csp2hc - Rules

Marcel Oliveira[1] and Jim Woodcock[2]

[1] Departamento de Informática e Matemática Aplicada, UFRN, Brazil
[2] Department of Computer Science, University of York, UK

1 Global Environments, Functions, Variables and Constants

- Environments
  - Processes information
    - *ProcArgsEnv*
      - Type: $String \rightarrow\!\!\!\!\!\!+\; \mathrm{seq}\, Argument$
      - Description: Maps processes names to the names of their arguments
    - *ProcDefsEnv*
      - Type: $String \rightarrow\!\!\!\!\!\!+\; ProcBody$
      - Description: Maps processes names to their bodies
    - *MainProc*
      - Type: *ProcBody*
      - Description: declares the main behaviour of the system
  - Types information
    - *TypeBranchesEnv*
      - Type: $String \rightarrow\!\!\!\!\!\!+\; \mathrm{seq}\, String$
      - Description: Maps type names to constants that represents the values that variables of each type may assume. For example, if we have the two datatypes in the CSP specification

        $datatype\, LETTER = A \mid B \mid C$
        $datatype\, ID = Alpha.LETTER \mid Unknown$

        we get the following *TypeBranchesEnv* environment

        $LETTER \mapsto \langle C, B, A \rangle$
        $ID \mapsto \langle Unknown, Alpha\_C, Alpha\_B, Alpha\_A \rangle$

    - *TagsParentEnv*
      - Type: $String \rightarrow\!\!\!\!\!\!+\; String$
      - Description: Maps constructor (tag) names to the name of the type whose declaration used it. For the example above, we have the following *TagsParentEnv* environment

        $Alpha \mapsto ID$

- $TagTypesEnv$
  - Type: $String \nrightarrow \operatorname{seq} String$
  - Description: Maps constructor (tag) names to a sequence of types. For the example above, we have the following $TagsParentEnv$ environment

$$Alpha \mapsto \langle LETTER \rangle$$

- $TagValuesEnv$
  - Type: $String \nrightarrow \operatorname{seq} String$
  - Description: Maps constructor (tag) names to a list of list of possible values for each type used in its declaration.

$$Alpha \mapsto \langle \langle C, B, A \rangle \rangle$$

- $ArgTypeEnv$
  - Type: $String \nrightarrow (String \nrightarrow String)$
  - Description: For each process name, returns a function that maps the name of the process arguments to its Handel-C type.
- Constants and functions information
  - $ConstFunArgsEnv$
    - Type: $String \nrightarrow \operatorname{seq} String$
    - Description: Maps the function names into the list of its arguments names (constants names are mapped into an empty list)
- Channels information
  - $CTypeEnv$
    - Type: $String \nrightarrow \operatorname{seq} String$
    - Description: Maps channel names into the list of types used in their respective declarations
  - $CCommEnv$
    - Type: $\operatorname{seq} String$
    - Description: List containing the names of the channels that communicate values using either ! or ?
  - $CProjEnv$
    - Type: $\operatorname{seq} String$
    - Description: List containing the names of the channels that communicate values using projections $c.*$
  - $CInOut$
    - Type: $String \nrightarrow String \nrightarrow \{IN, OUT\}$
    - Description: For each process name, returns a function that yields the usage of each channel name.
    - Example: If a channel c is an input channel within process P, then we have that $\{P \mapsto \{c \mapsto IN\}\}$ is a member of $CInOut$.

– Functions
  - $Bits(n)$:
    - Description: returns the number of bits needed to represent n values.
  - $Bitwise(total, on)$

* Description: returns a string with *total* characters, in which all characters are 0, but the *on*-th character is 1. If *on* is negative, all the characters are 0.
* Example:

$$\begin{array}{l} \mathit{Bitwise}(4,3) = \texttt{1000} \\ \mathit{Bitwise}(4,0) = \texttt{0001} \\ \mathit{Bitwise}(4,-1) = \texttt{0000} \end{array}$$

- $\Pi(lls)$
  * Description: distributed cartesian product of a list of lists.
- $\mathit{OrderedIntRange}(n)$
  * Description: returns a list of integers that can be represented using n bits ordered by their representation as unsigned integers.
  * Example:

$$\mathit{OrderedIntRange}(4) = \langle 0,1,2,3,4,5,6,7,-8,-7,-6,-5,-4,-3,-2,-1 \rangle$$

- $\mathit{IntName}(n)$
  * Description: returns the name of the constant that represents an integer integers.
  * Example:

$$\begin{array}{l} \mathit{IntName}(0) = \texttt{integer\_0} \\ \mathit{IntName}(-1) = \texttt{integer\_neg\_1} \end{array}$$

- $\mathit{Separate}(sep, lst)$
  * Description: returns the string containing the strings in *lst* separated by *sep*.
  * Example:

$$\mathit{Separate}(\mathsf{|}, \langle C, B, A \rangle) = \texttt{C | B | A}$$

- $\mathit{HCType}(csptype)$
  * Description: returns the string that declares the Handel-C type that corresponds to the given $\mathsf{CSP_M}$ type.
  * Example:

$$\mathit{HCType}(T) = \begin{cases} \texttt{integer} & \mathit{if}\ \mathrm{T} = \texttt{Int} \\ \texttt{boolean} & \mathit{if}\ \mathrm{T} = \texttt{Bool} \\ \texttt{T} & \mathit{otherwise} \end{cases}$$

– Variables
- $\mathit{IntBits} : \mathit{int}$: number of bits used to represent the integers
- $\mathit{MutualRec} : \mathit{bool}$: indicates if the specification presents mutual recursion or not
- $\mathit{TailRec} : \mathbb{P}\,\mathit{String}$: Set containing the name of the processes that are tail recursive.

– Constants
- ϵ: the empty string

2 Translation Rules

2.1 Programs

$[\![\mathit{Prog}]\!]^{CSPProg} =$

$\quad \mathit{declareClockAndSync}()$ (Page 4)

$\quad [\![$`datatype boolean = true | false`
`datatype integer = `$(\mathit{IntBranches}(\mathit{IntBits}))$
$\mathit{Prog}\,]\!]^{Types}$ (Page 5)

$\quad [\![\mathit{Prog}\,]\!]^{ConsAndFun}$ (Page 5)

$\quad \mathit{declareChannels}()$ (Page 6)

$\quad \mathit{declareAuxMacros}()$ (Page 4)

$\quad [\![\mathit{Prog}\,]\!]^{Proc}$ (Page 7)

$\quad [\![\mathit{MainProc}\,]\!]^{Main}$ (Page 11)

$\mathit{IntBranches}(n) = \mathit{Separate}(\texttt{|}, (\mathit{IntName})\ \mathit{map}\ (\mathit{OrderedIntRange}(n)))$

2.2 Clock and Sync Values

$\mathit{declareClockAndSync}() =$

```
set clock = external "clock1";
typedef unsigned int 1 SYNC;
const SYNC syncout = 0;
SYNC syncin;
```

2.3 Auxiliary Macros

$\mathit{declareAuxMacros}() =$

```
unsigned 32 random_var =1;
macro proc random(INTEGER_I){
    INTEGER_I =(INTEGER_I<-21)@                         }
               (INTEGER_I[12]^INTEGER_I[30])@
               (INTEGER_I[10:1]^INTEGER_I[29:20]);
macro expr IS_EMPTY_SET(SET_S) = (SET_S == 0);
macro expr SET_UNION(SET_S,SET_T) = (SET_S | SET_T);
macro expr SET_DIFF(SET_S,SET_T) = (SET_S & (~SET_T));
macro expr SET_INTER(SET_S,SET_T) = (SET_S & SET_T);
```

2.4 Types

$$
\begin{array}{l}
[\![\texttt{datatype}\ \ T = T\_decl\ Prog]\!]^{Types} = \\
\quad \textbf{let}\ \ branches = TypeBranchesEnv(T\_decl) \\
\quad \textbf{in}\ \left(\begin{array}{l} \textbf{if}\ \ T \neq \texttt{integer} \\ \textbf{then}\ \ declareType(T, branches) \\ \textbf{else}\ \ \epsilon \end{array}\right) \\
\qquad declareTypeSets(T, branches) \\
\qquad declareLookupTable(T,\ TagsParentEnvs \rhd T) \\
\\
\quad [\![Prog]\!]^{Types} \\
\\
[\![P\ Prog]\!]^{Types} = [\![Prog]\!]^{Types}
\end{array}
$$

$declareType(T, \langle b\_0, \ldots, b\_n \rangle) =$

$\quad$ `#define T unsigned int` $Bits(n)$

$\quad$ `#define b_n 0`

$\quad \ldots$

$\quad$ `#define b_0 n`

$\quad$ `#define T_card` $(n+1)$

$declareTypeSets(T, \langle b\_0, \ldots, b\_n \rangle) =$

$\quad$ `#define T_set unsigned int T_card`

$\quad$ `#define b_n_set 0b`$Bitwise(n+1, n)$

$\quad \ldots$

$\quad$ `#define b_0_set 0b`$Bitwise(n+1, 0)$

$\quad$ `#define T_set_nil 0b`$Bitwise(n+1, -1)$

$\quad$ `static T_set T__set_LUT[T_card] = {b_n_set, ..., b_0_set};`

$$
\begin{array}{l}
declareLookupTables(T, \langle\rangle) = \epsilon \\
declareLookupTables(T, tag\_0 : tags) = \\
\quad \textbf{let}\ \ \langle T\_0, \ldots, T\_m \rangle = TagTypesEnv(tag\_0), \\
\qquad Values = \langle \langle T\_0\_v\_0, \ldots \rangle, \ldots, \langle T\_m\_v\_0, \ldots, \rangle \rangle = TagValuesEnv(tag\_0) \\
\qquad \textbf{let}\ \ \langle \langle T\_0\_v\_0, \ldots, T\_m\_v\_0 \rangle, \ldots \rangle = \Pi\ Values \\
\qquad \textbf{in}\ \texttt{static T T\_tag\_0\_LUT[T\_0\_card]...[T\_m\_card] = \{} \\
\qquad\qquad \{ \ldots \{\ \texttt{T\_0\_v\_0, ..., T\_m\_v\_0}\ \}, \ldots \} \\
\qquad\quad \} \\
\\
\quad declareLookupTables(T, tags)
\end{array}
$$

2.5 Constants and Functions

$$
\begin{array}{l}
[\![Prog]\!]^{ConsAndFun} = \\
\quad declareCFPrototypes \\
\quad [\![Prog]\!]^{CFTrans}
\end{array}
$$

$declareCFPrototypes() =$

$\quad$ **let** $ConstFunArgsEnv = \{c_1 \mapsto args_1, \ldots, c_n \mapsto args_n\}$

$\quad$ **in** `macro expr c_1(`$Separate(,, args_1)$`);`

$\qquad \ldots$

$\qquad$ `macro expr c_n(`$Separate(,, args_n)$`);`

$[\![$ `C = body` $Prog]\!]^{CFTrans} =$

$\quad$ **if** $C \in \mathrm{dom}\, ConstFunArgsEnv$

$\quad$ **then let** $args = ConstFunArgsEnv(C)$

$\qquad\qquad$ **in** `macro expr name(`$Separate(,, args)$`) = (` $[\![body]\!]^{Exp}$ `);`

$\quad$ **else** ϵ

$\quad [\![Prog]\!]^{ConsAndFun}$

$[\![$ `F(args) = body` $Prog]\!]^{CFTrans} =$

$\quad$ **let** $args = ConstFunArgsEnv(C)$

$\quad$ **in** `macro expr name(`$Separate(,, args)$`) = (` $[\![body]\!]^{Exp}$ `);`(Page 11)

$\quad [\![Prog]\!]^{ConsAndFun}$

$[\![P\ Prog]\!]^{ConsAndFun} = [\![Prog]\!]^{ConsAndFun}$

2.6 Channels

$[\![Prog]\!]^{Channels} =$

$\quad$ `chan SYNC INEXISTENT_CHANNEL;` $declareChannels()$

$\quad declareChannelsCons()$

$declareChannels() =$

$\quad$ **let** $\{c_1 \mapsto types_1, \ldots, c_n \mapsto types_n\} = CTypeEnv$

$\quad$ **in** $declareChannel(c_1)$

$\qquad \ldots$

$\qquad declareChannel(c_n)$

$declareChannel(c) =$ `chan` $(declareChannelType(c))\ c\ (declareChannelArray(c))$ `;`

$declareChannelType(c) =$

$\quad$ **let** $types = CTypeEnv(c)$

$\qquad arrayDim = GetArrayDim(c, \#types)$

$\quad$ **in if** $(arrayDim == \#types)$

$\qquad$ **then** `SYNC`

$\qquad$ **else** $HCType(last(types))$

$getArrayDim(c) =$

$\quad$ **let** $size = \#CTypeEnv(c)$

$\quad$ **in if** $c \in CCommEnv$

$\qquad$ **then** $size - 1$

$\qquad$ **else** $size$

$$
\begin{array}{l}
\mathit{declareChannelArray}(c) = \\
\quad \textbf{if}\ \ c \in \mathit{CProjEnv} \\
\quad \textbf{then}\ \ \textbf{let}\ \ \langle T_0, \ldots, T_n \rangle = \mathit{CTypeEnv}(c) \\
\qquad\qquad\quad \mathit{arrayDim} = \mathit{GetArrayDim}(c, \#\mathit{types}) \\
\qquad\quad \textbf{in}\ \ \textbf{if}\ \ (\mathit{arrayDim} == n+1) \\
\qquad\qquad \textbf{then}\ \ \texttt{[}\mathit{HCType}(T_0)\texttt{\_card]} \ldots \texttt{[}\mathit{HCType}(T_{n-1})\texttt{\_card]}\,\texttt{[}\mathit{HCType}(T_n)\texttt{\_card]} \\
\qquad\qquad \textbf{else}\ \ \texttt{[}\mathit{HCType}(T_0)\texttt{\_card]} \ldots \texttt{[}\mathit{HCType}(T_{n-1})\texttt{\_card]} \\
\quad \textbf{else}\ \ \epsilon
\end{array}
$$

$$
\begin{array}{l}
\mathit{declareChannelCons}() = \\
\quad \textbf{let}\ \ \{c_1, \ldots, c_n\} = \mathrm{dom}\ \mathit{CTypeEnv} \\
\quad \textbf{in}\ \ [\![\ \texttt{datatype CHAN = chan\_c\_1 | ... | chan\_c\_n}\ ]\!]^{\mathit{Types}}
\end{array}
$$

2.7 Processes

$$
\begin{array}{l}
[\![\ \mathit{Prog}\ ]\!]^{\mathit{Proc}} = \\
\textbf{if}\ \ \mathit{mutualRec} \\
\textbf{then}\ \ \mathit{declareMutualRecCons}() \\
\qquad [\![\ \mathit{Prog}\ ]\!]^{\mathit{ProcDecl}} \\
\textbf{else}\ \ \mathit{declareProcPrototypes}() \\
\qquad [\![\ \mathit{Prog}\ ]\!]^{\mathit{ProcDeclMR}}
\end{array}
$$

$$
\begin{array}{ll}
\mathit{declareMutualRecCons}() = & \textbf{let}\ \ \{P_0, \ldots, P_n\} = \mathrm{dom}\ \mathit{ProcDefsEnv} \\
& \textbf{in}\ \texttt{\#define P\_n n} \\
& \quad \ldots \\
& \quad \texttt{\#define P\_0 0}
\end{array}
$$

$$
\begin{array}{ll}
\mathit{declareProcPrototypes}() = & \textbf{let}\ \ \{P_0, \ldots, P_n\} = \mathrm{dom}\ \mathit{ProcDefsEnv} \\
& \textbf{in}\ \texttt{inline void P\_0(}\mathit{declareFormalArgs}(P_0)\texttt{);} \\
& \quad \ldots \\
& \quad \texttt{inline void P\_n(}\mathit{declareFormalArgs}(P_n)\texttt{);}
\end{array}
$$

$$
\begin{array}{l}
\mathit{declareFormalArgs}(P) = \\
\quad \textbf{let}\ \ \langle \mathit{arg}_0, \ldots, \mathit{arg}_n \rangle = \mathit{ProcArgsEnv}(P) \\
\quad \textbf{in}\ \ \texttt{(}\,(\mathit{ArgTypeEnv}\ P\ \mathit{arg}_0)\,\texttt{arg\_0,}\ \ \ldots\texttt{,}\ \ (\mathit{ArgTypeEnv}\ P\ \mathit{arg}_n)\,\texttt{arg\_n)}
\end{array}
$$

$[\![P(\mathit{args}) = \mathit{body}\ \mathit{Prog}]\!]^{\mathit{ProcDecl}} =$

if $P \in \mathrm{dom}\,\mathit{ProcDefsEnv}$
then `inline void P(`$\mathit{declareFormalArgs}(P)$`){`
        if $P \in \mathit{TailRec}$
        then `boolean KEEP_LOOPING;`
            `KEEP_LOOPING = true;`
            `while(KEEP_LOOPING){`
                `KEEP_LOOPING = false;`
                $[\![\mathit{PDef}\ ]\!]^{\mathit{PBody}}\ P$
            `}`
        else $[\![\mathit{PDef}\ ]\!]^{\mathit{PBody}}\ P$
    `}`
    $[\![\mathit{Prog}]\!]^{\mathit{ProcDecl}}$
else $[\![\mathit{Prog}]\!]^{\mathit{ProcDecl}}$

$[\![\mathit{Prog}]\!]^{\mathit{ProcDeclMR}} =$

let $\mathit{nproc} = \#\mathit{ProcDefsEnv}$
    $\{P_0, \ldots, P_n\} = \mathrm{dom}\,\mathit{ProcDefsEnv}$
in $\mathit{declareFormalArgsCopies}(P_0)$
    $\ldots$
    $\mathit{declareFormalArgsCopies}(P_n)$

`inline void MUTUAL_REC(int` $\mathit{Bits}(\mathit{nproc})$ `PROGRAM_COUNTER){`
    `boolean KEEP_LOOPING;`
    `KEEP_LOOPING = true;`
    `while(KEEP_LOOPING){`
        `KEEP_LOOPING = false;`
        `switch(PROGRAM_COUNTER){`
            let $\langle P_0, \ldots, P_n \rangle = \mathrm{dom}\,\mathit{ProcDefsEnv}$
            in `case P_0 :{` $[\![\ \mathit{ProcDefsEnv}(P_0)\ ]\!]^{\mathit{PBody}}\ P_0\}$
                $\ldots$
                `case P_n :{` $[\![\ \mathit{ProcDefsEnv}(P_n)\ ]\!]^{\mathit{PBody}}\ P_n\}$
                `default: KEEP_LOOPING = false;`
        `}`
    `}`
`}`

$\mathit{declareFormalArgsCopies}(P) =$

let $\langle \mathit{arg}_0, \ldots, \mathit{arg}_n \rangle = \mathit{ProcArgsEnv}(P)$
in $(\mathit{ArgTypeEnv}\ P\ \mathit{arg}_0)$ `P_local_arg_0;`
    $\ldots$
    $(\mathit{ArgTypeEnv}\ P\ \mathit{arg}_n)$ `P_local_arg_n;`

Process Body

$$[\![\mathit{SKIP}]\!]^{PBody}\ N = \epsilon$$

$$[\![\mathit{STOP}]\!]^{PBody}\ N = \texttt{INEXISTENT\_CHANNEL?syncin;}$$

$$[\![P1 ; P2]\!]^{PBody}\ N = ([\![P1]\!]^{PBody}\ N);([\![P2]\!]^{PBody}\ N)$$

$$[\![\texttt{if}\ \ C\ \texttt{then}\ P1\ \texttt{else}\ \ P2]\!]^{PBody}\ N =$$
$$\quad \texttt{if (} [\![C]\!]^{Exp} \texttt{) \{ } [\![P1]\!]^{PBody}\ N \texttt{ \} else \{ } [\![P2]\!]^{PBody}\ N \texttt{\}}$$

$$[\![P1 \texttt{|\~{}|} P2]\!]^{PBody}\ N =$$
```
random(random_var);
if((random_var%2)==0){
    [[P1]]^PBody N}
else{
    [[P2]]^PBody N}
```

$$[\![C \texttt{\&} P]\!]^{PBody}\ N =$$
$$\quad \texttt{if (} [\![C]\!]^{Exp} \texttt{) \{ } [\![P]\!]^{PBody}\ N \texttt{ \} else \{ } [\![\mathit{STOP}]\!]^{PBody}\ N \texttt{\}}$$

Prefix

$$[\![comm\ \texttt{->}\ P]\!]^{PBody}\ N =$$
$$\quad \texttt{seq\{ } [\![comm]\!]^{DeclInVar} [\![comm]\!]^{Comm} \texttt{ ; } [\![P]\!]^{PBody}\ N \texttt{\}}$$

$$[\![c\, proj ? x]\!]^{DeclInVar} = \mathit{HCtype}(\mathit{last}(\mathit{CTypeEnv}(c)))\ \texttt{x;}$$
$$[\![comm]\!]^{DeclInVar} = \epsilon$$

$$\begin{aligned}[\![c\, proj]\!]^{Comm}\ N = &\ \textbf{if}\ \ N \in \operatorname{dom} \mathit{CInOut} \wedge c \in \operatorname{dom} \mathit{CInOut}(N)\\ &\ \textbf{then}\ \ \textbf{if}\ \ \mathit{CInOut}\ N\ c = \mathit{IN}\\ &\qquad \textbf{then}\ \ \texttt{c}([\![proj]\!]^{Proj}\ (\mathit{CTypeEnv}\ c))\texttt{?syncin}\\ &\qquad \textbf{else}\ \ \texttt{c}([\![proj]\!]^{Proj}\ (\mathit{CTypeEnv}\ c))\texttt{!syncout}\end{aligned}$$

$$[\![c\, proj\, ?x]\!]^{Comm} = \texttt{c}([\![proj]\!]^{Proj}\ (\mathit{CTypeEnv}\ c))\texttt{?x}$$
$$[\![c\, proj\, !e]\!]^{Comm} = \texttt{c}([\![proj]\!]^{Proj}\ (\mathit{CTypeEnv}\ c))\texttt{!}\ [\![e]\!]^{Exp}$$

$$[\![.e_0 \ldots .e_n]\!]^{Proj}\ \langle T_0, \ldots, T_n \rangle =$$
$$\quad \texttt{[}(\textbf{if}\ \ T_0 = \texttt{Int}\ \textbf{then}\ \ \texttt{(unsigned)}\ \textbf{else}\ \ \epsilon)\ [\![e_0]\!]^{Exp}\ \texttt{]}$$
$$\quad \ldots$$
$$\quad \texttt{[}(\textbf{if}\ \ T_n = \texttt{Int}\ \textbf{then}\ \ \texttt{(unsigned)}\ \textbf{else}\ \ \epsilon)\ [\![e_n]\!]^{Exp}\ \texttt{]}$$

External Choice

$$[\![\, comm_0 \texttt{ -> } P_0 \,\Box \ldots \Box\, comm_n \texttt{ -> } P_n \,]\!]^{PBody}\ N =$$
$$[\![\, comm_0 \,]\!]^{DeclInVar}$$
$$\ldots$$
$$[\![\, comm_1 \,]\!]^{DeclInVar}$$

$$\texttt{case}\ \ [\![\ comm_0\ ]\!]^{Comm}\ \texttt{:}\{\ [\![\ P_0\ ]\!]^{PBody}\ N\}\texttt{; break;}$$
$$\ldots$$
$$\texttt{case}\ \ [\![\ comm_n\ ]\!]^{Comm}\ \texttt{:}\{\ [\![\ P_0\ ]\!]^{PBody}\ N\}\texttt{; break;}$$

Parallel Composition

$$[\![\, P_1 \,\texttt{[|}\, cs \,\texttt{|]}\, P_2 \,]\!]^{PBody}\ \texttt{N} =$$
$$\quad TranslateConc(\texttt{N}, P_1, P_2, ((\alpha(P_1) \cap \alpha(P_2)) \setminus cs))$$

$$[\![\, P_1 \,\texttt{[}\, cs_1 \,\texttt{||}\, cs_2 \,\texttt{]}\, P_2 \,]\!]^{PBody}\ \texttt{N} =$$
$$\quad TranslateConc(\texttt{N}, P_1, P_2, ((\alpha(P_1) \cap \alpha(P_2)) \setminus (cs_1 \cap cs_2)))$$

$$[\![\, P_1 \,\texttt{|||}\, P_2 \,]\!]^{PBody}\ \texttt{N} =$$
$$\quad TranslateConc(\texttt{N}, P_1, P_2, (\alpha(P_1) \cap \alpha(P_2)))$$

$$TranslateConc(\texttt{N}, P_1, P_2, \emptyset) =$$
$$\quad \texttt{par}\ \{\ \{\ \ [\![\ P_1\ ]\!]^{PBody}\ \texttt{N}\ \}\ \ \texttt{;}\ \{\ \ [\![\ P_1\ ]\!]^{PBody}\ \texttt{N}\}\ \}$$

Process/Function Call

$$[\![\, P(e_0, \ldots, e_n) \,]\!]^{PBody}\ \texttt{N} =$$
$$\quad \textbf{if}\ \ P \in \mathrm{dom}\, ProcArgsEnv$$
$$\quad \textbf{then}\ \ \textbf{let}\ \ \langle arg_0, \ldots, arg_n \rangle = ProcArgsEnv(P)$$
$$\qquad \textbf{in}\ \ \textbf{if}\ \ (MutualRec \vee P = \texttt{N})$$
$$\qquad\quad \textbf{then}\ \texttt{P\_local\_arg\_0 = }\ [\![\ e_0\ ]\!]^{Exp}\ \texttt{;}$$
$$\qquad\qquad \ldots$$
$$\qquad\qquad \texttt{P\_local\_arg\_n = }\ [\![\ e_n\ ]\!]^{Exp}\ \texttt{;}$$
$$\qquad\qquad \left(\begin{array}{l} \textbf{if}\ \ (MutualRec) \\ \textbf{then}\ \ \textbf{if}\ \ \texttt{N} = \texttt{mainp} \\ \qquad \textbf{then}\ \ \texttt{MUTUAL\_REC(P);} \\ \qquad \textbf{else}\ \ \texttt{PROGRAM\_COUNTER = P;} \\ \textbf{else}\ \ \epsilon \end{array}\right)$$
$$\qquad\qquad \texttt{KEEP\_LOOPING = true;}$$
$$\qquad\quad \textbf{else}\ \ \texttt{P(}\ [\![\ e_0]\!]^{Exp}, \ldots, [\![e_n\ ]\!]^{Exp}\ \texttt{);}$$
$$\quad \textbf{else}\ \ \texttt{P(}\ [\![\ e_0]\!]^{Exp}, \ldots, [\![e_n\ ]\!]^{Exp}\ \texttt{);}$$

2.8 Main Process

$$[\![main]\!]^{Main} =$$

```
void main(){
    [[ main ]]^PBody mainp
}
```

2.9 Expressions

The function $[\![exp]\!]^{Exp}$ returns the corresponding Handel-C code for a given CSP_M expression.

3 Extensions

3.1 Allowing interleaved events

- Declaration of the branches
- Changes to translation (`this.useSemaphors`)