PFPL Syntax Package[*]

Robert Harper

December 29, 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; use starred form for an operator whose display is to be set 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; the starred form omits the optional arguments. If the parnethesized arguments are empty, then no parentheses are typeset. 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 $\texttt{dcl}[\tau;\rho](e;a\,.\,m)$ 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. These 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).

When I is a finite set of indices, the notation τ_I stands for the finite map $i \hookrightarrow \tau_i \mid i \in I$ and e_I stands for $i \hookrightarrow e_i \mid i \in I$.

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

Figure 1: Function Types

unit	$\mathbf{1}$	\unitTy
null	$\langle\rangle$	\unitEx
$\mathtt{prod}(\tau_1\,;\tau_2)$	$\tau_1 \times \tau_2$	\prodTy{\tau_1}{\tau_2}
$\mathtt{proj}\langle i\rangle[\tau_1\,;\tau_2](e)$	$e \cdot i$	\projEx<i>{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>[\tau]{e}
$\mathtt{vproj}\langle I\,;i\rangle[\tau_I](e)$	$e \cdot i$	\vprojEx<I><i>[\tau]{e} $(i \in I)$

Figure 2: Product Types

void	$\mathbf{0}$	\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} $(i = 1,$
$\mathtt{case}[\tau_1\,;\tau_2\,;\rho](e\,;x\,.\,e_1\,;x\,.\,e_2)$	$\mathtt{case}\ e\,\{\,x\,.\,e_1 \mid x\,.\,e_2\,\}$	\caseEx[\tau][\rho]{e}{x}{e}
bool	$\mathbf{2}$	\boolTy
true	true	\trueEx
false	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>[\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}`
$\mathtt{tlam}(t\,.\,e)$	$\Lambda(t\,.\,e)$	`\LamEx{t}[\tau]{e}`
$\mathtt{tap}[t\,.\,\tau](e\,;\sigma)$	$\mathtt{tap}(e\,;\sigma)$	`\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

ans	ans	`\ansTy`
yes	yes	`\yesEx`
no	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}`
I	I	`\uIEx`
K	K	`\uKEx`
S	S	`\uSEx`
B	B	`\uBEx`

Figure 7: Recursive Types

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

Figure 8: Command Types

`st`	`st`	`\staticLock`
`type`	`type`	`\univSg`
$\texttt{val}(\tau)$	$\texttt{val}(\tau)$	`\valSg{\tau}`
$\texttt{Ext}(S\,;M)$	$\{S \mid M\}$	`\extSg{S}{M}`
$\texttt{in}[S\,;M](M')$	$\texttt{in}(M')$	`\inMd[S][M]{M'}`
$\texttt{out}[S\,;M](M')$	$\texttt{out}(M')$	`\outMd[S][M]{M'}`
$\texttt{Comp}(S)$	$S\,\texttt{Comp}$	`\compSg{S}`
$\texttt{Pi}(S_1\,;X\,.\,S_2)$	$X{:}S_1 \to S_2$	`\piSg{S_1}{X}{S_2}`
$\texttt{fun}[S_1](X\,.\,M_2)$	$\texttt{fun}\,X : S_1\ \texttt{in}\ M_2$	`\funMd{S_1}{X}{M_2}`
$\texttt{inst}[S_1\,;X\,.\,S_2](M_1\,;M_2)$	$M_1(M_2)$	`\instMd[S_1][X][S_2]{M_1}{M_2}`
$\texttt{Sig}(S_1\,;X\,.\,S_2)$	$X{:}S_1 \times S_2$	`\sigSg{S_1}{X}{S_2}`
$\texttt{str}[S_1\,;X\,.\,S_2](M_1\,;M_2)$	$\texttt{str}\,M_1\ \texttt{in}\ M_2$	`\strMd[S_1][X][S_2]{M_1}{M_2}`
$\texttt{proj}\langle i\rangle[S_1\,;X\,.\,S_2](M)$	$M \cdot \texttt{i}$	`\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}`
$\mathtt{rcv}\langle a\rangle(e)$	$a\,?\,e$	`\rcvAc{a}{e}`
$\mathtt{any}(e)$	$?\,e$	`\anyAc{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}`
$\mathtt{force}(v)$	$\mathtt{force}(v)$	`\forceEx{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

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

Figure 13: Propositions and Proofs