

$t$	$::=$	terms	$S$	$::=$	tags
$b$		boolean value	Int		integer tag
$n$		numeric value	Bool		boolean tag
$op$		operator	Fun		function tag
$\lambda x. t$		abstraction			
$x$		variable	$v$	$::=$	configuration – values
$t t$		application	$b$		boolean value
$\text{mlet } x = t \text{ in } t$		overloading let	$n$		numeric value
$t :: T$		ascription	$op$		operator
			$(\lambda x. t)[s]$		closure
$b$	$::=$	boolean value	$c$	$::=$	configurations
true		true value	$v$		
false		false value	$t[s]$		
$op$	$::=$	operators	$c c$		
add1		sum	$\text{mlet } x = c \text{ in } c$		
not		negation	$c :: T$		
$T$	$::=$	types	error		
Int		type of integers	$s$	$::=$	explicit substitutions
Bool		type of booleans	$\bullet$		empty substitution
$T \rightarrow T$		type of functions	$x \mapsto \{(\overline{v : S})\}, s$		variable substitution

Figure 1: Syntax of the simply typed lambda-calculus with overloading.

- Deterministic.
- Type error detection.
- Dispatch error detection. In the case of the lambda functions, it is not effective because if the environment contains at least a value with tag function, it is not detected dispatch error, because we don't have type information for lambdas function. Similar with mlet, expression.
- Ambiguity error detection(restrictive).
- Without type annotation in lambda functions or mlet, only in ascription.
- Semantic "tag driven", introducing flat tag in the environment.

**Definition 1** ( $\oplus$ ). *Given an environment  $s$  and a variable binding  $x \mapsto (v_1 : S_1)$ , the operator  $\oplus$  is defined as follows:*

$$s \oplus x \mapsto (v_1 : S_1) = \begin{cases} x \mapsto \{(v_1 : S_1)\} & s = \emptyset \\ x \mapsto \{(\overline{v : S})\} \cup \{(v_1 : S_1)\}, s' & s = x \mapsto \{(\overline{v : S})\}, s' \\ y \mapsto \{(\overline{v : S})\}, s' \oplus x \mapsto (v_1 : S_1) & s = y \mapsto \{(\overline{v : S})\}, s' \end{cases}$$

	$c \longrightarrow c$
$b[s] \longrightarrow b$	(False)
$n[s] \longrightarrow n$	(Num)
$op[s] \longrightarrow op$	(Op)
$x[\ ] \longrightarrow \text{error}$	(ErrVarFail)
$x[x \mapsto \{(v : S_1)\}, s'] \longrightarrow v_i$	(VarOk)
$\frac{x \neq y}{x[y \mapsto \{(v : S_1)\}, s] \longrightarrow x[s]}$	(VarNext)
$v :: T \longrightarrow v$	(Asc)
$\frac{S_1 = \text{tagVal}(v)}{\text{mlet } x = v \text{ in } t_2[s] \longrightarrow t_2[x \mapsto (v : S_1) \oplus s]}$	(Let)
$(\lambda x. t_2)[s] v \longrightarrow ([x \mapsto v]t_2)[s]$	(App)
$\text{add1 } n \longrightarrow n + 1$	(Sum)
$\text{not } b \longrightarrow \neg b$	(Negation)

Figure 2: Configuration reduction rules.

	$c \longrightarrow c$
$\frac{S = \text{tagType}(T) \quad v = \text{lookup}(x, [s], S)}{x[s] :: T \longrightarrow v :: T}$	(AscVar)
$\frac{v_1 = \text{lookup}(x_1, [s_1], \text{Fun})}{x_1[s_1] \ v_2 \longrightarrow v_1 \ v_2}$	(AppVar)
$\frac{n = \text{lookup}(x, [s], \text{Int})}{\text{add1 } x[s] \longrightarrow \text{add1 } n}$	(SumVar)
$\frac{b = \text{lookup}(x, [s], \text{Bool})}{\text{not } x[s] \longrightarrow \text{not } b}$	(NegationVar)
$\frac{c \longrightarrow c' \quad \text{notVal\_Var}(c)}{c :: T \longrightarrow c' :: T}$	(Asc1)
$\frac{c_1 \longrightarrow c'_1 \quad \text{notVal}(c_1)}{\text{mlet } x = c_1 \text{ in } c_2 \longrightarrow \text{mlet } x = c'_1 \text{ in } c_2}$	(Let1)
$\frac{c_1 \longrightarrow c'_1 \quad \text{notVal\_Var}(c_1)}{c_1 \ c_2 \longrightarrow c'_1 \ c_2}$	(App1)
$\frac{c \longrightarrow c' \quad \text{notVal}(c)}{(\lambda x. t_2)[s] \ c \longrightarrow (\lambda x. t_2)[s] \ c'}$	(App2)
$\frac{c_2 \longrightarrow c'_2 \quad \text{notVal}(c_2)}{x[s] \ c_2 \longrightarrow x[s] \ c'_2}$	(App3)
$\frac{c \longrightarrow c' \quad \text{notVal\_Var}(c)}{op \ c \longrightarrow op \ c'}$	(App4)

Figure 3: Configuration reduction rules.

**Definition 2** (flat). *The function flat is defined as follows:*

$$\text{flat}(s) = \begin{cases} \emptyset & s = \emptyset \\ x \mapsto (v_1 : T_1) \cdots, x \mapsto (v_n : T_n), \text{flat}(s') & s = x \mapsto \{\overline{(v : T)}\}, s' \end{cases}$$

**Definition 3** (lookup). *The function lookup is defined as follows:*

$$\text{lookup}(x, s, S') = \begin{cases} v_i & !\exists(x \mapsto (v_i : S)) \in \text{flat}(s) \\ \text{lookup}(x, s', S') & s = y \mapsto \{\overline{(v : S)}\}, s' \\ \text{error} & s = \emptyset \end{cases}$$

**Definition 4** (tagType). *The function tagType is defined as follows:*

$$\text{tagType}(T) = \begin{cases} \text{Int} & T = \text{Int} \\ \text{Bool} & T = \text{Bool} \\ \text{Fun} & T = T_1 \rightarrow T_2 \end{cases}$$

**Definition 5** (tagVal). *The function tagVal is defined as follows:*

$$\text{tagVal}(v) = \begin{cases} \text{Int} & v = n \\ \text{Bool} & v = b \\ \text{Fun} & v = \lambda x. t \end{cases}$$