PFPL Syntax Package*

Robert Harper

December 24, 2022

Overview

Write `\usepackage{pfpl-syntax}` to define syntax macros for PFPL. 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{kwd}`: Format an operator as a keyword, or `\Opn*{symbol}` for an operator whose display is in math mode. For example, `\Opn{fun}` produces `fun` and `\Opn*{\lambda}` produces λ.

- \Abt<parameters>[optional](arguments): produce abstract syntax, with all arguments typeset, and the starred form to omit the optional arguments. Omitted arguments are omitted. For example, the command

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

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

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

  produces $\mathtt{set}\langle a\rangle[\tau](e)$. The starred forms produce $\mathtt{get}\langle a\rangle$, and $\mathtt{set}\langle a\rangle(e)$, 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 $\mathtt{dcl}[\tau;\rho](e;a\,.\,m)$ and the starred form produces $\mathtt{dcl}(e;a\,.\,m)$.

These general forms are not ordinarily used directly, they are rather for authors to define their own syntax macros, as illustrated for PFPL in the next section.

Language-Specific Definitions

The commands for formatting the many languages considered in PFPL are formulated in a series of figures organized around type structure. The tables display the abstract and concrete syntax, and the literal command used to create them (with starred forms for the concrete syntax).

$\texttt{fun}(\tau_1\,;\tau_2)$	$\tau_1 \to \tau_2$	\arrTy{\tau_1}{\tau_2}
$\texttt{lam}[\tau_1\,;\tau_2](x\,.\,e)$	$\lambda(x\,.\,e)$	\lamEx{x}{e}
$\texttt{ap}[\tau_1\,;\tau_2](e_1\,;e_2)$	$\texttt{ap}(e_1\,;e_2)$	\appEx{e_1}{e_2}

Figure 1: Function Types

$\texttt{unit}$	$\mathbf{1}$	\unitTy
$\texttt{null}$	$\langle\rangle$	\unitEx
$\texttt{prod}(\tau_1\,;\tau_2)$	$\tau_1 \times \tau_2$	\prodTy{\tau_1}{\tau_2}
$\texttt{proj}\langle i\rangle[\tau_1\,;\tau_2](e)$	$e \cdot i$	\projEx<i>{e} $(i = 1, 2)$
$\texttt{pair}[\tau_1\,;\tau_2](e_1\,;e_2)$	$\langle e_1, e_2\rangle$	\pairEx[\tau_1][\tau_2]{e_1}{e_2}
$\texttt{vprod}\langle I\rangle(\tau_I)$	$\times_{i\in I}(i \hookrightarrow \tau_i)$	\vprodTy<I><i>{\tau}
$\texttt{vtuple}\langle I\rangle[\tau_I](e_I)$	$\langle i \hookrightarrow e_i \mid i \in I\rangle$	\vtupleEx<I>[\tau]{e}
$\texttt{vproj}\langle I\,; i\rangle[\tau_I](e)$	$e \cdot i$	\vprojEx<I><i>[\tau]{e} $(i \in I)$

The notation τ_I stands for the finite map $i \hookrightarrow \tau_i \mid i \in I$ and e_I stands for $i \hookrightarrow e_i \mid i \in I$.

Figure 2: Product Types

$\texttt{void}$	$\mathbf{0}$	\voidTy
$\texttt{absurd}[\rho](e)$	$\texttt{absurd}(e)$	\absurdEx[\rho]{e}
$\texttt{sum}(\tau_1\,;\tau_2)$	$\tau_1 + \tau_2$	\sumTy{\tau_1}{\tau_2}
$\texttt{inj}\langle i\rangle[\tau_1\,;\tau_2](e)$	$i \cdot e$	$(i = 1, 2)$
$\texttt{case}[\tau_1\,;\tau_2\,;\rho](e\,; x\,.\,e_1\,; x\,.\,e_2)$	$\texttt{case}\ e\,\{\, x\,.\,e_1 \mid x\,.\,e_2\,\}$	\caseEx[\tau][\rho]{e}{x}{e}
$\texttt{bool}$	$\mathbf{2}$	\boolTy
$\texttt{true}$	$\texttt{true}$	\trueEx
$\texttt{false}$	$\texttt{false}$	\falseEx
$\texttt{if}[\rho](e\,;e_1\,;e_2)$	$\texttt{if}(e\,;e_1\,;e_2)$	\ifEx[\rho]{e}{e_1}{e_2}
$\texttt{vsum}\langle I\rangle(\tau_I)$	$+_{i\in I}(i \hookrightarrow \tau_i)$	\vsumTy<I><i>{\tau}
$\texttt{vinj}\langle I\,; i\rangle[\tau_I](e)$	$i \cdot e$	\vinjEx<I><i>[\tau]{e} $(i \in I)$
$\texttt{vcase}\langle I\rangle[\tau_I\,;\rho](e\,; x\,.\,e'_I)$	$\texttt{vcase}\ e\,\{\, i \hookrightarrow x\,.\,e'_i \mid i \in I\,\}$	\vcaseEx<I>[\rho][\tau]{e}{x}{e'}

Figure 3: Sum Types

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

Figure 4: Inductive and Coinductive Types

$\mathtt{All}(t\,.\,\tau)$	$\forall(t\,.\,\tau)$	`\AllTy{t}{\tau}`
$\mathtt{Lam}(t\,.\,e)$	$\Lambda(t\,.\,e)$	`\LamEx{t}[\tau]{e}`
$\mathtt{Ap}[t\,.\,\tau](e\,;\sigma)$	$\mathtt{Ap}(e\,;\sigma)$	`\AppEx[t][\tau]{e}{\sigma}`
$\mathtt{Some}(t\,.\,\tau)$	$\exists(t\,.\,\tau)$	`\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]{e}[\rho]{x}{e'}`

Figure 5: Polymorphic Types

$\mathtt{ans}$	$\mathtt{ans}$	`\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{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]`
$\mathtt{ext}[\tau_1\,;\tau_2](k\,;f)$	$k\ \mathtt{;}\ f)$	`\extStk[\tau_1][\tau_2]{k}{f}`

Figure 6: Continuation Types

$\mathtt{parr}(\tau_1\,;\tau_2)$	$\tau_1 \rightharpoonup \tau_2$	`\parrTy{\tau_1}{\tau_2}`
$\mathtt{fun}[\tau_1\,;\tau_2](f,x\,.\,e)$	$\mathtt{fun}(f,x\,.\,e)$	`\funEx{f}{x}{e}`
$\mathtt{ap}[\tau_1\,;\tau_2](e_1\,;e_2)$	$\mathtt{ap}(e_1\,;e_2)$	`\appEx{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}`
$\mathtt{roll}[\tau](e)$	$\mathtt{roll}(e)$	`\rollEx[\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 7: Recursive Types

$\mathtt{cmd}(\tau)$	τ `cmd`	`\cmdTy{\tau}`
$\mathtt{cmd}[\tau](m)$	$\mathtt{cmd}(m)$	`\cmdEx[\tau]{m}`
$\mathtt{ret}[\tau](e)$	$\mathtt{ret}(e)$	`\retCmd[\tau]{e}`
$\mathtt{bnd}[\tau\,;\rho](e\,;x\,.\,m)$	$\mathtt{bnd}\ x \leftarrow e\ \mathtt{;}\ m$	`\bndCmd{e}{x}{m}`
$\mathtt{dcl}[\tau\,;\rho](e\,;a\,.\,m)$	$\mathtt{dcl}\ a\ \mathtt{:=}\ e\ \mathtt{;}\ m$	`\dclCmd{e}{a}{m}`
$\mathtt{ref}(\tau)$	τ `ref`	`\refTy{\tau}`
$\mathtt{ref}\langle a\rangle$	$\mathtt{\&}\,a$	`\refEx{a}`
$\mathtt{get}\langle a\rangle$	$\mathtt{!}\,a$	`\getCmd{a}`
$\mathtt{getref}[\tau](e)$	$\mathtt{*}e$	`\getrefCmd[\tau]{e}`
$\mathtt{set}\langle a\rangle(e)$	$a\ \mathtt{:=}\ e$	`\setCmd{a}{e}`
$\mathtt{setref}[\tau](e_1\,;e_2)$	$e_1\ \mathtt{*=}\ e_2$	`\setrefCmd[\tau]{e_1}{e_2}`

Symbols a are constants of sort `loc`.

Figure 8: Command Types

`st`	`st`	`\staticLock`
`type`	`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'}`
$\mathtt{Comp}(S)$	$S\,\mathtt{Comp}$	`\compSg{S}`
$\mathtt{Pi}(S_1\,;X\,.\,S_2)$	$X{:}S_1 \rightarrow S_2$	`\piSg{S_1}{X}{S_2}`
$\mathtt{fun}[S_1](X\,.\,M_2)$	$\mathtt{fun}\,X : S_1\ \mathtt{in}\ M_2$	`\funMd{S_1}{X}{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}`
$\mathtt{str}[S_1\,;X\,.\,S_2](M_1\,;M_2)$	$\mathtt{str}\,M_1\ \mathtt{in}\ M_2$	`\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 9: Module Types

$\mathtt{one}$	1	`\unitPr`
$\mathtt{par}(p_1\,;p_2)$	$p_1 \otimes p_2$	`\parPr{p_1}{p_2}`
$\mathtt{null}$	0	`\nullPr`
$\mathtt{or}(p_1\,;p_2)$	$p_1 \oplus p_2$	`\choosePr{p_1}{p_2}`
$\mathtt{que}\langle a\rangle(p)$	$?\langle a\rangle(p)$	`\quePr{a}{p}`
$\mathtt{sig}\langle a\rangle(p)$	$!\langle a\rangle(p)$	
$\mathtt{rcv}\langle a\rangle(x\,.\,p)$	$?\langle a\rangle(x\,.\,p)$	`\sigPr{a}{p}`
$\mathtt{snd}\langle a\rangle(e\,;p)$	$!\langle a\rangle(e\,;p)$	`\sndPr{a}{e}{p}`
$\mathtt{asnd}\langle a\rangle(e)$	$!\langle a\rangle(e)$	`\asndPr{a}{e}`
$\mathtt{sync}(\varepsilon)$	$\mathtt{sync}(\varepsilon)$	`\syncPr{\varepsilon}`
$\mathtt{new}[\tau](a\,.\,p)$	$\nu(a\,.\,p)$	`\newPr{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}`
$\mathtt{rcv}\langle a\rangle(e)$	$a\,?\,e$	
$\mathtt{any}(e)$	$?\,e$	`\rcvAc{a}{e}`
$\mathtt{emit}(e)$	$!\,e$	`\emitAc{e}`

Figure 10: Proceses

$\mathtt{F}(\tau^{+})$	$\uparrow(\tau^{+})$	`\freeTy{\posTy{\tau}}`
$\mathtt{ret}[\tau^{+}](v)$	$\mathtt{ret}(v)$	`\freeEx[\posTy{\tau}]{v}`
$\mathtt{let}[\tau^{-}\,;\rho^{-}](e\,;x\,.\,e')$	$\mathtt{let}(e\,;x\,.\,e')$	`\fletEx[\negTy{\tau}][\negTy{\rho}]{e}{x}{e'}`
$\mathtt{U}(\tau^{-})$	$\downarrow(\tau^{-})$	`\thunkTy{\negTy{\tau}}`
$\mathtt{thunk}[\tau^{-}](e)$	$\mathtt{thunk}(e)$	`\thunkEx[\negTy{\tau}]{e}`
$\mathtt{force}(v)$	$\mathtt{force}(v)$	$\mathtt{force}(v)$

Figure 11: Polarized Types

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

Figure 12: Parallel Types

$\mathtt{truth}$	$\top$	\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{\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}
$\mathtt{falsity}$	$\bot$	\falseProp
$\mathtt{falseI}[\phi](\pi)$	$\bot\mathtt{I}(\pi)$	\falseIPf[\phi]{\pi}
$\mathtt{falseE}[\phi](\pi)$	$\bot\mathtt{E}(\pi)$	\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]{\pi}{x}{\pi_1}{x}{\
$\mathtt{imp}(\phi_1\,;\phi_2)$	$\phi_1 \supset \phi_2$	\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 13: Propositions and Proofs