System F Language Specification

Syntax

Expressions

e	$:=$	l	literals
		v	expression identifier
		(e)	parenthesized
		$e[\tau]$	type concretization
		$e_1\ e_2$	application
		$e_1\ op\ e_2$	binary operation
		$(e_1,\ \ldots,\ e_n)$	n-tuples, $n \geq 2$
		$\boldsymbol{\lambda}\ (v_1{:}t_1)\ \ldots\ (v_n{:}t_n){:}\ \tau_{\mathrm{ret}}.\ e$	lambda abstraction[1]
		$\boldsymbol{\Lambda}\ t_1\ \ldots\ t_n.\ e$	type abstraction
		$\texttt{let}\ p\ \texttt{=}\ e_1\ \texttt{in}\ e_2$	let binding
		$\texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3$	if expression
l	$:=$	$\texttt{null}$	unit literal: $\texttt{Unit}$
		$\texttt{true} \mid \texttt{false}$	boolean literals: $\texttt{Bool}$
		$\ldots \mid \sim\!2 \mid \sim\!1 \mid 0 \mid 1 \mid 2 \mid \ldots$	64-bit signed ints: $\texttt{Int}$
p	$:=$	$\_\ :\ \tau$	discarded pattern
		$v\ :\ \tau$	single argument
		$(p_1,\ \ldots,\ p_n)$	n-tuple destructor, $n \geq 2$

Types

τ	$:=$	t	type identifier
		(τ)	parenthesized
		$\tau_1\ \texttt{->}\ \tau_2$	arrow types
		$\boldsymbol{\forall}\ t.\ \tau$	universal types
		$\tau_1\ \texttt{*}\ \ldots\ \texttt{*}\ \tau_n$	tuple types, $n \geq 2$
		$\texttt{Int} \mid \texttt{Bool} \mid \texttt{Unit}$	built-in types

Declarations

δ	$:=$	$\texttt{let}\ p\ \texttt{=}\ e$	declaration

---

[1]Pattern must be enclosed in parentheses if not already

Alternative syntax

We can write \ or **lambda** instead of **λ**.
We can write **any** in place of **Λ**.
We can write **forall** in place of **∀**.

Semantics:

Call-by-value big step semantics.
When a bound variable is bound again, the new binding takes over.
There is no one-type tuples
Lexical scope.