Typify types

Oleg Grenrus

July 23, 2015

Parser and tools for type signatures of typify. Essentially the language is dependent type theory, with omitted lambda constructor.

1 Formal syntax definition

level	name		associativity	example
9	recursive	μ	prefix, right associative	$\mu\ a \to \mu\ b \to a\ b \equiv \mu\ a \to (\mu\ b \to (a\ b))$
8	optional	?	postfix	$a?$
7	application	$a\ b$	left associative	$a\ b\ c \equiv (a\ b)\ c$
6	conjunction	$\wedge$	associative	$(a \wedge b) \wedge c \equiv a \wedge (b \wedge c)$
5	disjunction	$\vee$	associative	$(a \vee b) \vee c \equiv a \vee (b \vee c)$
4	ellipsis	...	postfix	$a \ldots$
3	type	:	right associative	$x : a$
2	product	$\times$	associative	$(a \times b) \times c \equiv a \times (b \times c)$
1	function	$\to$	right associative	$a \to b \to c \equiv a \to (b \to c)$
0	semicolon	;	associative	

Table 1: Operator precedence

name		code variants
any	$\top$	$\top$ *
nothing	$\bot$	$\bot$ _|_
unit	$\mathbb{1}$	$\mathbb{1}$ ()
optional	?	?
conjunction	$\wedge$	$\wedge$ |
disjunction	$\vee$	$\vee$ &
ellipsis	...	. . . (either three dots, or unicode ellipsis)
type	:	:
product	$\times$	$\times$,
function	$\to$	$\to$ ->
semicolon	;	;
rec	μ	μ `rec`

Table 2: Special character code representations

Terminals:

$$\frac{}{\top : \mathsf{type}}\ \textsc{Any} \qquad \frac{}{\bot : \mathsf{type}}\ \textsc{Nothing} \qquad \frac{}{\mathbb{1} : \mathsf{type}}\ \textsc{Unit}$$

$$\frac{}{\texttt{number} : \mathsf{type}}\ \textsc{Number} \qquad \frac{}{\texttt{string} : \mathsf{type}}\ \textsc{String} \qquad \frac{}{\texttt{bool} : \mathsf{type}}\ \textsc{Boolean}$$

Identifiers:

$$\frac{n : \mathsf{name}}{n : \mathsf{type}}\ \textsc{Identifier}$$

Record pairs:

$$\frac{n : \mathsf{name} \quad a : \mathsf{type}}{n : a : \mathsf{pair}}\ \textsc{Record-Pair} \qquad \frac{r : \mathsf{pair}}{r : \mathsf{rlist}}\ \textsc{Record-Singleton} \qquad \frac{p : \mathsf{pair} \quad r : \mathsf{rlist}}{p; r : \mathsf{rlist}}\ \textsc{Record-Cons}$$

Records:

$$\frac{}{\{\} : \mathsf{type}}\ \textsc{Empty-Record} \qquad \frac{r : \mathsf{rlist}}{\{r\} : \mathsf{type}}\ \textsc{Record}$$

Row types:

$$\frac{a : \mathsf{type} \quad b : \mathsf{type}}{a \wedge b : \mathsf{type}}\ \textsc{Conjunction} \qquad \frac{a : \mathsf{type} \quad b : \mathsf{type}}{a \vee b : \mathsf{type}}\ \textsc{Disjunction} \qquad \frac{a : \mathsf{type}}{a? : \mathsf{type}}\ \textsc{Optional}$$

Application:

$$\frac{a : \mathsf{type} \qquad b : \mathsf{type}}{a\ b : \mathsf{type}}\ \textsc{Application}$$

Special modifiers:

$$\frac{a : \mathsf{type} \quad b : \mathsf{type}}{a \times b : \mathsf{type}}\ \textsc{Product} \qquad \frac{a : \mathsf{type}}{a \ldots : \mathsf{type}}\ \textsc{Variadic} \qquad \frac{n : \mathsf{name} \quad p : \mathsf{type}}{n : p : \mathsf{type}}\ \textsc{Named}$$

Brackets:

$$\frac{p : \mathsf{type}}{[p] : \mathsf{type}}\ \textsc{Brackets}$$

Functions:

$$\frac{a : \mathsf{type} \quad b : \mathsf{type}}{a \to b : \mathsf{type}}\ \textsc{Function}$$

Recursive types:

$$\frac{n : \mathsf{name} \quad a : \mathsf{type}}{\mu\ n \to a : \mathsf{type}}\ \textsc{Recursive}$$

Figure 1: Typify syntax rules

$$\begin{aligned}
&a \times ys : b \ldots \times c \to d \to e \\
&\quad \equiv x : a \times ys : (b \ldots) \times c \to d \to e \\
&\quad \equiv (x : a) \times (ys : (b \ldots)) \times c \to d \to e \\
&\quad \equiv ((x : a) \times (ys : (b \ldots)) \times c) \to d \to e \\
&\quad \equiv ((x : a) \times (ys : (b \ldots)) \times c) \to (d \to e) \\
&a \times y : b \to c \\
&\quad \equiv a \times (y : b) \to c \\
&\quad \equiv (a \times (y : b)) \to c
\end{aligned}$$

$$\begin{aligned}
&a \vee b? \wedge c\ d \ldots \to e \\
&\quad \equiv a \vee (b?) \wedge (c\ d) \ldots \to e \\
&\quad \equiv a \vee ((b?) \wedge (c\ d)) \ldots \to e \\
&\quad \equiv (a \vee ((b?) \wedge (c\ d))) \ldots \to e \\
&\quad \equiv ((a \vee ((b?) \wedge (c\ d))) \ldots) \to e
\end{aligned}$$

Figure 2: Examples of operator precedence

$$Either\ a\ b \equiv \{type : \texttt{"left"};\ value : a\} \vee \{type : \texttt{"right"};\ value : b\}$$
$$flatMap : (@ : Observable\ A \times f : A \to Observable\ B \vee Event\ B \vee B) \to EventStream\ B$$

Figure 3: Examples of real world types