1 Languages

1.1 Flexible Language

Flexible Language:

- Non deterministic.
- Type error means stuck.
- Without type annotation in lambda functions and mlet.
- Introduce the explicit substitution in mlet, and the substitution in lambdas, the difference.
- Characterize errors.
- Fix Index .

t	$::=$		terms
		b	boolean value
		n	numeric value
		op	operator
		x	variable
		$\lambda x.\ t$	abstraction
		$t\ t$	application
		$\mathsf{mlet}\ x = t\ \mathsf{in}\ t$	overloading let
b	$::=$		boolean value
		true	true value
		false	false value
op	$::=$		operators
		add1	sum
		not	negation

v	$::=$		values
		b	boolean value
		n	numeric value
		op	operator
		$(\lambda x.\ t)[s]$	closure
c	$::=$		configurations
		v	
		$t[s]$	
		$c\ c$	
		$\mathsf{mlet}\ x = c\ \mathsf{in}\ c$	
s	$::=$		explicit substitutions
		$\bullet$	empty substitution
		$x \mapsto \{\overline{v}\}, s$	variable substitution

Figure 1: Syntax of the Flexible Language.

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

$$s \oplus x \mapsto v_1 = \begin{cases} x \mapsto \{v_1\} & s = \varnothing \\ x \mapsto \{\overline{v}\} \cup \{v_1\}, s' & s = x \mapsto \{\overline{v}\}, s' \\ y \mapsto \{\overline{v}\}, s' \oplus x \mapsto v_1 & s = y \mapsto \{\overline{v}\}, s' \end{cases}$$

$$\boxed{c \longrightarrow c}$$

$$b[s] \longrightarrow b \qquad \textsf{(False)}$$

$$n[s] \longrightarrow n \qquad \textsf{(Num)}$$

$$op[s] \longrightarrow op \qquad \textsf{(Op)}$$

$$x[x \mapsto \{\overline{v}\}, s] \longrightarrow v_i \qquad \textsf{(VarOk)}$$

$$\frac{x \neq y}{x[y \mapsto \{\overline{v}\}, s] \longrightarrow x[s]} \qquad \textsf{(VarNext)}$$

$$(\textsf{mlet } x = t_1 \textsf{ in } t_2)[s] \longrightarrow \textsf{mlet } x = t_1[s] \textsf{ in } t_2[s] \qquad \textsf{(LetSub)}$$

$$(t_1\ t_2)[s] \longrightarrow t_1[s]\ t_2[s] \qquad \textsf{(AppSub)}$$

$$\textsf{mlet } x = v \textsf{ in } t_2[s] \longrightarrow t_2[x \mapsto v \oplus s] \qquad \textsf{(Let)}$$

$$(\lambda x.\ t_2)[s]\ v \longrightarrow ([x \mapsto v]t_2)[s] \qquad \textsf{(App)}$$

$$\textsf{add1 } n \longrightarrow n + 1 \qquad \textsf{(Sum)}$$

$$\textsf{not } b \longrightarrow \neg\, b \qquad \textsf{(Negation)}$$

$$\frac{c_1 \longrightarrow c_1'}{\textsf{mlet } x = c_1 \textsf{ in } c_2 \longrightarrow \textsf{mlet } x = c_1' \textsf{ in } c_2} \qquad \textsf{(Let1)}$$

$$\frac{c_1 \longrightarrow c_1'}{c_1\ c_2 \longrightarrow c_1'\ c_2} \qquad \textsf{(App1)}$$

$$\frac{c \longrightarrow c'}{v\ c \longrightarrow v\ c'} \qquad \textsf{(App2)}$$

Figure 2: Reduction rules for Flexible Language.

1.2 Tag Driven Language

Tag Driven Language:

- Non deterministic.
- Type error means stuck.
- Dispatch error means stuck.
- Without type annotation in lambda functions and mlet.
- Semantic "tag driven", introducing flat tag in the environment.

Definition 2 (lookup)**.** *The relation* lookup *is defined as follows:*

$$\textsf{lookup} = \{(x, s, S', v) \mid x \mapsto v \in \textsf{flat}(s) \wedge \textsf{tag}(v) = S\}$$

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

$$\textsf{flat}(s) = \begin{cases} \varnothing & s = \varnothing \\ x \mapsto v_1 \cdots, x \mapsto v_n, \textsf{flat}(s') & s = x \mapsto \{\overline{v}\}, s' \end{cases}$$

		...	
S	::=		tags
		Int	integer tag
		Bool	boolean tag
		Fun	function tag
		...	

Figure 3: Syntax of the Tag Driven Language(Extends Flexible Language).

$$\boxed{c \longrightarrow c}$$

...

$$\frac{v_1 = \mathsf{lookup}(x_1, [s_1], \mathsf{Fun})}{x_1[s_1]\ v_2 \longrightarrow v_1\ v_2} \quad (\mathsf{AppVar})$$

$$\frac{n = \mathsf{lookup}(x, [s], \mathsf{Int})}{\mathsf{add1}\ x[s] \longrightarrow \mathsf{add1}\ n} \quad (\mathsf{SumVar})$$

$$\frac{b = \mathsf{lookup}(x, [s], \mathsf{Bool})}{\mathsf{not}\ x[s] \longrightarrow \mathsf{not}\ \ b} \quad (\mathsf{NegationVar})$$

$$\frac{c_1 \longrightarrow c_1' \qquad \mathsf{notVal}(c_1)}{\mathsf{mlet}\ x = c_1\ \mathsf{in}\ c_2 \longrightarrow \mathsf{mlet}\ x = c_1'\ \mathsf{in}\ c_2} \quad (\mathsf{Let1})$$

$$\frac{c_1 \longrightarrow c_1' \qquad \mathsf{notVal\_Var}(c_1)}{c_1\ c_2 \longrightarrow c_1'\ c_2} \quad (\mathsf{App1})$$

$$\frac{c \longrightarrow c' \qquad \mathsf{notVal}(c)}{(\lambda x.\ t_2)[s]\ c \longrightarrow (\lambda x.\ t_2)[s]\ c'} \quad (\mathsf{App2})$$

$$\frac{c_2 \longrightarrow c_2' \qquad \mathsf{notVal}(c_2)}{x[s]\ c_2 \longrightarrow x[s]\ c_2'} \quad (\mathsf{App3})$$

$$\frac{c \longrightarrow c' \qquad \mathsf{notVal\_Var}(c)}{op\ c \longrightarrow op\ c'} \quad (\mathsf{App4})$$

...

Figure 4: Reduction rules for Tag Driven Language(Extends Flexible Language).

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

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

1.3 Tag Driven Language with ascription

t	$::=$		terms
		$\ldots$	
		$t :: S$	ascription
c	$::=$		configurations
		$\ldots$	
		$c :: S$	

Figure 5: Syntax of the Tag Driven Languagewith ascriptions.

$$\boxed{c \longrightarrow c}$$

$$\ldots$$

$$v :: S \longrightarrow v \qquad \text{(Asc)}$$

$$\frac{v = \mathsf{lookup}(x, [s], S)}{x[s] :: S \longrightarrow v :: T} \qquad \text{(AscVar)}$$

$$\frac{c \longrightarrow c' \qquad \mathsf{notVal\_Var}(c)}{c :: S \longrightarrow c' :: S} \qquad \text{(Asc1)}$$

Figure 6: Reduction rules for Tag Driven Language with ascriptions.

1.4 Strict Language

		$\ldots$	
w	$::=$		multi − value
		$\{\overline{v}\}$	set of values

Figure 7: Syntax of the Strict Language(Extends Tag Driven Language with ascriptions).

1.5 Overloading Language

$$\boxed{c \longrightarrow c}$$

$$w[s] \longrightarrow w \qquad \text{(MultiValue)}$$

$$x[x \mapsto w, s] \longrightarrow w \qquad \text{(VarOk)}$$

$$\frac{x \neq y}{x[y \mapsto w, s] \longrightarrow x[s]} \qquad \text{(VarNext)}$$

$$(\mathsf{mlet}\ x = t_1\ \mathsf{in}\ t_2)[s] \longrightarrow \mathsf{mlet}\ x = t_1[s]\ \mathsf{in}\ t_2[s] \qquad \text{(LetSub)}$$

$$(t_1\ t_2)[s] \longrightarrow t_1[s]\ t_2[s] \qquad \text{(AppSub)}$$

$$\mathsf{mlet}\ x = v\ \mathsf{in}\ t_2[s] \longrightarrow t_2[x \mapsto v \oplus s] \qquad \text{(Let)}$$

$$(\lambda x.\ t_2)[s]\ v \longrightarrow ([x \mapsto v]t_2)[s] \qquad \text{(App)}$$

$$\mathsf{add1}\ n \longrightarrow n + 1 \qquad \text{(Sum)}$$

$$\mathsf{not}\ b \longrightarrow \neg\ b \qquad \text{(Negation)}$$

$$\frac{c_1 \longrightarrow c_1'}{\mathsf{mlet}\ x = c_1\ \mathsf{in}\ c_2 \longrightarrow \mathsf{mlet}\ x = c_1'\ \mathsf{in}\ c_2} \qquad \text{(Let1)}$$

$$\frac{c_1 \longrightarrow c_1'}{c_1\ c_2 \longrightarrow c_1'\ c_2} \qquad \text{(App1)}$$

$$\frac{c \longrightarrow c'}{v\ c \longrightarrow v\ c'} \qquad \text{(App2)}$$

Figure 8: Reduction rules for Strict Language.