1 Languages

1.1 Flexible Language

Flexible Language:

- Non deterministic semantic, with branches that reduce to a value and other ending in error.
- Type error means stuck.
- Without any type information in the syntax of the language.
- Uses the explicit substitution in mlet, and the implicit substitution in lambdas. In the case of the mlet is used the explicit substitution because the implicit substitution of a variable by a value would eliminate the overloading.

Characterization of the errors for Flexible Language:

- Free variable error is detected if a variable is evaluated in the empty environment.
- Type error is detected if the operators not or $\mathsf{add1}$ are applied with parameters that are not of type Bool or Int, respectively. Also, if the left side of the function application is not a lambda.

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
		$\mathsf{add1}$	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.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Figure 2: Reduction rules for Flexible Language.

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

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

1.2 Tag Driven Language

Tag Driven Language:

- Non deterministic semantic, with branches that reduce to a value and other ending in error.
- Type error means stuck.
- Dispatch error means stuck.
- Without any type information in the syntax of the language.

		$\ldots$	
S	::=		tags
		Int	integer tag
		Bool	boolean tag
		Fun	function tag
		$\ldots$	

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

- Semantic "tag driven", introducing flat tag.

Characterization of the errors for Tag Driven Language:

- Free variable error is detected if a variable is evaluated in the empty environment.
- Type error is detected if the operators not or $\mathsf{add1}$ are applied with parameters that are not of type Bool or Int, respectively. Also, if the left side of the function application is not a lambda.

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

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

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

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

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

1.4 Strict Language

Definition 5 ($\mathsf{filter}(\cdot, \cdot)$). *The function* filter *is defined as follows:*

$$\mathsf{filter}(w, S) = \{v \in w \mid \mathsf{tag}(v) = S\}$$

Definition 6 (lookup). *The function* lookup *is defined as follows:*
$\mathsf{lookup}(w_1, w_2) = (v_1, v_2)\ !\exists v_1 \in w_1 \mid \mathsf{tag}(v_1) = \mathsf{Fun}\ \wedge\ w_2 = \{v_2\}$

1.5 Overloading Language

- Deterministic.
- Type error detection.
- Dispatch error detection.
- Ambiguity error detection.
- Type annotation in lambda functions, mlet and ascription.

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

$$\ldots$$

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

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

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

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

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

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

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

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

$$\ldots$$

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

$$
\begin{array}{|llll|}
\hline
t & ::= & & \textsf{terms} \\
 & & \ldots & \\
 & & t :: S & \textsf{ascription} \\
 & & & \\
c & ::= & & \textsf{configurations} \\
 & & \ldots & \\
 & & c :: S & \\
\hline
\end{array}
$$

Figure 5: Syntax of the Tag Driven Languagewith ascriptions.

$$
\begin{array}{|cr|}
\hline
 & \boxed{c \longrightarrow c} \\
\ldots & \\
(t :: S)[s] \longrightarrow t[s] :: S & (\textsf{AscSub}) \\
\dfrac{\textsf{tag}(v) = S}{v :: S \longrightarrow v} & (\textsf{Asc}) \\
\dfrac{v = \textsf{lookup}(x, [s], S)}{x[s] :: S \longrightarrow v :: T} & (\textsf{AscVar}) \\
\dfrac{c \longrightarrow c' \qquad \textsf{notVal\_Var}(c)}{c :: S \longrightarrow c' :: S} & (\textsf{Asc1}) \\
\hline
\end{array}
$$

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

- More expressive than Semantic 3, with the use structural tags.
- Do not support context-dependent overloading.

Definition 7 ($\oplus$). *Given an environment s and a variable binding $x \mapsto (v_1 : T_1)$, the operator $\oplus$ is defined as follows:*

$$
s \oplus x \mapsto (v_1 : T_1) = \begin{cases} x \mapsto \{(v_1 : T_1)\} & s = \varnothing \\ x \mapsto \{\overline{(v : T)}\} \cup \{(v_1 : T_1)\}, s' & s = x \mapsto \{\overline{(v : T)}\}, s' \\ y \mapsto \{\overline{(v : T)}\}, s' \oplus x \mapsto (v_1 : T_1) & s = y \mapsto \{\overline{(v : S)}\}, s' \end{cases}
$$

Definition 8 (flat). *The function* flat *is defined as follows:*

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

Definition 9 (lookup). *The function* lookup *is defined as follows:*

$$
\textsf{lookup}(x, s, S') = \begin{cases} v_i & s = x \mapsto \{\overline{(v : S)}\}, s' \wedge S' = S_i \\ \textsf{lookup}(x, s', S') & s = y \mapsto \{\overline{(v : S)}\}, s' \\ \textsf{error} & s = \varnothing \end{cases}
$$

$$
\textsf{lookup}(x_1, x_2, s_1, s_2) = \begin{cases} (v_1, v_2) & !\exists x_1 \mapsto (v_1 : T_1) \in \textsf{flat}(s_1) \wedge !\exists x_2 \mapsto (v_2 : T_1) \in \textsf{flat}(s_2) \\ \textsf{error} & otrw \end{cases}
$$

$$
\begin{array}{lll r}
 & & \ldots & \\
w & ::= & & \textsf{multi} - \textsf{value} \\
 & & \{\overline{v}\} & \textsf{set of values} \\
 & & & \\
c & ::= & & \textsf{configurations} \\
 & & w & \\
 & & t[s] & \\
 & & c\ c & \\
 & & \textsf{mlet}\ x = c\ \textsf{in}\ c & \\
 & & c :: S &
\end{array}
$$

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

Definition 10 (tag)**.** *The function* tag *is defined as follows:*

$$
\textsf{tag}(v) = \begin{cases} \textsf{Int} & v = n \\ \textsf{Bool} & v = b \\ T_1 \to T_2 & v = \lambda x : T_1.\ t_2 \end{cases}
$$

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

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

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

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

$$(t :: S)[s] \longrightarrow t[s] :: S \qquad \text{(AscSub)}$$

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

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

$$\frac{\mathsf{filter}(v, S) = w' \qquad w' \neq \emptyset}{w :: S \longrightarrow w'} \qquad \text{(Asc)}$$

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

$$\frac{((\lambda x.\ t_2)[s], v_2) = \mathsf{lookup}(w_1, w_2)}{w_1\ w_2 \longrightarrow ([x \mapsto v_2]t_2)[s]} \qquad \text{(App)}$$

$$\frac{\mathsf{filter}(w, \mathsf{Int}) = \{\overline{n}\}}{\mathsf{add1}\ w \longrightarrow \{\overline{n+1}\}} \qquad \text{(Sum)}$$

$$\frac{\mathsf{filter}(w, \mathsf{Bool}) = \{\overline{b}\}}{\mathsf{not}\ w \longrightarrow \{\overline{\neg\ b}\}} \qquad \text{(Negation)}$$

$$\frac{c \longrightarrow c' \qquad \mathsf{notVal\_Var}(c)}{c :: T \longrightarrow c' :: T} \qquad \text{(Asc1)}$$

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

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

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

Figure 8: Reduction rules for Strict Language.

t	$::=$		terms
		b	boolean value
		n	numeric value
		op	operator
		$\lambda x.\ t$	abstraction
		x	variable
		$t\ t$	application
		$\mathsf{mlet}\ x = t\ \mathsf{in}\ t$	overloading let
		$t :: T$	ascription
b	$::=$		boolean value
		true	true value
		false	false value
op	$::=$		operators
		add1	sum
		not	negation
T	$::=$		types
		Int	type of integers
		Bool	type of booleans
		$T \to T$	type of functions

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

Figure 9: Syntax of the Overloading Language.

$\boxed{c \longrightarrow c}$

$b[s] \longrightarrow b$	(Bool)
$n[s] \longrightarrow n$	(Num)
$op[s] \longrightarrow op$	(Op)
$(t :: T)[s] \longrightarrow t[s] :: T$	(AscSub)
$(\mathsf{mlet}\ x = t_1\ \mathsf{in}\ t_2)[s] \longrightarrow \mathsf{mlet}\ x = t_1[s]\ \mathsf{in}\ t_2[s]$	(LetSub)
$(t_1\ t_2)[s] \longrightarrow t_1[s]\ t_2[s]$	(AppSub)
$v :: T \longrightarrow v$	(Asc)
$\mathsf{mlet}\ x = v\ \mathsf{in}\ t_2[s] \longrightarrow t_2[x \mapsto v \oplus s]$	(Let)
$(\lambda x : T_1.\ t_2)[s]\ v \longrightarrow ([x \mapsto v]t_2)[s]$	(App)
$\mathsf{add1}\ n \longrightarrow n + 1$	(Sum)
$\mathsf{not}\ b \longrightarrow \neg\ b$	(Negation)

Figure 10: Configuration reduction rules.

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

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

$$\frac{v = \mathsf{lookup}(x_1, [s_1], T_1)}{\mathsf{mlet}\ x = x_1[s_1]\ \mathsf{in}\ c_2 \longrightarrow \mathsf{mlet}\ x = v\ \mathsf{in}\ c_2} \quad (\mathsf{LetVar})$$

$$\frac{v_2 = \mathsf{lookup}(x_2, [s_2], T_1)}{(\lambda x : T_1.\ t_2)[s]\ x_2[s_2] \longrightarrow (\lambda x : T_1.\ t_2)[s]\ v_2} \quad (\mathsf{AppVar1})$$

$$\frac{T = \mathsf{tag}(v_2) \qquad v_1 = \mathsf{lookup}(x_1, [s_1], T \to *)}{x_1[s_1]\ v_2 \longrightarrow v_1\ v_2} \quad (\mathsf{AppVar2})$$

$$\frac{(v_1, v_2) = \mathsf{lookup}(x_1, x_2, [s_1], [s_2])}{x_1[s_1]\ x_2[s_2] \longrightarrow v_1\ v_2} \quad (\mathsf{AppVar3})$$

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

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

$$\frac{c \longrightarrow c' \qquad \mathsf{notVal\_Var}(c)}{c :: T \longrightarrow c' :: T} \quad (\mathsf{Asc1})$$

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

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

$$\frac{c \longrightarrow c' \qquad \mathsf{notVal\_Var}(c)}{v\ c \longrightarrow v\ c'} \quad (\mathsf{App2})$$

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

Figure 11: Configuration reduction rules.