`pl-syntax` Package[*]

Robert Harper

February 24, 2024

Overview[1]

Use the declaration `\usepackage{pfpl-syntax}` to define syntax macros for programming languages inspired by the author's textbook *Practical Foundations for Programming Languages*. The package provides an integrated, systematic treatment of abstract and concrete syntax for a wide range of languages.

Abstract Binding Trees

To typeset an abstract binding tree yourself, provide the package with the `[abt]` option, and use the following commands:

- `\Opn{op}`: Format an operator as a keyword; use starred form for an operator that is to be set in math mode. For example, `\Opn{fun}` produces `fun` and `\Opn*{\lambda}` produces λ.

- `\Abt{operator}<parameters>[minor](major)`: produce an abt, with all arguments typeset. In typical usage the parameters are symbols that index certain forms of abt's; the minor arguments are types that would appear in full abstract syntax; and the major arguments are abt's or abstractors constituting the components of a compound abt. The starred form provides a "default" concrete syntax that omits the minor arguments. If the parenthesized arguments are empty, then no parentheses are typeset. For example, the command

  ```
  \Abt{\Opn{get}}<a>[\tau]
  ```

  produces $\texttt{get}\langle a\rangle[\tau]$, and the command

  ```
  \Abt{\Opn{set}}[\tau]<a>(e)
  ```


[1]I am grateful to Kartik Singal for help with testing and improving this package.

produces $\texttt{set}\langle a\rangle[\tau](e)$. The starred forms produce $\texttt{get}\langle a\rangle$, and $\texttt{set}\langle a\rangle(e)$, respectively.

- `\Abs<symbols>(variables){abt}`: produce an abstractor over symbols and/or variables within an ABT. For example, the command

  ```
  \Abt{\Opn*{\lambda}}[\tau](\Abs(x){e})
  ```

  produces $\lambda[\tau](x\,.\,e)$ and the starred form produces $\lambda(x\,.\,e)$. Similarly, the command

  ```
  \Abt{\Opn{dcl}}[\tau;\rho](e;\Abs<a>{m})
  ```

  produces $\texttt{dcl}[\tau;\rho](e;a\,.\,m)$ and the starred form produces $\texttt{dcl}(e;a\,.\,m)$.

The package option `[sf]` indicates that operations are to be set in sans serif font, and the package option `[sc]` indicates that they are to be set in small capitals. The package option `[oldstyle]`, aka `[book]`, uses the notational conventions from the second edition in which symbol parameters to operators are typeset in square brackets, and their minor arguments are typeset in curly braces. *Warning: this option only works in typical cases in which there are at most one each of parameter, minor, and major arguments! Otherwise you must define your own version of the macro to get the book-style formatting.*

The command `\Sub{e}{x}{e'}` expands to $[e/x]e'$, and `\Sub*{e}{x}{e'}` expands to $e'[e/x]$, providing pre- and post-fix notation for substitution.

Language-Specific Definitions

The commands for typesetting the many language constructs in PFPL, plus some others, are organized into a collection of tables grouped roughly according to type structure. The tables display the abstract and concrete syntax for a language construct, and give the LaTeX command used to create them. The minor arguments are generally required for the abstract syntax, but may be omitted for the concrete syntax.[2] For the variadic forms in Figures 1, the notation τ_I, for I an index set, stands for the finite map $(i \hookrightarrow \tau_i \mid i \in I)$; the notation e_I is defined analogously.

[2] In some cases defaults for the minor arguments are provided, but they are hereby denigrated and should not be relied upon.

Abstract	*Concrete*	*Invocation*
$\mathtt{unit}$	1	\unitTy
$\mathtt{null}$	$\langle\rangle$	\unitEx
$\mathtt{check}[\rho](e_1\,;\,e_2)$	$\mathtt{check}\ e_1\,\{\,e_2\,\}$	\checkEx[\rho]{e_1}{e_2}
$\mathtt{prod}(\tau_1\,;\,\tau_2)$	$\tau_1 \times \tau_2$	\prodTy{\tau_1}{\tau_2}
$\mathtt{proj}\langle i\rangle[\tau_1\,;\,\tau_2](e)$	$e \cdot i$	\projEx<i>[\tau_1][\tau_2]{e} $(i = 1, 2)$
$\mathtt{pair}[\tau_1\,;\,\tau_2](e_1\,;\,e_2)$	$\langle e_1, e_2\rangle$	\pairEx[\tau_1][\tau_2]{e_1}{e_2}
$\mathtt{vprod}\langle I\rangle(\tau_I)$	$\times_{i\in I}(i \hookrightarrow \tau_i)$	\vprodTy<I><i>{\tau}
$\mathtt{vtuple}\langle I\rangle[\tau_I](e_I)$	$\langle i \hookrightarrow e_i \mid i \in I\rangle$	\vtupleEx<I><i>[\tau]{e}
$\mathtt{vproj}\langle I\,;\,i\rangle[\tau_I](e)$	$e \cdot i$	\vprojEx<I><i>[\tau]{e} $(i \in I)$
$\mathtt{void}$	0	\voidTy
$\mathtt{absurd}[\rho](e)$	$\mathtt{absurd}(e)$	\absurdEx[\rho]{e}
$\mathtt{sum}(\tau_1\,;\,\tau_2)$	$\tau_1 + \tau_2$	\sumTy{\tau_1}{\tau_2}
$\mathtt{inj}\langle i\rangle[\tau_1\,;\,\tau_2](e)$	$i \cdot e$	\injEx<i>[\tau_1][\tau_2]{e} $(i = 1, 2)$
$\mathtt{case}[\tau_1\,;\,\tau_2\,;\,\rho](e\,;\,x_1\,.\,e_1\,;\,x_2\,.\,e_2)$	$\mathtt{case}\ e\,\{\,x_1\,.\,e_1 \mid x_2\,.\,e_2\,\}$	\caseEx[\tau][\rho]{e}{x}{e}
$\mathtt{case}[\tau_1\,;\,\tau_2\,;\,\rho](e\,;\,x_1\,.\,e_1\,;\,x_2\,.\,e_2)$	$\mathtt{case}\ e\,\{\,x_1\,.\,e_1 \mid x_2\,.\,e_2\,\}$	\xcaseEx[\tau_1][\tau_2][\rho]{e}{x_1}{e_1}{x_2}{e_2}
$\mathtt{bool}$	2	\boolTy
$\mathtt{true}$	$\mathtt{true}$	\trueEx
$\mathtt{false}$	$\mathtt{false}$	\falseEx
$\mathtt{if}[\rho](e\,;\,e_1\,;\,e_2)$	$\mathtt{if}(e\,;\,e_1\,;\,e_2)$	\ifEx[\rho]{e}{e_1}{e_2}
$\mathtt{vsum}\langle I\rangle(\tau_I)$	$+_{i\in I}(i \hookrightarrow \tau_i)$	\vsumTy<I><i>{\tau}
$\mathtt{vinj}\langle I\,;\,i\rangle[\tau_I](e)$	$i \cdot e$	\vinjEx<I><i>[\tau]{e} $(i \in I)$
$\mathtt{vcase}\langle I\rangle[\tau_I\,;\,\rho](e\,;\,(x\,.\,e')_I)$	$\mathtt{vcase}\ e\,\{\,i \hookrightarrow (x\,.\,e')_i \mid i \in I\,\}$	\vcaseEx<I><i>[\tau][\rho]{e}{x}{e'}

Figure 1: Product and Sum Types

Abstract	*Concrete*	*Invocation*
$\mathtt{fun}(\tau_1\,;\,\tau_2)$	$\tau_1 \to \tau_2$	\arrTy{\tau_1}{\tau_2}
$\mathtt{lam}[\tau_1\,;\,\tau_2](x\,.\,e)$	$\lambda(x\,.\,e)$	\lamEx[\tau_1][\tau_2]{x}{e}
$\mathtt{ap}[\tau_1\,;\,\tau_2](e_1\,;\,e_2)$	$\mathtt{ap}(e_1\,;\,e_2)$	\appEx[\tau_1][\tau_2]{e_1}{e_2}
$\mathtt{all}(t\,.\,\tau)$	$\forall(t\,.\,\tau)$	\allTy{t}{\tau}
$\mathtt{tlam}(t\,.\,e)$	$\Lambda(t\,.\,e)$	\tlamEx{t}[\tau]{e}
$\mathtt{tap}[t\,.\,\tau](e\,;\,\rho)$	$\mathtt{tap}(e\,;\,\rho)$	\tapEx[t][\tau]{e}{\rho}
$\mathtt{some}(t\,.\,\tau)$	$\exists(t\,.\,\tau)$	\someTy{t}{\tau}
$\mathtt{pack}[t\,.\,\tau](\rho\,;\,e)$	$\mathtt{pack}(\rho\,;\,e)$	\packEx[t][\tau]{\rho}{e}
$\mathtt{open}[t\,.\,\tau\,;\,\rho](e\,;\,t, x\,.\,e')$	$\mathtt{open}(e\,;\,t, x\,.\,e')$	\openEx[t][\tau][\rho]{e}{t}{x}{e'}

Figure 2: Function, Universal, and Existential Types

Abstract	*Concrete*	*Invocation*
$\texttt{ind}(t\,.\,\tau)$	$\texttt{ind}(t\,.\,\tau)$	`\indTy{t}{\tau}`
$\texttt{in}[t\,.\,\tau](e)$	$\texttt{in}(e)$	`\inEx[t][\tau]{e}`
$\texttt{rec}[t\,.\,\tau\ ;\ \rho](e\ ;\ x\,.\,e')$	$\texttt{rec}(e\ ;\ x\,.\,e')$	`\recEx[t][\tau][\rho]{e}{x}{e’}`
$\texttt{nat}$	$\texttt{nat}$	`\natTy`
$\texttt{zero}$	$\texttt{zero}$	`\zeroEx`
$\texttt{succ}(e)$	$\texttt{succ}(e)$	`\succEx{e}`
$\texttt{natrec}[\rho](e\ ;\ e_0\ ;\ x,y\,.\,e_1)$	$\texttt{natrec}\ e\,\{\,e_0 \mid x,y\,.\,e_1\,\}$	`\natrecEx[\rho]{e}{e_0}{x}{y}{e_1}`
$\texttt{natit}[\rho](e\ ;\ e_0\ ;\ x\,.\,e_1)$	$\texttt{natit}\ e\,\{\,e_0 \mid x\,.\,e_1\}$	`\natitEx[\rho]{e}{e_0}{x}{e_1}`
$\texttt{ifz}[\rho](e\ ;\ e_0\ ;\ x\,.\,e_1)$	$\texttt{ifz}\ e\,\{\,e_0 \mid x\,.\,e_1\}$	`\ifzEx[\rho]{e}{e_0}{x}{e_1}`
$\texttt{list}(\tau)$	$\tau\ \texttt{list}$	`\listTy{\tau}`
$\texttt{nil}[\tau]$	$\texttt{nil}$	`\nilEx[\tau]`
$\texttt{cons}[\tau](e_1\ ;\ e_2)$	$\texttt{cons}(e_1\ ;\ e_2)$	`\consEx[\tau]{e_1}{e_2}`
$\texttt{listrec}[\rho](e\ ;\ e_1\ ;\ x,y\,.\,e_2)$	$\texttt{listrec}\ e\,\{\,e_1 \mid x,y\,.\,e_2\}$	`\listrecEx[\rho]{e}{e_1}{x}{y}{e_2}`
$\texttt{listcase}[\rho](e\ ;\ e_1\ ;\ x,y\,.\,e_2)$	$\texttt{listcase}\ e\,\{\,e_1 \mid x,y\,.\,e_2\}$	`\listcaseEx[\rho]{e}{e_1}{x}{y}{e_2}`
$\texttt{coi}(t\,.\,\tau)$	$\texttt{coi}(t\,.\,\tau)$	`\coiTy{t}{\tau}`
$\texttt{out}[t\,.\,\tau](e)$	$\texttt{out}(e)$	`\outEx[t][\tau]{e}`
$\texttt{gen}[t\,.\,\tau\ ;\ \sigma](e\ ;\ x\,.\,e')$	$\texttt{gen}(e\ ;\ x\,.\,e')$	`\genEx[t][\tau][\sigma]{e}{x}{e’}`
$\texttt{conat}$	$\texttt{conat}$	`\conatTy`
$\texttt{stream}(\tau)$	$\tau\ \texttt{stream}$	`\streamTy{\tau}`

Figure 3: Inductive and Coinductive Types

Abstract	*Concrete*	*Invocation*
$\texttt{tens}(\tau_1\ ;\ \tau_2)$	$\tau_1 \otimes \tau_2$	`\tensorTy{\tau_1}{\tau_2}`
$\texttt{with}[\tau_1\ ;\ \tau_2](v_1\ ;\ v_2)$	$v_1 \otimes v_2$	`\tensorEx[\tau_1][\tau_2]{v_1}{v_2}`
$\texttt{split}[\tau_1\ ;\ \tau_2\ ;\ \rho](v\ ;\ x_1,x_2\,.\,e)$	$\texttt{split}\,v\,\{\,x_1,x_2\,.\,e\,\}$	`\splitEx[\tau_1][\tau_2][\rho]{v}{x_1}{x_2}{e}`
$\texttt{both}(\tau_1\ ;\ \tau_2)$	$\tau_1\ \&\ \tau_2$	`\bothTy{\tau_1}{\tau_2}`
$\texttt{both}(e_1\ ;\ e_2)$	$e_1\ \&\ e_2$	`\bothEx{e_1}{e_2}`
$\texttt{par}(e\ ;\ x,y\,.\,e')$	$\texttt{par}(e\ ;\ x,y\,.\,e')$	`\parEx{e}{x}{y}{e’}`

Figure 4: Parallel Types

Abstract	*Concrete*	*Invocation*
$\mathtt{parr}(\tau_1\,;\,\tau_2)$	$\tau_1 \rightharpoonup \tau_2$	`\parrTy{\tau_1}{\tau_2}`
$\mathtt{fun}[\tau_1\,;\,\tau_2](f, x\,.\,e)$	$\mathtt{fun}(f, x\,.\,e)$	`\funEx[\tau_1][\tau_2]{f}{x}{e}`
$\mathtt{ap}[\tau_1\,;\,\tau_2](e_1\,;\,e_2)$	$\mathtt{ap}(e_1\,;\,e_2)$	`\papEx[\tau_1][\tau_2]{e_1}{e_2}`
$\mathtt{fix}[\tau](x\,.\,e)$	$\mathtt{fix}(x\,.\,e)$	`\fixEx[\tau]{x}{e}`
$\mathtt{rec}(t\,.\,\tau)$	$\mathtt{rec}(t\,.\,\tau)$	`\recTy{t}{\tau}`
$\mathtt{fold}[t\,.\,\tau](e)$	$\mathtt{fold}(e)$	`\foldEx[t][\tau]{e}`
$\mathtt{unfold}[t\,.\,\tau](e)$	$\mathtt{unfold}(e)$	`\unfoldEx[t][\tau]{e}`
$\mathtt{self}(\tau)$	$\mathtt{self}(\tau)$	`\selfTy{\tau}`
$\mathtt{unroll}[\tau](e)$	$\mathtt{unroll}(e)$	`\unrollEx[\tau]{e}`
$\mathtt{self}[\tau](x\,.\,e)$	$\mathtt{self}(x\,.\,e)$	`\selfEx[\tau]{x}{e}`
$\mathtt{lam}(x\,.\,M)$	$\lambda(x\,.\,M)$	`\ulamEx{x}{M}`
$\mathtt{app}(M_1\,;\,M_2)$	$M_1(M_2)$	`\uapEx{M_1}{M_2}`
$\mathtt{I}$	$\mathtt{I}$	`\uIEx`
$\mathtt{K}$	$\mathtt{K}$	`\uKEx`
$\mathtt{S}$	$\mathtt{S}$	`\uSEx`
$\mathtt{B}$	$\mathtt{B}$	`\uBEx`

Figure 5: Partial and Recursive Types

Abstract	*Concrete*	*Invocation*
$\mathtt{ans}$	$\mathtt{ans}$	`\ansTy`
$\mathtt{yes}$	$\mathtt{yes}$	`\yesEx`
$\mathtt{no}$	$\mathtt{no}$	`\noEx`
$\mathtt{cont}(\tau)$	$\tau\;\mathtt{cont}$	`\contTy{\tau}`
$\mathtt{cont}[\tau](k)$	$\mathtt{cont}(k)$	`\contEx[\tau]{k}`
$\mathtt{letcc}[\tau](x\,.\,e)$	$\mathtt{letcc}(x\,.\,e)$	`\letccEx[\tau]{x}{e}`
$\mathtt{throw}[\tau\,;\,\rho](e_1\,;\,e_2)$	$\mathtt{throw}(e_1\,;\,e_2)$	`\throwEx[\tau][\rho]{e_1}{e_2}`
$\mathtt{emp}[\tau]$	$\bullet$	`\empStk[\tau]`
$\mathtt{ext}[\tau_1\,;\,\tau_2](k\,;\,f)$	$k\,;\,f$	`\extStk[\tau_1][\tau_2]{k}{f}`

Figure 6: Continuation Types

Abstract	*Concrete*	*Invocation*
$\mathtt{susp}(\tau)$	$\mathtt{susp}(\tau)$	`\suspTy{\tau}`
$\mathtt{susp}[\tau](e)$	$\mathtt{susp}(e)$	`\suspEx[\tau]{e}`
$\mathtt{force}[\tau](e)$	$\mathtt{force}(e)$	`\forceEx[\tau]{e}`
$\mathtt{comp}(\tau)$	$\mathtt{comp}(\tau)$	`\compTy{\tau}`
$\mathtt{comp}(m)$	$\mathtt{comp}(m)$	`\compEx{m}`
$\mathtt{ret}[\tau](e)$	$\mathtt{ret}(e)$	`\retEx[\tau]{e}`
$\mathtt{bind}[\tau](e_1\,;\,x\,.\,e_2)$	$\mathtt{bind}(e_1\,;\,x\,.\,e_2)$	`\bndEx[\tau]{e_1}{x}{e_2}`

Figure 7: Suspension and Computation Types

Abstract	*Concrete*	*Invocation*
$\top$	$\top$	\topTy
$\star$	$\star$	\topEx
$\mathtt{check}[\rho](e \mathbin{;} e')$	$\mathtt{split}\, e\, \{e'\}$	\checkEx[\rho]{e}{e'}
$\mathtt{F}(\tau)$	$\uparrow(\tau)$	\freeTy{\tau}
$\mathtt{ret}[\tau](v)$	$\mathtt{ret}(v)$	\freeEx[\tau]{v}
$\mathtt{bind}[\tau \mathbin{;} \rho](e \mathbin{;} x \,.\, e')$	$\mathtt{bind}\, x \leftarrow e \,;\, e'$	\fletEx[\tau][\rho]{e}{x}{e'}
$\mathtt{U}(\tau)$	$\downarrow(\tau)$	\thunkTy{\tau}
$\mathtt{susp}[\tau](e)$	$\mathtt{susp}(e)$	\thunkEx[\tau]{e}
$\mathtt{susp}\langle k\rangle[\tau](x \,.\, e)$	$\mathtt{susp}\langle k\rangle(x \,.\, e)$	\recthunkEx<k>[\tau]{x}{e}
$\mathtt{force}[\tau](v)$	$\mathtt{force}(v)$	\forceEx[\tau]{v}

Figure 8: Polarized Types

$\mathtt{sym}(\tau)$	$\tau\, \mathtt{sym}$	\symTy{\tau}
$\mathtt{quote}\langle a\rangle$	$\mathtt{'}a$	\quoteEx<a>
$\mathtt{is}\langle a\rangle[t \,.\, \tau](e \mathbin{;} m_1 \mathbin{;} m_2)$	$\mathtt{is}\langle a\rangle(e \mathbin{;} m_1 \mathbin{;} m_2)$	\ifisCmd<a>[t][\tau]{e}{m_1}{m_2}
$\mathtt{newsym}[\tau]$	$\mathtt{newsym}[\tau]$	\newsymCmd[\tau]
$\mathtt{eq}[t \,.\, \tau](e_1 \mathbin{;} e_2 \mathbin{;} m_1 \mathbin{;} m_2)$	$\mathtt{eq}(e_1 \mathbin{;} e_2 \mathbin{;} m_1 \mathbin{;} m_2)$	\ifeqCmd[t][\tau]{e_1}{e_2}{m_1}{m_2}

Figure 9: Symbols

Abstract	*Concrete*	*Invocation*
$\mathtt{cmd}(\tau)$	$\tau\ \mathtt{cmd}$	`\cmdTy{\tau}`
$\mathtt{cmd}[\tau](m)$	$\mathtt{cmd}(m)$	`\cmdEx[\tau]{m}`
$\mathtt{ret}[\tau](e)$	$\mathtt{ret}(e)$	`\retCmd[\tau]{e}`
$\mathtt{raise}[\rho](e)$	$\mathtt{raise}(e)$	`\raiseCmd[\rho]{e}`
$\mathtt{bnd}[\tau\ ;\ \rho](e\ ;\ x\ .\ m)$	$\mathtt{bnd}\ x \leftarrow e\ ;\ m$	`\bndCmd[\tau]{e}[\rho]{x}{m}`
$\mathtt{bndow}[\tau](e\ ;\ x\ .\ m_1\ ;\ x\ .\ m_2)$	$\mathtt{bnd}\ x \leftarrow e\ ;\ m_1\ \mathtt{ow}\ x \rightarrow m_2$	`\bndowCmd[\tau]{e}{x}{m_1}{x}{m_2}`
$\mathtt{dcl}[\tau\ ;\ \rho](e\ ;\ a\ .\ m)$	$\mathtt{dcl}\ a := e\ \mathtt{in}\ m$	`\dclCmd[\tau]{e}[\rho]<a>{m}`
$\mathtt{seq}(m_1\ ;\ x\ .\ m_2)$	$x \leftarrow m_1\ ;\ m_2$	`\seqCmd{m_1}[x]{m_2}`
$\mathtt{seq}(m_1\ ;\ \_\ .\ m_2)$	$m_1\ ;\ m_2$	`\seqCmd{m_1}{m_2}`
$\mathtt{ref}(\tau)$	$\tau\ \mathtt{ref}$	`\refTy{\tau}`
$\mathtt{ref}\langle a\rangle$	$\&\, a$	`\refEx<a>`
$\mathtt{get}\langle a\rangle$	$\mathtt{!}\, a$	`\getCmd<a>`
$\mathtt{getref}[\tau](e)$	$*e$	`\getrefCmd[\tau]{e}`
$\mathtt{set}\langle a\rangle(e)$	$a := e$	`\setCmd<a>{e}`
$\mathtt{setref}[\tau](e_1\ ;\ e_2)$	$e_1 \mathrel{*=} e_2$	`\setrefCmd[\tau]{e_1}{e_2}`
$\mathtt{newref}[\tau](e)$	$\mathtt{newref}(e)$	`\newrefCmd[\tau]{e}`
$\mathtt{do}\langle o\rangle(e)(x\ .\ m)$	$\mathtt{do}\ o(e)(x\ .\ m)$	`\doCmd<o>(e)[\Abs(x){m}]`
$\mathtt{hdl}\langle o\rangle[\tau](m\ ;\ x\ .\ m_1\ ;\ x,k\ .\ m_2)$	$\mathtt{hdl}\ m\ \{\ \mathtt{ret}(x) \hookrightarrow m_1 \mid o(x,k) \hookrightarrow m_2\ \}$	`\hdlCmd<o>[\tau]{m}{x}{m_1}{x}{k}{m_2}`

Figure 10: Command Types

Abstract	*Concrete*	*Invocation*
$\mathtt{st}$	$\mathtt{st}$	`\staticLock`
$\mathtt{type}$	$\mathtt{type}$	`\univSg`
$\mathtt{val}(\tau)$	$\mathtt{val}(\tau)$	`\valSg{\tau}`
$\mathtt{Ext}(S\ ;\ M)$	$\{S \mid M\}$	`\extSg{S}{M}`
$\mathtt{in}[S\ ;\ M](M')$	$\mathtt{in}(M')$	`\inMd[S][M]{M'}`
$\mathtt{out}[S\ ;\ M](M')$	$\mathtt{out}(M')$	`\outMd[S][M]{M'}`
$\mathtt{Comp}(S)$	$\mathtt{Comp}(S)$	`\compSg{S}`
$\mathtt{Pi}(S_1\ ;\ X\ .\ S_2)$	$X : S_1 \rightarrow S_2$	`\piSg{S_1}[X]{S_2}`
$\mathtt{Pi}(S_1\ ;\ \_\ .\ S_2)$	$S_1 \rightarrow S_2$	`\piSg{S_1}{S_2}`
$\mathtt{func}[S_1\ ;\ X\ .\ S_2](X\ .\ M_2)$	$\mathtt{func}\ X : S_1\ \mathtt{in}\ M_2$	`\funMd{S_1}{X}{S_2}{M_2}`
$\mathtt{inst}[S_1\ ;\ X\ .\ S_2](M_1\ ;\ M_2)$	$M_1(M_2)$	`\instMd[S_1][X][S_2]{M_1}{M_2}`
$\mathtt{Sig}(S_1\ ;\ X\ .\ S_2)$	$X : S_1 \times S_2$	`\sigSg{S_1}[X]{S_2}`
$\mathtt{Sig}(S_1\ ;\ \_\ .\ S_2)$	$S_1 \times S_2$	`\sigSg{S_1}{S_2}`
$\mathtt{struct}[S_1\ ;\ X\ .\ S_2](M_1\ ;\ M_2)$	$\mathtt{struct}\ M_1\ ;\ M_2$	`\strMd[S_1][X][S_2]{M_1}{M_2}`
$\mathtt{proj}\langle i\rangle[S_1\ ;\ X\ .\ S_2](M)$	$M \cdot \mathtt{i}$	`\projMd<i>[S_1][X][S_2]{M}` $(i = 1, 2)$

Figure 11: Module Types

Abstract	*Concrete*	*Invocation*
$\mathtt{truth}$	$\top$	\trueProp
$\mathtt{truthI}$	$\top\mathtt{I}$	\trueIPf
$\mathtt{and}(\phi_1 ; \phi_2)$	$\phi_1 \wedge \phi_2$	\andProp{\phi_1}{\phi_2}
$\mathtt{andI}[\phi_1 ; \phi_2](\pi_1 ; \pi_2)$	$\wedge\mathtt{I}(\pi_1 ; \pi_2)$	\andIPf[\phi_1][\phi_2]{\pi_1}{\pi_2}
$\mathtt{andE}\langle i\rangle[\phi_1 ; \phi_2](\pi)$	$\wedge\mathtt{E}\langle i\rangle(\pi)$	\andEPf[\phi_1][\phi_2]<i>{\pi}
$\mathtt{falsity}$	$\perp$	\falseProp
$\mathtt{falseI}[\phi](\pi)$	$\perp\mathtt{I}(\pi)$	\falseIPf[\phi]{\pi}
$\mathtt{falseE}[\phi](\pi)$	$\perp\mathtt{E}(\pi)$	\falseEPf[\phi]{\pi}
$\mathtt{or}(\phi_1 ; \phi_2)$	$\phi_1 \vee \phi_2$	\orProp{\phi_1}{\phi_2}
$\mathtt{orI}\langle i\rangle[\phi_1 ; \phi_2](\pi)$	$\vee\mathtt{I}\langle i\rangle(\pi)$	\orIPf[\phi_1][\phi_2]<i>{\pi}
$\mathtt{orE}[\phi_1 ; \phi_2 ; \rho](\pi ; x . \pi_1 ; x . \pi_2)$	$\vee\mathtt{E}(\pi ; x . \pi_1 ; x . \pi_2)$	\orEPf[\phi_1][\phi_2][\rho]{\pi}{x}{\pi_1}{x}{\pi_2}
$\mathtt{imp}(\phi_1 ; \phi_2)$	$\phi_1 \supset \phi_2$	\impProp{\phi_1}{\phi_2}
$\mathtt{impI}[\phi_1 ; \phi_2](x . \pi_2)$	$\supset\mathtt{I}(x . \pi_2)$	\impIPf[\phi_1][\phi_2]{x}{\pi_2}
$\mathtt{impE}[\phi_1 ; \phi_2](\pi ; \pi_1)$	$\supset\mathtt{E}(\pi ; \pi_2)$	\impEPf[\phi_1][\phi_2]{\pi}{\pi_1}
$\mathtt{not}(\phi)$	$\neg\phi$	\notProp{\phi}
$\mathtt{notI}[\phi](x . \pi)$	$\neg\mathtt{I}(x . \pi)$	\notIPf[\phi]{x}{\pi}
$\mathtt{notE}[\phi](\pi ; \pi_2)$	$\neg\mathtt{E}(\pi ; \pi_2)$	\notEPf[\phi]{\pi}{\pi_2}

Figure 12: Propositions and Proofs

Abstract	*Concrete*	*Invocation*
$\mathtt{one}$	1	`\unitPr`
$\mathtt{par}(p_1 \mathbin{;} p_2)$	$p_1 \otimes p_2$	`\parPr{p_1}{p_2}`
$\mathtt{null}$	0	`\nullPr`
$\mathtt{or}(p_1 \mathbin{;} p_2)$	$p_1 \oplus p_2$	`\choosePr{p_1}{p_2}`
$\mathtt{que}\langle a\rangle(p)$	$?\langle a\rangle(p)$	`\quePr{a}{p}`
$\mathtt{sig}\langle a\rangle(p)$	$!\langle a\rangle(p)$	`\sigPr{a}{p}`
$\mathtt{rcv}\langle a\rangle(x \,.\, p)$	$?\langle a\rangle(x \,.\, p)$	`\rcvPr{a}{x}{p}`
$\mathtt{snd}\langle a\rangle(e \mathbin{;} p)$	$!\langle a\rangle(e \mathbin{;} p)$	`\sndPr{a}{e}{p}`
$\mathtt{asnd}\langle a\rangle(e)$	$!\langle a\rangle(e)$	`\asndPr{a}{e}`
$\mathtt{sync}(e)$	$\mathtt{sync}(e)$	`\syncPr{e}`
$\mathtt{new}[\tau](a \,.\, p)$	$\nu(a \,.\, p)$	`\newPr[\tau]{a}{p}`
$\mathtt{sil}$	ϵ	`\silAc`
$\mathtt{sig}\langle a\rangle$	$a\,!$	`\sigAc{a}`
$\mathtt{que}\langle a\rangle$	$a\,?$	`\queAc{a}`
$\mathtt{snd}\langle a\rangle(e)$	$a\,!\,e$	`\sndAc{a}{e}`
$\mathtt{rcv}\langle a\rangle(e)$	$a\,?\,e$	`\rcvAc{a}{e}`
$\mathtt{any}(e)$	$e\,?$	`\anyAc{e}`
$\mathtt{emit}(e)$	$e\,!$	`\emitAc{e}`

Figure 13: Processes

$\mathtt{emit}[\tau](e)$	$\mathtt{emit}(e)$	`\emitCmd[\tau]{e}`
$\mathtt{acc}[\tau]$	$\mathtt{acc}$	`\accCmd[\tau]`
$\mathtt{sync}[\tau](e)$	$\mathtt{sync}(e)$	`\syncCmd[\tau]{e}`
$\mathtt{spawn}[\tau](e)$	$\mathtt{spawn}(e)$	`\spawnCmd[\tau]{e}`
$\mathtt{newch}[\tau]$	$\mathtt{newch}_{\tau}$	`\newchCmd{\tau}`
$\mathtt{send}\langle a\rangle(e)$	$\mathtt{send}\langle a\rangle(e)$	`\sndCmd<a>{e}`
$\mathtt{recv}\langle a\rangle(x \,.\, e)$	$\mathtt{recv}\langle a\rangle(x \,.\, e)$	`\rcvCmd<a>{x}{e}`
$\mathtt{rcv}\langle a\rangle$	$\mathtt{rcv}\langle a\rangle$	`\rcvEv<a>`
$\mathtt{snd}\langle a\rangle(v)$	$\mathtt{snd}\langle a\rangle(v)$	`\sndEv<a>`
$\mathtt{or}[\tau](e_1 \mathbin{;} e_2)$	$e_1 \oplus e_2$	`\orEv[\tau]{e_1}{e_2}`
$\mathtt{wrap}[\tau](e_1 \mathbin{;} x \,.\, e_2)$	$\mathtt{wrap}[\tau](e_1 \mathbin{;} x \,.\, e_2)$	`\wrapEv[\tau]{e_1}{x}{e_2}`

Figure 14: Concurrent Algol