

PFPL Syntax Package*

Robert Harper

December 23, 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 `get` $\langle a \rangle[\tau]$, and the command

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

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

*© 2022 Robert Harper. All Rights Reserved.

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

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).

$\text{fun}(\tau_1 ; \tau_2)$	$\tau_1 \rightarrow \tau_2$	$\backslash\text{arrTy}\{\backslash\tau_1\}\{\backslash\tau_2\}$
$\text{lam}[\tau_1 ; \tau_2](x . e)$	$\lambda(x . e)$	$\backslash\text{lamEx}\{x\}\{e\}$
$\text{ap}[\tau_1 ; \tau_2](e_1 ; e_2)$	$\text{ap}(e_1 ; e_2)$	$\backslash\text{appEx}\{e_1\}\{e_2\}$

Figure 1: Function Types

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

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

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

Figure 3: Sum Types

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

Figure 4: Inductive and Coinductive Types

$\text{All}(t.\tau)$	$\forall(t.\tau)$	$\backslash\text{AllTy}\{t\}\{\tau\}$
$\text{Lam}(t.e)$	$\Lambda(t.e)$	$\backslash\text{LamEx}\{t\}[\tau]\{e\}$
$\text{Ap}[t.\tau](e;\sigma)$	$\text{Ap}(e;\sigma)$	$\backslash\text{AppEx}[t][\tau]\{e\}\{\sigma\}$
$\text{Some}(t.\tau)$	$\exists(t.\tau)$	$\backslash\text{SomeTy}\{t\}\{\tau\}$
$\text{Pack}[t.\tau](\rho;e)$	$\text{Pack}(\rho;e)$	$\backslash\text{PackEx}[t][\tau]\{\rho\}\{e\}$
$\text{Open}[t.\tau;\rho](e;t,x.e')$	$\text{Open}(e;t,x.e')$	$\backslash\text{OpenEx}[t][\tau]\{e\}[\rho]\{x\}\{e'\}$

Figure 5: Polymorphic Types

ans	ans	<code>\ansTy</code>
yes	yes	<code>\yesEx</code>
no	no	<code>\noEx</code>
cont (τ)	τ cont	<code>\contTy{\tau}</code>
cont [τ](k)	cont (k)	<code>\contEx{k}</code>
letcc [τ]($x.e$)	letcc ($x.e$)	<code>\letccEx[\tau]{x}{e}</code>
throw [$\tau ; \rho$]($e_1 ; e_2$)	throw ($e_1 ; e_2$)	<code>\throwEx[\tau][\rho]{e_1}{e_2}</code>
emp [τ]	\bullet	<code>\empStk[\tau]</code>
ext [$\tau_1 ; \tau_2$]($k ; f$)	$k ; f$	<code>\extStk[\tau_1][\tau_2]{k}{f}</code>

Figure 6: Continuation Types

parr ($\tau_1 ; \tau_2$)	$\tau_1 \multimap \tau_2$	<code>\parrTy{\tau_1}{\tau_2}</code>
fun [$\tau_1 ; \tau_2$]($f, x.e$)	fun ($f, x.e$)	<code>\funEx{f}{x}{e}</code>
ap [$\tau_1 ; \tau_2$]($e_1 ; e_2$)	ap ($e_1 ; e_2$)	<code>\appEx{e_1}{e_2}</code>
fix [τ]($x.e$)	fix ($x.e$)	<code>\fixEx[\tau]{x}{e}</code>
rect .(τ)	rect .(τ)	<code>\recTy{t}{\tau}</code>
fold [$t . \tau$](e)	fold (e)	<code>\foldEx[t][\tau]{e}</code>
unfold [$t . \tau$](e)	unfold (e)	<code>\unfoldEx[t][\tau]{e}</code>
self (τ)	self (τ)	<code>\selfTy{\tau}</code>
roll [τ](e)	roll (e)	<code>\rollEx[\tau]{e}</code>
self [τ]($x.e$)	self ($x.e$)	<code>\selfEx[\tau]{x}{e}</code>
lam ($x.M$)	$\lambda(x.M)$	<code>\ulamEx{x}{M}</code>
app ($M_1 ; M_2$)	$M_1(M_2)$	<code>\uapEx{M_1}{M_2}</code>
I	I	<code>\uIEx</code>
K	K	<code>\uKEx</code>
S	S	<code>\uSEx</code>
B	B	<code>\uBEx</code>

Figure 7: Recursive Types

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

Symbols a are constants of sort `loc`.

Figure 8: Command Types

<code>st</code>	<code>st</code>	$\backslash\text{staticLock}$
<code>type</code>	<code>type</code>	$\backslash\text{univSg}$
$\text{val}(\tau)$	$\text{val}(\tau)$	$\backslash\text{valSg}\{\tau\}$
$\text{Ext}(S; M)$	$\{S \mid M\}$	$\backslash\text{extSg}\{S\}\{M\}$
$\text{in}[S; M](M')$	$\text{in}(M')$	$\backslash\text{inMd}[S][M]\{M'\}$
$\text{out}[S; M](M')$	$\text{out}(M')$	$\backslash\text{outMd}[S][M]\{M'\}$
$\text{Comp}(S)$	$S \text{ Comp}$	$\backslash\text{compSg}\{S\}$
$\text{Pi}(S_1; X.S_2)$	$X:S_1 \rightarrow S_2$	$\backslash\text{piSg}\{S_1\}\{X\}\{S_2\}$
$\text{fun}[S_1](X.M_2)$	$\text{fun } X : S_1 \text{ in } M_2$	$\backslash\text{funMd}\{S_1\}\{X\}\{M_2\}$
$\text{inst}[S_1; X.S_2](M_1; M_2)$	$M_1(M_2)$	$\backslash\text{instMd}[S_1][X][S_2]\{M_1\}\{M_2\}$
$\text{Sig}(S_1; X.S_2)$	$X:S_1 \times S_2$	$\backslash\text{sigSg}\{S_1\}\{X\}\{S_2\}$
$\text{str}[S_1; X.S_2](M_1; M_2)$	$\text{str } M_1 \text{ in } M_2$	$\backslash\text{strMd}[S_1][X][S_2]\{M_1\}\{M_2\}$
$\text{proj}\langle i \rangle[S_1; X.S_2](M)$	$M \cdot i$	$\backslash\text{projMd}\langle i \rangle[S_1][X][S_2]\{M\} \quad (i = 1, 2)$

Figure 9: Module Types

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

Figure 10: Processes

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

Figure 11: Polarized Types

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

Figure 12: Parallel Types

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

Figure 13: Propositions and Proofs