HoCL Manual - 1.0a

J. Sérot

Contents

1 Syntax **2**

Chapter 1

Syntax

This appendix gives a BNF definition of the concrete syntax for HoCL programs. The meta-syntax is conventional. Keywords are written in **bold** and non-terminals like ⟨this⟩. Vertical bars `|` are used to indicate alternatives. Constructs enclosed in brackets `[` ... `]` are optional. The notation ϵ denotes an empty construct. The notation E_s^* (resp. E_s^+) denotes a list of zero (resp. one) or more elements E separated by s. Value-attributed terminals are denoted like *this*. The terminals *infix3*, *infix2* and *infix0* respectively correspond to infix operators {`*`,`/`,`%`}, {`+`,`-`} and {`@@`, `|>`, `|->`}. The other definitions (*ident*, *int*, *string*) are classical.

⟨program⟩	::=	⟨decl⟩*
⟨decl⟩	::=	⟨type_decl⟩ **;**
	|	⟨value_decl⟩ **;**
	|	⟨node_decl⟩ **;**
	|	⟨graph_decl⟩ **;**
⟨type_decl⟩	::=	**type** *ident*
⟨value_decl⟩	::=	**val** [**rec**] ⟨net_binding⟩
⟨node_decl⟩	::=	⟨node_intf⟩ ⟨node_impl⟩
⟨node_intf⟩	::=	**node** *ident* [⟨node_params⟩] **in** ⟨io_decls⟩ **out** ⟨io_decls⟩
⟨node_impl⟩	::=	ϵ
	|	**actor** ⟨actor_desc⟩* **end**
	|	**struct** ⟨struct_graph_desc⟩ **end**
	|	**fun** ⟨fun_graph_desc⟩ **end**
⟨actor_desc⟩	::=	*ident* **(** ⟨impl_attr⟩$^*_{,}$ **)**
⟨impl_attr⟩	::=	*ident* = *string*
	|	*ident*
⟨node_params⟩	::=	**param** **(** ⟨node_param_decl⟩$^*_{,}$ **)**

⟨node_param_decl⟩	::=	*ident* **:** ⟨simple_type_expr⟩
⟨io_decls⟩	::=	**(** ⟨io_decl⟩$^*_{,}$ **)**
⟨io_decl⟩	::=	*ident* **:** ⟨simple_type_expr⟩ ⟨opt_io_annots⟩
⟨opt_io_annots⟩	::=	ϵ
	|	[⟨core_expr⟩]
	|	**{** ⟨io_annot⟩$^*_{,}$ **}**
⟨io_annot⟩	::=	*ident* **=** *string*
⟨core_expr⟩	::=	⟨simple_core_expr⟩
	|	⟨core_expr⟩ *infix3* ⟨core_expr⟩
	|	⟨core_expr⟩ *infix2* ⟨core_expr⟩
	|	⟨core_expr⟩ $*$ ⟨core_expr⟩
⟨simple_core_expr⟩	::=	*ident*
	|	*int*
	|	**true**
	|	**false**
	|	**(** ⟨core_expr⟩ **)**
⟨simple_type_expr⟩	::=	*ident*
	|	**int**
	|	**bool**
⟨graph_decl⟩	::=	**graph** *ident* [⟨graph_params⟩] **in** ⟨io_decls⟩ **out** ⟨io_decls⟩ ⟨graph_defn⟩
⟨graph_params⟩	::=	**param (** ⟨graph_param_value⟩$^*_{;}$ **)**
⟨graph_param_value⟩	::=	*ident* **:** ⟨simple_type_expr⟩ **=** ⟨const_param_value⟩
⟨const_param_value⟩	::=	*int*
	|	**true**
	|	**false**
⟨graph_defn⟩	::=	**struct** ⟨struct_graph_desc⟩ **end**
	|	**fun** ⟨fun_graph_desc⟩ **end**
⟨struct_graph_desc⟩	::=	⟨struct_defn⟩*
⟨struct_defn⟩	::=	⟨gwire_defn⟩
	|	⟨gnode_defn⟩
⟨gwire_defn⟩	::=	**wire** *ident*$^*_{,}$ **:** ⟨simple_type_expr⟩
⟨gnode_defn⟩	::=	**node** *ident* **:** *ident* [⟨gnode_params⟩] ⟨gnode_ios⟩ ⟨gnode_ios⟩

$$\begin{array}{rcl}
\langle\text{gnode\_params}\rangle & ::= & < \langle\text{core\_expr}\rangle^{*}_{,} > \\[1em]
\langle\text{gnode\_ios}\rangle & ::= & (\ \langle\text{gnode\_io}\rangle^{*}_{,}\ ) \\[1em]
\langle\text{gnode\_io}\rangle & ::= & \textit{ident} \\[1em]
\langle\text{fun\_graph\_desc}\rangle & ::= & \langle\text{net\_defn}\rangle^{*} \\[1em]
\langle\text{net\_defn}\rangle & ::= & \textbf{val}\ [\textbf{rec}]\ \langle\text{net\_binding}\rangle^{+}_{\textbf{and}} \\[1em]
\langle\text{net\_binding}\rangle & ::= & \langle\text{net\_pattern}\rangle = \langle\text{net\_expr}\rangle \\
 & | & \langle\text{net\_binding\_name}\rangle\ \langle\text{simple\_net\_pattern}\rangle^{+} = \langle\text{net\_expr}\rangle \\[1em]
\langle\text{net\_binding\_name}\rangle & ::= & \textit{ident} \\
 & | & (\ \textit{infix0}\ ) \\[1em]
\langle\text{net\_expr}\rangle & ::= & \langle\text{simple\_net\_expr}\rangle \\
 & | & \langle\text{simple\_net\_expr}\rangle\ \langle\text{simple\_net\_expr}\rangle^{+} \\
 & | & \langle\text{net\_expr\_comma\_list}\rangle \\
 & | & \langle\text{net\_expr}\rangle\ \textbf{::}\ \langle\text{net\_expr}\rangle \\
 & | & \langle\text{simple\_net\_expr}\rangle\ [\ \langle\text{simple\_net\_expr}\rangle\ ] \\
 & | & \textbf{let}\ [\textbf{rec}]\ \langle\text{net\_binding}\rangle^{+}_{\textbf{and}}\ \textbf{in}\ \langle\text{net\_expr}\rangle \\
 & | & \textbf{fun}\ \langle\text{net\_pattern}\rangle \rightarrow \langle\text{net\_expr}\rangle \\
 & | & \textbf{match}\ \langle\text{net\_expr}\rangle\ \textbf{with}\ \langle\text{net\_case}\rangle^{+}_{|} \\
 & | & \textbf{if}\ \langle\text{net\_expr}\rangle\ \textbf{then}\ \langle\text{net\_expr}\rangle\ \textbf{else}\ \langle\text{net\_expr}\rangle \\
 & | & \langle\text{net\_expr}\rangle\ \textit{infix3}\ \langle\text{net\_expr}\rangle \\
 & | & \langle\text{net\_expr}\rangle\ \textit{infix2}\ \langle\text{net\_expr}\rangle \\
 & | & \langle\text{net\_expr}\rangle\ \textit{infix0}\ \langle\text{net\_expr}\rangle \\
 & | & \langle\text{net\_expr}\rangle > \langle\text{net\_expr}\rangle \\
 & | & \langle\text{net\_expr}\rangle < \langle\text{net\_expr}\rangle \\
 & | & \langle\text{net\_expr}\rangle * \langle\text{net\_expr}\rangle \\
 & | & \langle\text{net\_expr}\rangle = \langle\text{net\_expr}\rangle \\
 & | & \langle\text{net\_expr}\rangle \neq \langle\text{net\_expr}\rangle \\[1em]
\langle\text{simple\_net\_expr}\rangle & ::= & \textit{ident} \\
 & | & \textit{ident} < \langle\text{core\_expr}\rangle^{+}_{,} > \\
 & | & (\,) \\
 & | & [\ \langle\text{net\_expr\_comma\_list}\rangle\ ] \\
 & | & [\,] \\
 & | & \textit{int} \\
 & | & \textbf{true} \\
 & | & \textbf{false} \\
 & | & (\ \langle\text{net\_expr}\rangle\ ) \\[1em]
\langle\text{net\_expr\_comma\_list}\rangle & ::= & \langle\text{net\_expr\_comma\_list}\rangle\ \textbf{,}\ \langle\text{net\_expr}\rangle \\
 & | & \langle\text{net\_expr}\rangle\ \textbf{,}\ \langle\text{net\_expr}\rangle \\[1em]
\langle\text{net\_case}\rangle & ::= & \langle\text{net\_pattern}\rangle \rightarrow \langle\text{net\_expr}\rangle
\end{array}$$

```
          ⟨net_pattern⟩  ::=  ⟨simple_net_pattern⟩
                           |  ⟨net_pattern_comma_list⟩
                           |  ⟨net_pattern⟩ :: ⟨net_pattern⟩
                           |  [ ⟨net_pattern_comma_list⟩ ]

   ⟨simple_net_pattern⟩  ::=  ident
                           |  _
                           |  ( ⟨net_pattern⟩ )
                           |  [ ]
                           |  ( )

⟨net_pattern_comma_list⟩  ::=  ⟨net_pattern_comma_list⟩ , ⟨net_pattern⟩
                           |  ⟨net_pattern⟩ , ⟨net_pattern⟩
```

Notes

- a `node_decl` with an empty `node_impl`, such as

  ```
  node foo in (...) out (...);
  ```

  is equivalent to

  ```
  node foo in (...) out (...)
  actor
  end;
  ```

  Both define opaque actors (viewed as black boxes).

- the description attached to non-opaque actors is a list of backend-specific descriptors. Each descriptor gives the name of the target backend and a list of (*attribute*, *value*) pair. Ex :

  ```
  node foo in (i: int) (o: int)
  actor
    preesm(loop_fn="foo_c", incl_file="foo.h")
  end;
  ```

- annotations can be attached to node inputs and outputs by appending them between braces; each annotation is a (*name*, *value*) pair. For example (for an SDF actor) :

  ```
    actor foo in (i: int {rate="k"}) out (o: bool {rate="k*2"}) ...
  ```

  Rate annotation can be abbreviated using the `[...]` syntax. For example, the previous example can be written as :

  ```
    actor foo in (i: int[k]) out (o: bool[k*2]) ...
  ```