1.5 Static typing

As in Zélus, we must statically discriminate between *discrete* and *continuous* computations. In ZSy, the transition between *continuous* and *discrete* contexts is realized via signals emitted by the guards. A variable is typed *discrete* if it is activated on signal emissions, and *continuous* otherwise. We can thus adapt the Zélus type system presented in [BBCP11a, §3.2] to ZSy.

Types and kinds

Each function has a type of the form $t_1 \xrightarrow{k} t_2$ where k is a *kind* with three possible values: C denotes continuous functions that can only be used in continuous contexts, D denotes discrete functions that must be activated on the emission of a signal, A denotes a function that can be used in any context. The subkind relation $\subseteq$ is defined as $\forall k,\ k \subseteq k$ and $\mathrm{A} \subseteq k$. The type language is:

$$
\begin{array}{lcl}
t & ::= & t \times t \ \mid\ \alpha \ \mid\ bt \\
k & ::= & \mathrm{D} \ \mid\ \mathrm{C} \ \mid\ \mathrm{A} \\
bt & ::= & \texttt{int} \ \mid\ \texttt{bool} \ \mid\ \texttt{signal} \ \mid\ \texttt{timer} \\
\sigma & ::= & \forall \alpha_1, ..., \alpha_n.\ t \xrightarrow{k} t
\end{array}
$$

A type (t) can be a pair ($t \times t$), a type variable (α) or a base type (bt). The base types are `int` and `bool` for constants, `signal` for signals emitted by guards, and `timer` for timer variables. Timers have a particular type to prevent their concrete values being used in an expression. Functions are associated to a type scheme σ where type variables are generalized.

A global environment G tracks the type schemes of functions, and another environment H assigns types to variables. We write $x : t$ to state that x is of type t, and if H_1 and H_2 are two environments, $H_1 + H_2$ denotes their union, provided their domains are disjoint.

Generalization and instantiation Type schemes are obtained by generalizing the free variables in function types $t_1 \xrightarrow{k} t_2$:

$$
gen(t_1 \xrightarrow{k} t_2) = \forall \alpha_1, \ldots, \alpha_n.\ t_1 \xrightarrow{k} t_2 \text{ where } \{\alpha_1, \ldots, \alpha_n\} = \mathit{ftv}(t_1 \xrightarrow{k} t_2),
$$

where $\mathit{ftv}(t)$ denotes the set of free type variables in type t.

A type scheme can be instantiated by substituting type variables with actual types. $\mathit{Inst}(\sigma)$ denotes the set of possible instantiations of a type scheme σ. The kind of a type $t_1 \xrightarrow{k} t_2$ can be instantiated with any kind k' where $k \subseteq k'$:

$$
\frac{k \subseteq k'}{(t \xrightarrow{k'} t')[t_1/\alpha_1, ..., t_n/\alpha_n] \in \mathit{Inst}(\forall \alpha_1, ..., \alpha_n.\ t \xrightarrow{k} t')}
$$

Typing rules

Typing is defined by four judgments which resemble those of Zélus:

(TYP-EXP) $G, H \vdash_k e : t$	(TYP-ENV) $G, H \vdash_k E : H'$
(TYP-PAT) $\vdash_{pat} p : t, H$	(TYP-HANDLER) $G, H \vdash_k h : t$

The judgment (TYP-EXP) states that in environments G and H, expression e has kind k and type t. The judgment (TYP-ENV) states that in environments G and H, a set of equations E has kind k and produces the type environment H'. The judgment (TYP-PAT) states that a pattern p has type t and defines a type environment H. The judgment (TYP-HANDLER) states that in environments G and H, the value defined by a handler h has type t and kind k.

We add a fifth judgment:

$$\begin{array}{c}\text{(CHECK-ZONE)}\\ G, H \vdash_{zone} c\end{array}$$

to check if a constraint defines a valid zone. In particular, (CHECK-ZONE) requires that the definition of zones only involve timer differences and integer bounds.

The initial environment G_0 contains the type of primitive operators, like `fby`, and imported operators, like (`+`) and (`=`).

$$\begin{array}{lcl}(\texttt{+}) & : & \texttt{int} \times \texttt{int} \xrightarrow{\mathrm{A}} \texttt{int}\\ (\texttt{=}) & : & \forall \alpha,\ \alpha \times \alpha \xrightarrow{\mathrm{A}} \texttt{bool}\\ \texttt{fby} & : & \forall \alpha,\ \alpha \times \alpha \xrightarrow{\mathrm{D}} \alpha\end{array}$$

Imported operators have kind A since they can be used in any context. The unit delay `fby` has kind D since it is only allowed in discrete contexts.

The typing rules are presented in figure 1.7.

(EQ) An equation x `=` e is well-typed if the types of x and e coincide. The kind of the equation is the kind of e.

(AND) The parallel composition of two sets of equations E_1 `and` E_2 is well-typed if both E_1 and E_2 are well-typed. The kind must be the same for E_1 and E_2.

(PRESENT) The equation x `=` `present` h `init` e_0 activates at instants defined by the handler h. The equation is well-typed if the handler is well-typed and produces a value of type t that coincides with the type of the initialization expression e_0. This equation can be used in continuous and discrete contexts depending on the handler. In any case, the initialization value must be of kind D, even in continuous contexts.

(PRESENT-ELSE) When a default value is provided it must also have the type t returned by the handler h and the same kind. In particular, in continuous contexts the default value is not guarded by a signal and must thus have kind C.

(TIMER) The equation `timer` x `init` e_0 `reset` h defines a variable of type `timer`. This equation is well-typed if the reset handler h is well-typed and returns a value of type `int`, and if the initialization expression is also of type `int`. The reset handler must have kind C and the overall kind is C. Timers can only be defined in continuous contexts.

(ALWAYS) The equation `always { ` c ` }` introduces an invariant and does not define a variable. This equation is well-typed if it has kind C and if the constraint c is a valid zone. Invariants are only allowed in continuous contexts.

(GUARD) The equation `emit` s `when { ` c ` }` is well-typed if constraint c defines a valid zone. Variable s is then of type `signal` and the overall kind is C. Guards are only allowed in continuous contexts.

$$\text{(EQ)}\quad \frac{G, H \vdash_k e : t}{G, H \vdash_k x \texttt{ = } e : [x : t]}$$

$$\text{(AND)}\quad \frac{G, H \vdash_k E_1 : H_1 \qquad G, H \vdash_k E_2 : H_2}{G, H \vdash_k E_1 \texttt{ and } E_2 : H_1 + H_2}$$

$$\text{(PRESENT)}\quad \frac{G, H \vdash_k h : t \qquad G, H \vdash_{\mathrm{D}} e_0 : t}{G, H \vdash_k x \texttt{ = present } h \texttt{ init } e_0 : [x : t]}$$

$$\text{(PRESENT-ELSE)}\quad \frac{G, H \vdash_k h : t \qquad G, H \vdash_k e : t}{G, H \vdash_k x \texttt{ = present } h \texttt{ else } e : [x : t]}$$

$$\text{(TIMER)}\quad \frac{G, H \vdash_{\mathrm{D}} e_0 : \texttt{int} \qquad G, H \vdash_{\mathrm{C}} h : \texttt{int}}{G, H \vdash_{\mathrm{C}} \texttt{timer } x \texttt{ init } e_0 \texttt{ reset } h : [x : \texttt{timer}]}$$

$$\text{(ALWAYS)}\quad \frac{G, H \vdash_{zone} c}{G, H \vdash_{\mathrm{C}} \texttt{always \{ } c \texttt{ \}} : []}$$

$$\text{(GUARD)}\quad \frac{G, H \vdash_{zone} c}{G, H \vdash_{\mathrm{C}} \texttt{emit } s \texttt{ when \{ } c \texttt{ \}} : [s : \texttt{signal}]}$$

$$\text{(CONST)}\quad G, H \vdash_k 42 : \texttt{int}$$

$$\text{(VAR)}\quad G, H + [x : t] \vdash_k x : t$$

$$\text{(PAIR)}\quad \frac{G, H \vdash_k e_1 : t_1 \qquad H \vdash_k e_2 : t_2}{G, H \vdash_k (e_1, e_2) : t_1 \times t_2}$$

$$\text{(APP)}\quad \frac{t \xrightarrow{k} t' \in \mathit{Inst}(G(f)) \qquad G, H \vdash_k e : t}{G, H \vdash_k f(e) : t'}$$

$$\text{(WHERE-REC)}\quad \frac{G,\ H \vdash_k E : H_e \qquad G,\ H + H_e \vdash_k e : t}{G, H \vdash_k e \texttt{ where rec } E : t}$$

$$\text{(DEF-HYBRID)}\quad \frac{\vdash_{pat} p : t_1, H_p \qquad G, H_p \vdash_{\mathrm{C}} e : t_2}{G \vdash \texttt{let hybrid } f(p) \texttt{ = } e : [f : \mathit{gen}(t_1 \xrightarrow{\mathrm{C}} t_2)]}$$

$$\text{(DEF-NODE)}\quad \frac{\vdash_{pat} p : t_1, H_p \qquad G, H_p \vdash_{\mathrm{D}} e : t_2}{G \vdash \texttt{let node } f(p) \texttt{ = } e : [f : \mathit{gen}(t_1 \xrightarrow{\mathrm{D}} t_2)]}$$

$$\text{(DEF-ANY)}\quad \frac{\vdash_{pat} p : t_1, H_p \qquad G, H_p \vdash_{\mathrm{A}} e : t_2}{G \vdash \texttt{let } f(p) \texttt{ = } e : [f : \mathit{gen}(t_1 \xrightarrow{\mathrm{A}} t_2)]}$$

$$\text{(DEF-SEQ)}\quad \frac{G \vdash d_1 : G_1 \qquad G + G_1 \vdash d_2 : G_2}{G \vdash d_1\ d_2 : G_1 + G_2}$$

$$\text{(PAT-VAR)}\quad \vdash_{pat} x : t, [x : t]$$

$$\text{(PAT-PAIR)}\quad \frac{\vdash_{pat} p_1 : t_1, H_1 \qquad \vdash_{pat} p_2 : t_2, H_2}{\vdash_{pat} (p_1,\ p_2) : t_1 \times t_2, H_1 + H_2}$$

$$\text{(HANDLER-C)}\quad \frac{\forall i \in \{1, ..., n\} \qquad G, H \vdash_{\mathrm{D}} e_i : t \qquad G, H \vdash_{\mathrm{C}} c_i : \texttt{signal}}{G, H \vdash_{\mathrm{C}} c_1 \ \rightarrow\ e_1 \ \texttt{|}\ \cdots\ \texttt{|}\ c_n \ \rightarrow e_n : t}$$

$$\text{(HANDLER-D)}\quad \frac{\forall i \in \{1, .., n\} \qquad G, H \vdash_{\mathrm{D}} e_i : t \qquad G, H \vdash_{\mathrm{D}} c_i : \texttt{bool}}{G, H \vdash_{\mathrm{D}} c_1 \ \rightarrow\ e_1 \ \texttt{|}\ \cdots\ \texttt{|}\ c_n \ \rightarrow e_n : t}$$

$$\text{(ZONE-VAR)}\quad \frac{G, H \vdash_{\mathrm{C}} t : \texttt{timer} \qquad G, H \vdash_{\mathrm{C}} e : \texttt{int}}{G, H \vdash_{zone} t \sim e}$$

$$\text{(ZONE-DIFF)}\quad \frac{G, H \vdash_{\mathrm{C}} t_1 : \texttt{timer} \qquad G, H \vdash_{\mathrm{C}} t_2 : \texttt{timer} \qquad G, H \vdash_{\mathrm{C}} e : \texttt{int}}{G, H \vdash_{zone} t_1 - t_2 \sim e}$$

$$\text{(ZONE-AND)}\quad \frac{G, H \vdash_{zone} c_1 \qquad G, H \vdash_{zone} c_2}{G, H \vdash_{zone} c_1 \texttt{ \&\& } c_2}$$

Figure 1.7: The typing rules.

(const) The typing of constants is illustrated with the integer constant 42. Constants can be used in any context.

(var) A variable of type t can be used in any context.

(pair) A pair (e_1, e_2) is of type $t_1 \times t_2$ if e_1 has type t_1 and e_2 has type t_2; e_1 and e_2 must have the same kind.

(app) An application $f(e)$ is of type t' if e has type t and if $t \xrightarrow{k} t'$ is a valid instantiation of the type scheme of f. The kind of the application $f(e)$ is given by the kind of f.

(where-rec) A local definition e `where rec` E is well-typed if the set of equations E is well-typed and expression e is well-typed in the extended environment.

(def-hybrid) (def-node) (def-any) A function definition has type $t_1 \xrightarrow{k} t_2$ if the input pattern p has type t_1 and the defining expression has type t_2. Function types are generalized. The kind is given in the definition: `hybrid` for C, `node` for D, nothing for A.

(def-seq) Function definitions are typed sequentially.

(pat-var) (pat-pair) Patterns return an environment containing the types of their variables.

(handler-c) (handler-d) A handler $c_1 \rightarrow e_1 \mid \cdots \mid c_n \rightarrow e_n$ is well-typed if all expressions e_i have the same type t and conditions c_i have type `signal` in continuous contexts or `bool` in discrete contexts. The expressions e_i must have kind D since, in any case, they are only activated at discrete instants.

(zone-var) (zone-diff) (zone-and) A constraint c defines a valid zone if variables are of type `timer` and bounds are of kind C and type `int`; the values are thus piecewise constant and can only change on signal emissions.

Since expressions of kind A can be executed in any context we also have the following subtyping property:

Property 1.1 (Subtyping)**.**

$$G, H \vdash_{\mathrm{A}} e : t \implies (G, H \vdash_{\mathrm{C}} e : t) \wedge (G, H \vdash_{\mathrm{D}} e : t)$$

Proof. By induction on the typing derivation of $G, H \vdash_{\mathrm{A}} e : t$. □