The manual

Don't know yet for the authors

August 7, 2015

1 First steps of Copilot

Copilot is a wonderful language, the best EDSL in Haskell for writing real time monitors in C that also produce ACSL contracts and compile in CompCert, and you definitely may use it (because there is no other one that can do such things) ! We give here a grammar of the language, and you will follow its rules to write your code, or you'll get bad surprises. This is for instance the case for let local bindings. We ask you to do the β-expansion yourself, because we think you are clever and you definitely may be able to do it. If you don't do it, the program will go on strike and refuse to check the assertions on ACSL (unless you pay a lot of money to buy the latest plugin of frama-c that supports let bindings).

2 Be careful !

The convention on muxes are the following : mux cond iftrue iffalse.

3 About git

On our beautiful github we have a lot of different projects : here is a litte overview of them. https://github.com/Copilot
Language

1. Copilot : https://github.com/Copilot-Language/Copilot This is the basic directory. Many branches but only two really active. Master and Struct. This repository contains all unit tests, basic documentation, and so on. Those two branches should be merged into master soon.

2. atom_for_copilot : https://github.com/Copilot-Language/atom_for_copilot The modified package ATOM for generating ACSL contracts with Copilot. Some headers modified so NOT BACKWARD COMPATIBLE. One branch, probably deprecated.

3. sbv-for-copilot : https://github.com/Copilot-Language/sbv-for-copilot This used to be a modified package for SBV. Deprecated, DO NOT USE. All the changes have been merged into the official SBV version 5.0.

4. examplesForACSL : https://github.com/Copilot-Language/examplesForACSL Some examples and projects in it to test. Makefiles provided. One branch : master. When using it, do not forget at each example to initialize the sandbox by cabal sandbox init –sandbox ../../Copilot/.cabal-sandbox/

   Becareful ! Some examples are not fitted to be used with frama-c, or other tools (splint). Always check before running it, or you comnputer can crash by swapping.

5. copilot-cbmc : https://github.com/Copilot-Language/copilot-cbmc insert description here TODO. One branch : master

6. copilot-c99 : https://github.com/Copilot-Language/copilot-c99 DEPRECATED : atom backend for SBV. Two branches. An original one, and an other using the atom_for_copilot modified package.

7. copilot-discussion : https://github.com/Copilot-Language/copilot-discussion Some blabla TODO

8. copilot-librairies : https://github.com/Copilot-Language/copilot-libraries If you do not want to reinvent the wheel.

4 Devlopper mode : compile it from github

You are a developper, and you want to compile it from source in a sandbox ? Here is the right section !

What you need to do :

1. First install all the useful stuff : CompCert (2.4 or later, coq needed, opam recommanded), Frama-c (version sodium or later), Z3 (at least 4.3.2), CVC4 (at least 1.4), ghc 7.10 or later (and its whole toolchain, like the haskell platform).

   Optionnaly, you could install splint, an up to date gcc (5.1 or later).

   First install the haskell platform

   Then install the right version of ghc (¿= 7.10)

2. Make sure that cabal is installed, and update it to the version 1.22 or later (run `cabal install cabal cabal-install` for that).

3. create a directory named Copilot-Language and cd into it. Then clone the following :

   https://github.com/Copilot-Language/examplesForACSL

   https://github.com/LeventErkok/sbv

   https://github.com/Copilot-Language/Copilot

4. cd into the directory named Copilot and then do the following

   `git submodule update --init`

   go into the submodules and change the branches you want (for example lib/copilot-sbv has a branch named acsl) with the command `git checkout BRANCHNAME`.

   In the Copilot folder (the one cded in 3) `make test`

5. Normally, it should have failed ! Because it has installed all dependecies on their official release in a cabal sandbox located in this folder Copilot (if not, you should probably do a `cabal install --only-dependencies`). But you need at least a sbv 5.0 (which is not released yet). You have now to do the following :

   Go into Copilot-Language/sbv and do `cabal sandbox init --sandbox ../Copilot/.cabal-sandbox/` and then `cabal install`.

6. Now go in Copilot and redo make test : it should compile everything, but the Copilot-regression should fail (which is normal because it is deprecated).

7. It's done ! You now have the latest version of Copilot in your sandbox ! To run some test, go into the Copilot-Language/examplesForACSL

4.1 And after getting all the stuff from git

1. Install opam
2. Install coq, menhir with opam
3. Install dot (graphviz package), GNU parallel, m4 (apt-get)
4. Install CompCert from source
5. Install gtksourceview, gnomecanvas (libgtksoureview-2.0-dev and libgnomecanvas2-dev on debian)
6. Install with opam frama-c, why3
7. Install CVC4 (if from source, you will need antlr3 and boost)
8. Do `why3 config --detect` . It's done !
9. Choose one example randomly (choose the WCV if you want to try our most famous example) from examplesForACSL. Cd into it
10. do `cabal sandbox init --sandbox ../../Copilot/.cabal-sandbox/`
11. Do `make compile` to compile it with ccomp, and `make acsl` to prove everything with frama-c
12. If you want to control the whole process, you can look at the makefile, especially `make sandbox` to build it from the sandbox.
13. It should have created you a folder named copilot-sbv-codegen. Cd into it :

    do `make fwp` to use the wp plugin of frama-c to check, `make all` to compile it with compcert (into a internal.a library), `make splint` to check it with splint, `make fval` to check it with value analysis plugin (it will unroll the infinite loop, so the analysis may never terminate).
14. For the examples 29 and bigger, there is also a m4 preprocessing done before compilation. Do not hesitate to use it if you need it !

4.2 update it

You had a party on week end, and when you come it does not work anymore ? You need to update it ? Follow these instructions :

1. First you have to pull all the following (according to the branch you want to execute) :

    sbv

    examplesForACSL

    Copilot

    Copilot/lib/*
2. Then, go in Copilot, and `make veryclean`, then `make test`. When it fails, go to sbv and `cabal sandbox init --sandbox ../Copilot/.cabal-sandbox/` and `cabal install`. Then go again in Copilot and `make test`. Then you can go in the examples and make them again (if they do not compile, do a `cabal sandbox init --sandbox ../../Copilot/.cabal-sandbox/`)

5 Prover mode : prove it with ACSL

If you want to prove it with ACSL you have to follow some basic rules :

1. Forget about ATOM. It works only with sbv
2. Never use let bindings
3. Use labels : if the label starts with a ? character, then it will split what is labeled in an other file, which may be easier to prove (see example29 for that kind of usage). If you don't, the prover may only do timeouts, or even say unknown to everything.
4. Caution ! You need to install gnu parallel in case of many files !
5. If you want, you can change the prover in the file copilot-sbv-codegen/copilot.mk (default CVC4), and add other options, do it as you want. But don't forget, as soon as you will do a make sandbox in the example29 folder, it will delete all your modifications !
6. forget about bitwise operators
7. if-then-else does not work in the sandbox. Use a mux instead (actualy ite is a syntaxical sugar for mux) ! Same for all imports in the form Language.Copilot, yoiu have to import yourself Copilot.Language.THEMODULEYOUWANT

6 Board mode

Once you have compiled everything into the right architecture, you may want to try it. For it you need to have a binary file into the ARM architecture for BeagleBone/PX4,... (Arduino usage : TODO). Then upload it and run it. You can if you want program triggers that would send special signal on pins, which will allow to control via a transistor a LED or something like that.

For this you would need a cross compiler of CompCert. Or compile CompCert directly on the board (or an other that has a similar instruction set). For ARM-v7A, raspberry pi would do the affair (just as BeagleBones). For ARMv7E-M (PX4) see there : http://compcert.inria.fr/man/manual002.html

TODO : put the experimental results there

7 Grammar and Typing

TODO : move this section up, before all the implementation details.

Here you can paste a big part of the former manual. And change some details also. If needed, just copy paste the grammar and typing rules there and split them in the adequate sections.

$$
\begin{array}{lcl}
\langle defs \rangle & ::= & (\langle def \rangle)^* \\
\\
\langle ctype \rangle & ::= & \texttt{Bool} \\
 & | & \texttt{Int8} \\
 & | & \texttt{Int16} \\
 & | & \texttt{Int32} \\
 & | & \texttt{Int64} \\
 & | & \texttt{Word8} \\
 & | & \texttt{Word16} \\
 & | & \texttt{Word32} \\
 & | & \texttt{Word64} \\
 & | & \texttt{Float} \\
 & | & \texttt{Double} \\
\\
\langle def \rangle & ::= & (\langle id \rangle \texttt{::Stream } \langle ctype \rangle)?\langle id \rangle \texttt{=} \langle spec \rangle \\
\\
\langle id \rangle & ::= & (\mathtt{a}-\mathtt{z})(\mathtt{a}-\mathtt{z}|\mathtt{A}-\mathtt{Z}|\mathtt{0}-\mathtt{9}|\_|\texttt{-}|\texttt{'})^* \\
\\
\langle string \rangle & ::= & \texttt{"}(\mathtt{a}-\mathtt{z}|\mathtt{A}-\mathtt{Z})(\mathtt{a}-\mathtt{z}|\mathtt{A}-\mathtt{Z}|\_|\mathtt{0}-\mathtt{9})^*_{\leq 30}\texttt{"} \\
\\
\langle stream \rangle & ::= & \langle valueList \rangle \texttt{++} \langle stream \rangle \\
 & | & \langle funStream \rangle \\
\\
\langle valueList \rangle & ::= & [(\langle vbool \rangle)^*_{,}] \\
 & | & [(\langle vint \rangle)^*_{,}] \\
 & | & [(\langle vfloat \rangle)^*_{,}] \\
\\
\langle vbool \rangle & ::= & \texttt{true} \\
 & | & \texttt{false} \\
\\
\langle vint \rangle & ::= & [\texttt{+}|\texttt{-}](\mathtt{0}-\mathtt{9})^+ \\
\\
\langle vfloat \rangle & ::= & \langle vint \rangle.(\mathtt{0}-\mathtt{9})^+ \\
\\
\langle funStream \rangle & ::= & \langle externV \rangle\ \langle string \rangle\ \langle contextV \rangle \\
 & | & \texttt{label}\ \langle string \rangle\ \langle funStream \rangle \\
 & | & \texttt{externFun}\ \langle string \rangle\ \langle argList \rangle\ \langle contextF \rangle \\
 & | & \langle externA \rangle\ \langle string \rangle\ \langle stream \rangle\ \langle int \rangle\ \langle contextA \rangle \\
 & | & \langle op1 \rangle\ \langle funStream \rangle \\
 & | & \langle funStream \rangle\ \langle op2Infix \rangle\ \langle funStream \rangle \\
 & | & \langle op3 \rangle\ \langle funStream \rangle\ \langle funStream \rangle\ \langle funStream \rangle \\
 & | & \langle dropStream \rangle
\end{array}
$$

⟨*externV*⟩	::=	`extern`
	|	`externB`
	|	`externI8`
	|	`externI16`
	|	`externI32`
	|	`externI64`
	|	`externW8`
	|	`externW16`
	|	`externW32`
	|	`externW64`
	|	`externF`
	|	`externD`
⟨*externA*⟩	::=	`externArray`
	|	`externArrayB`
	|	`externArrayI8`
	|	`externArrayI16`
	|	`externArrayI32`
	|	`externArrayI64`
	|	`externArrayW8`
	|	`externArrayW16`
	|	`externArrayW32`
	|	`externArrayW64`
	|	`externArrayF`
	|	`externArrayD`
⟨*contextV*⟩	::=	`Nothing`
	|	`Just` ⟨*stream*⟩
⟨*contextF*⟩	::=	`Nothing`
	|	`Just` ⟨*stream*⟩
⟨*contextA*⟩	::=	`Nothing`
	|	`Just` $[(\langle valueList \rangle)^{*}_{,}]$
⟨*argList*⟩	::=	[(`Arg` ⟨*stream*⟩)$^{*}_{,}$]
	|	`[]`
	|	(`Arg` ⟨*stream*⟩)`:`⟨*argList*⟩

```
⟨dropStream⟩  ::=  ⟨id⟩
               |   constant ⟨value⟩
               |   drop ⟨int⟩ ⟨stream⟩

⟨op1⟩         ::=  not | abs | signum | complement
               |   recip | exp | sqrt
               |   log | sin | cos | tan
               |   asin | acos | atan
               |   sinh | cosh | tanh
               |   asinh | acosh | atanh
               |   cast | unsafeCast

⟨op2Infix⟩    ::=  + | - | * | `mod` | `div`
               |   / | ** | `logBase`
               |   < | <= | == | /= | >= | >
               |   || | && | `xor` | ==>
               |   .&. | .|. | .^. | .>>. | .<<.

⟨op3⟩         ::=  mux
```

$$\frac{s \text{ is a } \langle \textit{string} \rangle \text{ token}}{\Gamma \vdash s : \texttt{String}} \quad (\textsc{stringConst})$$

$$\frac{\tau \in \mathrm{inst}(\Gamma(s))}{\Gamma \vdash s : \tau} \quad (\textsc{inst})$$

$$\frac{\Gamma \vdash s : \texttt{Stream}\ \tau}{\Gamma \vdash \texttt{Arg}\ s : \texttt{Arg}} \quad (\textsc{arg})$$

$$\frac{\Gamma \vdash x : \tau}{\Gamma \vdash \texttt{constant}\ x : \texttt{Stream}\ \tau} \quad (\textsc{valConst})$$

$$\frac{\Gamma \vdash i : \texttt{Integer} \qquad \Gamma \vdash x : \texttt{Stream}\ \tau}{\Gamma \vdash \texttt{drop}\ i\ x : \texttt{Stream}\ \tau} \quad (\textsc{drop})$$

$$\frac{\Gamma \vdash ls : [a] \qquad \Gamma \vdash s : \text{Spec } a}{\Gamma \vdash ls + +s : \text{Spec } a} \quad (\textsc{Append})$$

$$\frac{\Gamma \vdash s : \texttt{String} \qquad \Gamma \vdash x : \texttt{Stream}\ \tau}{\Gamma \vdash \texttt{label}\ s\ x : \texttt{Stream}\ \tau} \quad (\textsc{label})$$

$$\frac{\Gamma \vdash s : \texttt{String} \qquad \Gamma \vdash x : \texttt{Stream}\ \tau}{\Gamma \vdash \texttt{extern}\ s\ x : \texttt{Stream}\ \tau} \quad (\textsc{ext})$$

$$\frac{\Gamma \vdash s : \texttt{String} \qquad \Gamma \vdash x : \texttt{Stream Bool}}{\Gamma \vdash \texttt{externB}\ s\ x : \texttt{Stream Bool}} \quad (\textsc{ext}\text{B})$$

$$\frac{\Gamma \vdash s : \texttt{String} \qquad \Gamma \vdash x : \texttt{Stream Int8}}{\Gamma \vdash \texttt{externI8}\ s\ x : \texttt{Stream Int8}} \quad (\textsc{ext}\text{I8})$$

$$\frac{\Gamma \vdash s : \texttt{String} \qquad \Gamma \vdash x : \texttt{Stream Int16}}{\Gamma \vdash \texttt{externI16}\ s\ x : \texttt{Stream Int16}} \quad (\textsc{ext}\text{I16})$$

$$\frac{\Gamma \vdash s : \texttt{String} \qquad \Gamma \vdash x : \texttt{Stream Int32}}{\Gamma \vdash \texttt{externI32}\ s\ x : \texttt{Stream Int32}} \quad (\textsc{ext}\text{I32})$$

$$\frac{\Gamma \vdash s : \texttt{String} \qquad \Gamma \vdash x : \texttt{Stream Int64}}{\Gamma \vdash \texttt{externI64}\ s\ x : \texttt{Stream Int64}} \quad (\textsc{ext}\text{I64})$$

$$\frac{\Gamma \vdash s : \texttt{String} \qquad \Gamma \vdash x : \texttt{Stream Word8}}{\Gamma \vdash \texttt{externW8}\ \ s\ x : \texttt{Stream Word8}} \quad (\textsc{ext}\text{W8})$$

$$\frac{\Gamma \vdash s : \texttt{String} \qquad \Gamma \vdash x : \texttt{Stream Word16}}{\Gamma \vdash \texttt{externW16}\ \ s\ x : \texttt{Stream Word16}} \quad (\textsc{ext}\text{W16})$$

$$\frac{\Gamma \vdash s : \texttt{String} \qquad \Gamma \vdash x : \texttt{Stream Word32}}{\Gamma \vdash \texttt{externW32}\ \ s\ x : \texttt{Stream Word32}} \quad (\textsc{ext}\text{W32})$$

$$\frac{\Gamma \vdash s : \texttt{String} \qquad \Gamma \vdash x : \texttt{Stream Word64}}{\Gamma \vdash \texttt{externW64}\ \ s\ x : \texttt{Stream Word64}} \quad (\textsc{ext}\text{W64})$$

$$\frac{\Gamma \vdash s : \texttt{String} \qquad \Gamma \vdash x : \texttt{Stream Float}}{\Gamma \vdash \texttt{externF}\ \ s\ x : \texttt{Stream Float}} \quad (\textsc{ext}\textsc{Float})$$

$$\frac{\Gamma \vdash s : \texttt{String} \qquad \Gamma \vdash x : \texttt{Stream Double}}{\Gamma \vdash \texttt{externD}\ \ s\ x : \texttt{Stream Double}} \quad (\textsc{ext}\textsc{Double})$$

$$\frac{\Gamma \vdash s : \texttt{String} \qquad \Gamma \vdash a : \texttt{[Arg]} \qquad \Gamma \vdash x : \texttt{Stream}\ \tau}{\Gamma \vdash \texttt{externFun}\ \ s\ a\ x : \texttt{Stream}\ \tau} \quad (\textsc{ext}\textsc{Fun})$$

$$\frac{\Gamma \vdash s : \texttt{String} \qquad \Gamma \vdash i : \texttt{Integral}\ \tau_1 \Rightarrow \texttt{Stream}\ \tau_1 \qquad \Gamma \vdash m : \texttt{Integer} \qquad \Gamma \vdash x : \texttt{[[}\tau\texttt{]]}}{\Gamma \vdash \texttt{externArray}\ \ s\ i\ m\ x : \texttt{Stream}\ \tau} \quad (\textsc{ext}\text{A})$$

$$\frac{\Gamma \vdash s : \texttt{String} \qquad \Gamma \vdash i : \texttt{Integral}\ \tau_1 \Rightarrow \texttt{Stream}\ \tau_1 \qquad \Gamma \vdash m : \texttt{Integer} \qquad \Gamma \vdash x : \texttt{[[Bool]]}}{\Gamma \vdash \texttt{externArrayB}\ \ s\ i\ m\ x : \texttt{Stream Bool}} \quad (\textsc{ext}\text{AB})$$

$$\frac{\Gamma \vdash s : \texttt{String} \qquad \Gamma \vdash i : \texttt{Integral}\ \tau_1 \Rightarrow \texttt{Stream}\ \tau_1 \qquad \Gamma \vdash m : \texttt{Integer} \qquad \Gamma \vdash x : \texttt{[[Int8]]}}{\Gamma \vdash \texttt{externArrayI8}\ \ s\ i\ m\ x : \texttt{Stream Int8}} \quad (\textsc{ext}\text{AI8})$$

$$\frac{\Gamma \vdash s : \texttt{String} \qquad \Gamma \vdash i : \texttt{Integral}\ \tau_1 \Rightarrow \texttt{Stream}\ \tau_1 \qquad \Gamma \vdash m : \texttt{Integer} \qquad \Gamma \vdash x : \texttt{[[Int16]]}}{\Gamma \vdash \texttt{externArrayI16}\ \ s\ i\ m\ x : \texttt{Stream Int16}} \quad (\textsc{ext}\text{AI16})$$

$$\frac{\Gamma \vdash s : \texttt{String} \qquad \Gamma \vdash i : \texttt{Integral}\ \tau_1 \Rightarrow \texttt{Stream}\ \tau_1 \qquad \Gamma \vdash m : \texttt{Integer} \qquad \Gamma \vdash x : \texttt{[[Int32]]}}{\Gamma \vdash \texttt{externArrayI32}\ \ s\ i\ m\ x : \texttt{Stream Int32}} \quad (\textsc{ext}\text{AI32})$$

$$\frac{\Gamma \vdash s : \texttt{String} \qquad \Gamma \vdash i : \texttt{Integral}\ \tau_1 \Rightarrow \texttt{Stream}\ \tau_1 \qquad \Gamma \vdash m : \texttt{Integer} \qquad \Gamma \vdash x : \texttt{[[Int64]]}}{\Gamma \vdash \texttt{externArrayI64}\ s\ i\ m\ x : \texttt{Stream Int64}} \quad (\textsc{extAI64})$$

$$\frac{\Gamma \vdash s : \texttt{String} \qquad \Gamma \vdash i : \texttt{Integral}\ \tau_1 \Rightarrow \texttt{Stream}\ \tau_1 \qquad \Gamma \vdash m : \texttt{Integer} \qquad \Gamma \vdash x : \texttt{[[Word8]]}}{\Gamma \vdash \texttt{externArrayW8}\ s\ i\ m\ x : \texttt{Stream Word8}} \quad (\textsc{extAW8})$$

$$\frac{\Gamma \vdash s : \texttt{String} \qquad \Gamma \vdash i : \texttt{Integral}\ \tau_1 \Rightarrow \texttt{Stream}\ \tau_1 \qquad \Gamma \vdash m : \texttt{Integer} \qquad \Gamma \vdash x : \texttt{[[Word16]]}}{\Gamma \vdash \texttt{externArrayW16}\ s\ i\ m\ x : \texttt{Stream Word16}} \quad (\textsc{extAW16})$$

$$\frac{\Gamma \vdash s : \texttt{String} \qquad \Gamma \vdash i : \texttt{Integral}\ \tau_1 \Rightarrow \texttt{Stream}\ \tau_1 \qquad \Gamma \vdash m : \texttt{Integer} \qquad \Gamma \vdash x : \texttt{[[Word32]]}}{\Gamma \vdash \texttt{externArrayW32}\ s\ i\ m\ x : \texttt{Stream Word32}} \quad (\textsc{extAW32})$$

$$\frac{\Gamma \vdash s : \texttt{String} \qquad \Gamma \vdash i : \texttt{Integral}\ \tau_1 \Rightarrow \texttt{Stream}\ \tau_1 \qquad \Gamma \vdash m : \texttt{Integer} \qquad \Gamma \vdash x : \texttt{[[Word64]]}}{\Gamma \vdash \texttt{externArrayW64}\ s\ i\ m\ x : \texttt{Stream Word64}} \quad (\textsc{extAW64})$$

$$\frac{\Gamma \vdash s : \texttt{String} \qquad \Gamma \vdash i : \texttt{Integral}\ \tau_1 \Rightarrow \texttt{Stream}\ \tau_1 \qquad \Gamma \vdash m : \texttt{Integer} \qquad \Gamma \vdash x : \texttt{[[Float]]}}{\Gamma \vdash \texttt{externArrayF}\ s\ i\ m\ x : \texttt{Stream Float}} \quad (\textsc{extAFloat})$$

$$\frac{\Gamma \vdash s : \texttt{String} \qquad \Gamma \vdash i : \texttt{Integral}\ \tau_1 \Rightarrow \texttt{Stream}\ \tau_1 \qquad \Gamma \vdash m : \texttt{Integer} \qquad \Gamma \vdash x : \texttt{[[Double]]}}{\Gamma \vdash \texttt{externArrayD}\ s\ i\ m\ x : \texttt{Stream Double}} \quad (\textsc{extADouble})$$

$$\frac{\Gamma \vdash x : \texttt{Stream Bool} \qquad op \in \{\texttt{not}\}}{\Gamma \vdash op\ x : \texttt{Stream Bool}} \quad (\textsc{op1Bool})$$

$$\frac{\Gamma \vdash x : \texttt{Bits}\ \tau \Rightarrow \texttt{Stream}\ \tau \qquad op \in \{\texttt{complement}\}}{\Gamma \vdash op\ x : \texttt{Stream}\ \tau} \quad (\textsc{op1Bitwise})$$

$$\frac{\Gamma \vdash x : \texttt{Integral}\ \tau \Rightarrow \texttt{Stream}\ \tau \qquad op \in \{\texttt{abs}, \texttt{signum}\}}{\Gamma \vdash op\ x : \texttt{Stream}\ \tau} \quad (\textsc{op1Num})$$

$$\frac{\Gamma \vdash x : \texttt{Stream Bool} \qquad \Gamma \vdash y : \texttt{Stream Bool} \qquad op \in \{\texttt{||}, \texttt{\&\&}, \texttt{`xor`}, \texttt{==>}\}}{\Gamma \vdash op\ x\ y : \texttt{Stream Bool}} \quad (\textsc{op2Bool})$$

$$\frac{\Gamma \vdash x : \texttt{Integral}\ \tau \Rightarrow \texttt{Stream}\ \tau \qquad \Gamma \vdash y : \texttt{Integral}\ \tau \Rightarrow \texttt{Stream}\ \tau \qquad op \in \{\texttt{`mod`}, \texttt{`div`}\}}{\Gamma \vdash op\ x\ y : \texttt{Stream}\ \tau}$$

(op2Integral)

$$\frac{\Gamma \vdash x : \texttt{Fractionnal}\ \tau \Rightarrow \texttt{Stream}\ \tau \qquad \Gamma \vdash y : \texttt{Fractionnal}\ \tau \Rightarrow \texttt{Stream}\ \tau \qquad op \in \{\texttt{/}\}}{\Gamma \vdash op\ x\ y : \texttt{Stream}\ \tau}$$

(op2Fractionnal)

$$\frac{\Gamma \vdash x : \texttt{Floating}\ \tau \Rightarrow \texttt{Stream}\ \tau \qquad \Gamma \vdash y : \texttt{Floating}\ \tau \Rightarrow \texttt{Stream}\ \tau \qquad op \in \{\texttt{**}, \texttt{`logBase`}\}}{\Gamma \vdash op\ x\ y : \texttt{Stream}\ \tau}$$

(op2Floating)

$$\frac{\Gamma \vdash x : \texttt{Num}\ \tau \Rightarrow \texttt{Stream}\ \tau \qquad \Gamma \vdash y : \texttt{Num}\ \tau \Rightarrow \texttt{Stream}\ \tau \qquad op \in \{\texttt{+}, \texttt{-}, \texttt{*}\}}{\Gamma \vdash op\ x\ y : \texttt{Stream}\ \tau}$$

(op2Num)

$$\frac{\Gamma \vdash x : \texttt{Eq}\ \tau \Rightarrow \texttt{Stream}\ \tau \qquad \Gamma \vdash y : \texttt{Eq}\ \tau \Rightarrow \texttt{Stream}\ \tau \qquad op \in \{\texttt{==}, \texttt{/=}\}}{\Gamma \vdash op\ x\ y : \texttt{Stream}\ \tau}$$

(op2Eq)

$$\frac{\Gamma \vdash x : \texttt{Ord}\ \tau \Rightarrow \texttt{Stream}\ \tau \qquad \Gamma \vdash y : \texttt{Ord}\ \tau \Rightarrow \texttt{Stream}\ \tau \qquad op \in \{\texttt{<}, \texttt{<=}, \texttt{>=}, \texttt{>}\}}{\Gamma \vdash op\ x\ y : \texttt{Stream}\ \tau}$$

(op2Ord)

$$\frac{\Gamma \vdash x : \texttt{Bits}\ \tau \Rightarrow \texttt{Stream}\ \tau \qquad \Gamma \vdash y : \texttt{Bits}\ \tau \Rightarrow \texttt{Stream}\ \tau \qquad op \in \{\texttt{.\&.}, \texttt{.|.}, \texttt{.\^{}.}\}}{\Gamma \vdash op\ x\ y : \texttt{Stream}\ \tau}$$

(op2Bitwise)

$$\frac{\Gamma \vdash x : \texttt{Bits}\ \tau \Rightarrow \texttt{Stream}\ \tau \qquad \Gamma \vdash y : \texttt{Integral}\ \tau_1 \Rightarrow \texttt{Stream}\ \tau_1 \qquad op \in \{\texttt{.>>.}, \texttt{.<<.}\}}{\Gamma \vdash op\ x\ y : \texttt{Stream}\ \tau}$$

(op2BwShift)

$$\frac{\Gamma \vdash x : \texttt{Stream Bool} \qquad \Gamma \vdash y : \texttt{Stream}\ \tau \qquad \Gamma \vdash z : \texttt{Stream}\ \tau \qquad op \in \{\texttt{mux}\}}{\Gamma \vdash op\ x\ y\ z : \texttt{Stream}\ \tau}$$

(op3)