

System F Language Specification

Syntax

Expressions

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

Types

τ	$:=$	t (τ) $\tau_1 \rightarrow \tau_2$ $\forall t. \tau$ $\tau_1 * \dots * \tau_n$ Int Bool Unit	type identifier parenthesized arrow types universal types tuple types, $n \geq 2$ built-in types
--------	------	--	---

Declarations

δ	$:=$	let $p = e$	declaration
----------	------	--------------------	-------------

¹One can also denote a curried multi-argument function with the syntax $\lambda (v_1:t_1) \dots (v_n:t_n). e$, which desugars to n nested lambda expressions. Note that in this case, parentheses are needed around each annotated argument.

²Similarly, $\Lambda t_1 \dots t_n. e$ is n nested type abstractions.

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 \forall .

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.