1 \leq i \leq m, \exists x_i, rt_i.(r_i = VDecl(x_i, rt_i) \wedge \mathcal{H}_{id} \vdash x_i : base) \\ \mathcal{H} \vdash \diamond \qquad \forall i : 1 \leq i \leq n.(\mathcal{H} \vdash e_i : ck) \end{array}}{\mathcal{H} \vdash e : ck} \textsc{ Map Usr-Op}$$

$$\frac{\begin{array}{c} e = ApplyExpr(iter, expr) \qquad iter = Iterator(Ofill, nh, IntConst(k)) \\ nd = (funcType, modifier, id, parameters, returns, body) \\ parameters = VDecl(x, pt) :: nil \\ returns = r_1 :: r_2 :: ... :: r_m, where\ m \geq 2 \qquad \mathcal{H}_{id} \vdash \diamond \qquad \mathcal{H}_{id} \vdash x : base \\ \forall i : 1 \leq i \leq m, \exists x_i, rt_i.(r_i = VDecl(x_i, rt_i) \wedge \mathcal{H}_{id} \vdash x_i : base) \\ \mathcal{H} \vdash \diamond \qquad \mathcal{H} \vdash expr : ck) \end{array}}{\mathcal{H} \vdash e : ck} \textsc{ Fill Usr-Op}$$

$$\frac{\begin{array}{c} e = ApplyExpr(iter, expr\_list) \qquad iter = Iterator(Ored, nh, IntConst(k)) \\ nd = (funcType, modifier, id, parameters, returns, body) \\ parameters = d_1 :: d_2 :: ... :: d_n, where\ n \geq 2 \\ returns = VDecl(x, rt) :: nil \qquad expr\_list = e_1 :: e_2 :: ... :: e_n \\ \mathcal{H}_{id} \vdash \diamond \qquad \forall i : 1 \leq i \leq n, \exists x_i, pt_i.(d_i = VDecl(x_i, pt_i) \wedge \mathcal{H}_{id} \vdash x_i : base) \\ \mathcal{H}_{id} \vdash x : base \qquad \mathcal{H} \vdash \diamond \qquad \forall i : 1 \leq i \leq n.(\mathcal{H} \vdash e_i : ck) \end{array}}{\mathcal{H} \vdash e : ck} \textsc{ Red Usr-Op}$$

$$\frac{\begin{array}{c} e = ApplyExpr(iter, expr\_list) \\ iter = Iterator(Ofillred, nh, IntConst(k)) \\ nd = (funcType, modifier, id, parameters, returns, body) \\ parameters = d_1 :: d_2 :: ... :: d_n, where\ n \geq 2 \\ returns = r_1 :: r_2 :: ... :: r_m, where\ m \geq 2 \qquad expr\_list = e_1 :: e_2 :: ... :: e_n \\ \mathcal{H}_{id} \vdash \diamond \qquad \forall i : 1 \leq i \leq n, \exists x_i, pt_i.(d_i = VDecl(x_i, pt_i) \wedge \mathcal{H}_{id} \vdash x_i : base) \\ \forall i : 1 \leq i \leq m, \exists x_i, rt_i.(r_i = VDecl(x_i, rt_i) \wedge \mathcal{H}_{id} \vdash x_i : base) \\ \mathcal{H} \vdash \diamond \qquad \forall i : 1 \leq i \leq n.(\mathcal{H} \vdash e_i : ck) \end{array}}{\mathcal{H} \vdash e : ck} \textsc{ Fillred Usr-Op}$$

$$\frac{\begin{array}{c} e = ApplyExpr(iter, expr) \qquad iter = Iterator(Omap, un\_op, IntConst(k)) \\ un\_op \in \{Oshort, Oint, ..., Oneg\} \\ \mathcal{H} \vdash \diamond \qquad \mathcal{H} \vdash expr : ck' \qquad ck = clock\_check\_calc(ck') \end{array}}{\mathcal{H} \vdash e : ck} \textsc{Map PrefixUnOp}$$

$$\frac{\begin{array}{c} e = ApplyExpr(iter, expr_1 :: expr_2) \\ iter = Iterator(Omap, bin\_op, IntConst(k)) \\ bin\_op \in \{Oadd, Osub, ..., Oge\} \qquad \mathcal{H} \vdash \diamond \\ \mathcal{H} \vdash expr_1 : ck_1 \qquad \mathcal{H} \vdash expr_2 : ck_2 \qquad ck = clock\_check\_calc(ck_1 :: ck_2) \end{array}}{\mathcal{H} \vdash e : ck} \textsc{Map PrefixBinOp}$$

$$\frac{\begin{array}{c} e = ApplyExpr(iter, expr_1 :: expr_2) \\ iter = Iterator(Ored, bin\_op, IntConst(k)) \\ bin\_op \in \{Oadd, Osub, ..., Oge\} \qquad \mathcal{H} \vdash \diamond \\ \mathcal{H} \vdash expr_1 : ck_1 \qquad \mathcal{H} \vdash expr_2 : ck_2 \qquad ck = clock\_check\_calc(ck_1 :: ck_2) \end{array}}{\mathcal{H} \vdash e : ck} \textsc{Red PrefixBinOp}$$

$$\frac{\begin{array}{c} e = ApplyExpr(iter, expr_1 :: expr_2) \\ iter = Iterator(Ofillred, bin\_op, IntConst(k)) \\ bin\_op \in \{Oadd, Osub, ..., Oge\} \qquad \mathcal{H} \vdash \diamond \\ \mathcal{H} \vdash expr_1 : ck_1 \qquad \mathcal{H} \vdash expr_2 : ck_2 \qquad ck = clock\_check\_calc(ck_1 :: ck_2) \end{array}}{\mathcal{H} \vdash e : ck} \textsc{Fillred PrefixBinOp}$$

4.4.6 Array and Struct Expressions

$$\frac{\begin{array}{c} e = Econstruct(field\_expr\_list) \\ field\_expr\_list = (id_1, e_1) :: (id_2, e_2) :: ... :: (id_n, e_n) \\ \mathcal{H} \vdash \diamond \qquad \mathcal{H} \vdash e_1 : c_1 \quad \mathcal{H} \vdash e_2 : c_2 \quad ... \quad \mathcal{H} \vdash e_n : c_n \\ ck = clock\_check\_calc(c_1 :: c_2 :: ... :: c_n) \end{array}}{\mathcal{H} \vdash e : ck} \textsc{Construct}$$

$$\frac{\begin{array}{c} e = Earrayacc(expr, index) \qquad \mathcal{H} \vdash \diamond \\ \mathcal{H} \vdash expr : c_1 \qquad \mathcal{H} \vdash index : c_2 \qquad ck = clock\_check\_calc(c_1 :: c_2) \end{array}}{\mathcal{H} \vdash e : ck} \textsc{Array Acc}$$

$$\frac{e = Earraydef(expr, IntConst(n)) \qquad \mathcal{H} \vdash \diamond \qquad \mathcal{H} \vdash expr : ck}{\mathcal{H} \vdash e : ck} \textsc{Array Def}$$

$$\frac{\begin{array}{c} e = Earraydiff(expr\_list) \\ expr\_list = e_1 :: e_2 :: ... :: e_n \qquad \mathcal{H} \vdash \diamond \qquad \forall i : 1 \leq i \leq n.(\mathcal{H} \vdash e_i : ck_i) \\ ck = clock\_check\_calc(ck_1 :: ck_2 :: ... :: ck_n) \end{array}}{\mathcal{H} \vdash e : ck} \textsc{Array Diff}$$

$$\frac{\begin{array}{c} e = Earrayproj(expr_1, expr\_list, expr_2) \\ expr\_list = index_1 :: index_2 :: ... :: index_k \qquad \mathcal{H} \vdash \diamond \\ \mathcal{H} \vdash expr_1 : ck_1 \qquad \mathcal{H} \vdash expr_2 : ck_2 \qquad \forall i : 1 \leq i \leq n.(\mathcal{H} \vdash index_i : c_i) \\ ck = clock\_check\_calc(ck_1 :: ck_2 :: c_1 :: c_2 :: ... :: c_n) \end{array}}{\mathcal{H} \vdash e : ck} \textsc{Array Proj}$$

$$\frac{\begin{array}{c} e = Earrayslice(expr, expr_1, expr_2) \\ \mathcal{H} \vdash \diamond \qquad \mathcal{H} \vdash expr : ck' \qquad \mathcal{H} \vdash expr_1 : ck_1 \\ \mathcal{H} \vdash expr_2 : ck_2 \qquad ck = clock\_check\_calc(ck' :: c_1 :: c_2) \end{array}}{\mathcal{H} \vdash e : ck} \textsc{Array Slice}$$

$$\frac{\begin{array}{c} e = Emix(expr, label\_index\_list, expr_1) \\ label\_index\_list = li_1 :: li_2 :: ... :: li_n \qquad \mathcal{H} \vdash \diamond \\ \mathcal{H} \vdash expr : ck \qquad \forall i : 1 \leq i \leq n.(li_i \ is\ an\ array\ index \rightarrow \mathcal{H} \vdash li_i : ck_i) \\ \forall i : 1 \leq i \leq n.(li_i \ is\ a\ struct\ field\ id \rightarrow \mathcal{H} \vdash ck_i = ConstCK) \\ \mathcal{H} \vdash expr_1 : ck' \\ c = clock\_check\_calc(ck :: ck' :: ck_1 :: ck_2 :: c_1 :: c_2 :: ... :: c_n) \end{array}}{\mathcal{H} \vdash e : c} \textsc{Mix}$$

$$\frac{e = Efield(expr, ident) \qquad \mathcal{H} \vdash \diamond \qquad \mathcal{H} \vdash expr : ck}{\mathcal{H} \vdash e : ck} \textsc{Field}$$

4.4.7 Other Expressions or Sub-Expressions

$$\frac{e = Econst(atom\_c) \qquad \mathcal{H} \vdash \diamond}{\mathcal{H} \vdash e : ConstCK} \textsc{Const}$$

$$\frac{e = Evar(x) \qquad \mathcal{H} \vdash \diamond \qquad \mathcal{H} \vdash x : ck}{\mathcal{H} \vdash e : ck} \textsc{Var}$$

$$\frac{\begin{array}{c} e = ListExpr(e_1, e_2, ..., e_n) \\ \mathcal{H} \vdash \diamond \qquad \mathcal{H} \vdash e_1 : ck_1 \quad \mathcal{H} \vdash e_2 : ck_2 \quad ... \quad \mathcal{H} \vdash e_n : ck_n \\ ck = clock\_check\_calc(ck_1 :: ck_2 :: ... :: ck_n) \end{array}}{\mathcal{H} \vdash e : (ck)^n} \textsc{ListExpr}$$

$$\frac{\begin{array}{c} e = Eunop(un\_op, expr) \qquad un\_op \in \{Oshort, Oint, ..., Oneg\} \\ \mathcal{H} \vdash \diamond \qquad \mathcal{H} \vdash expr : ck' \qquad ck = clock\_check\_calc(ck') \end{array}}{\mathcal{H} \vdash e : ck} \textsc{UnOp}$$

$$\frac{\begin{array}{c} e = Ebinop(bin\_op, expr_1, expr_2) \\ bin\_op \in \{Oadd, Osub, ..., Oge\} \qquad \mathcal{H} \vdash \diamond \\ \mathcal{H} \vdash expr_1 : ck_1 \qquad \mathcal{H} \vdash expr_2 : ck_2 \qquad ck = clock\_check\_calc(ck_1 :: ck_2) \end{array}}{\mathcal{H} \vdash e : ck} \textsc{BinOp}$$

$$\frac{\begin{array}{c} e = Eif(expr, expr', expr'') \\ expr' = e'_1 :: e'_2 :: ... :: e'_n \qquad expr'' = e''_1 :: e''_2 :: ... :: e''_n \\ \mathcal{H} \vdash \diamond \qquad \mathcal{H} \vdash expr : c \qquad \mathcal{H} \vdash e'_1 : ck'_1 \quad \mathcal{H} \vdash e'_2 : ck'_2 \quad ... \quad \mathcal{H} \vdash e'_n : ck'_n \\ \mathcal{H} \vdash e''_1 : ck''_1 \quad \mathcal{H} \vdash e''_2 : ck''_2 \quad ... \quad \mathcal{H} \vdash e''_n : ck''_n \\ ck' = clock\_check\_calc(ck'_1 :: ck'_2 :: ... :: ck'_n) \\ ck'' = clock\_check\_calc(ck''_1 :: ck''_2 :: ... :: ck''_n) \\ ck = clock\_check\_calc(c :: ck' :: ck'') \end{array}}{\mathcal{H} \vdash e : (ck)^n} \textsc{If}$$

$$\frac{\begin{array}{c} e = Ecase(expr, p\_list) \\ p\_list = (p_1, e_1) :: (p_2, e_2) :: ... :: (p_n, e_n) \qquad \mathcal{H} \vdash \diamond \\ \mathcal{H} \vdash expr : ck' \qquad \forall i : 1 \leq i \leq n.\ (\mathcal{H} \vdash p_i : ConstCK \vee p_i = PatternAny) \\ \forall i : 1 \leq i \leq n.\ (\mathcal{H} \vdash e_i : ck_i) \\ ck = clock\_check\_calc(ck' :: ck_1 :: ck_2 :: ... :: ck_n) \end{array}}{\mathcal{H} \vdash e : ck} \textsc{Case}$$

$$\frac{\begin{array}{c} e = Eboolred(IntConst(i), IntConst(j), expr) \qquad expr = e_1 :: e_2 :: ... :: e_k \\ 0 \leq i \leq j \leq k \wedge k > 0 \qquad \mathcal{H} \vdash \diamond \qquad \forall l : 1 \leq l \leq k.\ (\mathcal{G}, \mathcal{T}, \mathcal{F} \vdash e_l : ck_i) \\ ck = clock\_check\_calc(ck_1 :: ck_2 :: ... :: ck_k) \end{array}}{\mathcal{H} \vdash e : ck} \textsc{ Boolred}$$

$$\frac{\begin{array}{c} e = Ediese(expr) \qquad expr = e_1 :: e_2 :: ... :: e_n \qquad \mathcal{H} \vdash \diamond \\ \forall i : 1 \leq i \leq n.\ (\mathcal{H} \vdash e_i : ck_i) \qquad ck = clock\_check\_calc(ck_1 :: ck_2 :: ... :: ck_n) \end{array}}{\mathcal{H} \vdash e : ck} \textsc{ Diese}$$

$$\frac{\begin{array}{c} e = EnorS(expr) \qquad expr = e_1 :: e_2 :: ... :: e_n \qquad \mathcal{H} \vdash \diamond \\ \forall i : 1 \leq i \leq n.\ (\mathcal{H} \vdash e_i : ck_i) \qquad ck = clock\_check\_calc(ck_1 :: ck_2 :: ... :: ck_n) \end{array}}{\mathcal{H} \vdash e : ck} \textsc{ Nor}$$

4.4.8 Equations

$$\frac{\begin{array}{c} eq = Equation(x, expr) \\ \mathcal{H}' \vdash \diamond \qquad x \notin dom(\mathcal{H}') \qquad \mathcal{H}' \vdash expr : x\_ck \qquad \mathcal{H} = \mathcal{H}' \cup \{x : x\_ck\} \end{array}}{\mathcal{H} \vdash eq} \textsc{ Usual Equation Sig1}$$

$$\frac{eq = Equation(x, expr) \qquad \mathcal{H} \vdash \diamond \qquad \mathcal{H} \vdash x : x\_ck \qquad \mathcal{H} \vdash expr : x\_ck}{\mathcal{H} \vdash eq} \textsc{ Usual Equation Sig2}$$

$$\frac{\begin{array}{c} eq = Equation(lhs, exprs) \\ lhs = lh :: lhs' \qquad exprs = expr :: exprs' \qquad eq' = Equation(lh, expr) \\ eq'' = Equation(lhs', exprs') \qquad \mathcal{H} \vdash \diamond \qquad \mathcal{H} \vdash eq' \qquad \mathcal{H} \vdash eq'' \end{array}}{\mathcal{H} \vdash eq} \textsc{ Usual Equation Com}$$

$$\frac{\begin{array}{c} eq = (lhs, ApplyExpr(op, args\_list)) \\ \mathcal{H}' \vdash \diamond \qquad \mathcal{H}' \vdash ApplyExpr(op, args\_list) : ck_1 :: ck_2 :: ... :: ck_m \\ lhs = x_1 :: x_2 :: ... :: x_m \qquad \forall i : 1 \leq i \leq m. \forall ck. (\mathcal{H}' \vdash x_i : ck \rightarrow ck = ck_i) \\ \mathcal{H} = \mathcal{H}' \cup \{x_i : ck_i | x_i \notin dom(\mathcal{H}'), 1 \leq i \leq m\} \end{array}}{\mathcal{H} \vdash eq} \textsc{ Apply Equation}$$

4.4.9 Equation List, Nodes and Programs

$$\frac{eqs = eq :: eqs' \qquad \mathcal{H} \vdash \diamond \qquad \mathcal{H} \vdash eq \qquad \mathcal{H} \vdash eqs'}{\mathcal{H} \vdash eqs}\ \textsc{EquationList}$$

$$\frac{\begin{array}{c} nd = (funcType, modifier, id, parameters, returns, body) \\ body = (locals, eq\_list) \qquad \mathcal{H}_{id} \vdash \diamond \qquad \mathcal{H}_{id} \vdash eq\_list \end{array}}{\phi \vdash nd}\ \textsc{Node}$$

$$\frac{\forall\ nd \in p.\ (nd\ is\ a\ node\ declaration\ in\ p \rightarrow \phi \vdash nd}{\phi \vdash p}\ \textsc{Program}$$