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 add1 are applied with parameters that are not boolean or numeric value, 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
		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.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Figure 2: Reduction rules for Flexible Language.

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.

$$
\begin{array}{lll}
 & & \ldots \\
S & ::= & & \text{tags} \\
 & & \mathsf{Int} & \text{integer tag} \\
 & & \mathsf{Bool} & \text{boolean tag} \\
 & & \mathsf{Fun} & \text{function tag} \\
 & & \ldots
\end{array}
$$

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 and type error are detected in the same cases that the Flexible Language.
- Dispatch error is detected if the operators not or add1 are applied with overloaded parameters that do not have instance with tag Bool or Int in the environment, respectively. Also, if the left side of the function application is an overloaded variable, but does not exist an instance with tag Fun in the environment.

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

$$\ldots$$

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

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

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

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

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

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

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

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

$$\ldots$$

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

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

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

Figure 5: Syntax of the Tag Driven Language with ascriptions.

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

$$\ldots$$

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

$$\frac{\mathsf{tag}(v) = S}{v :: S \longrightarrow v} \qquad (\mathsf{Asc})$$

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

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

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

1.4 Strict Language

Strict Language:

- Deterministic semantic. With the use of multi-values a program can reduce to a set of value.
- Type error means stuck.
- Dispatch error means stuck.
- Ambiguity error means stuck.

Characterization of the errors for Strict Language:

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

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

- Free variable and type error are detected in the same cases that the Tag Driven Language with ascription.
- Ambiguity error is detected if the left side of the function application have more than one instance with Fun tag or in the right side have more than one value for the application. Strict detection of ambiguity.

Definition 5 ($\oplus$). *Given an environment s and a variable binding $x \mapsto v_1$, the operator $\oplus$ is defined as follows:*

$$
s \oplus x \mapsto w = \begin{cases} x \mapsto \{w\} & s = \varnothing \wedge w = v \\ x \mapsto w & s = \varnothing \wedge w \neq v \\ x \mapsto w' \cup \{w\}, s' & s = x \mapsto w', s' \wedge w = v \\ x \mapsto w' \cup w, s' & s = x \mapsto w', s' \wedge w \neq v \\ y \mapsto w', s' \oplus x \mapsto w & s = y \mapsto w', s' \end{cases}
$$

Definition 6 ($\mathsf{filter}(\cdot,\cdot)$). *The function* filter *is defined as follows:*

$$
\mathsf{filter}(w, S) = \begin{cases} \{w\} & w = v \wedge \mathsf{tag}(w) = S \\ \{\overline{v}\} & v_i \in w \wedge \mathsf{tag}(v_i) = S \end{cases}
$$

Definition 7 (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 semantic. With the use of multi-values a program can reduce to a set of value.
- Type error detection.
- Dispatch error detection.
- Ambiguity error detection.
- Type annotation in lambda functions, mlet and ascription.
- More expressive than Strict Language, with the use structural tags.
- Do not support context-dependent overloading.

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

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

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

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

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

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

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

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

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

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

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

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

$$\frac{c \longrightarrow c'}{c :: S \longrightarrow c' :: S} \quad \text{(Asc1)}$$

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

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

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

Figure 8: Reduction rules for Strict Language.

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

Figure 9: Syntax of the Overloading Language.

- Como no esta verificacda la informacion de tipo, no se puede decir nada acerca de la semantica.

Definition 8 ($\oplus$). *Given an environment s and a variable binding $x \mapsto v_1$, the operator $\oplus$ is defined as follows:*

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

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

$$\mathsf{tag}(v) = \begin{cases} \mathsf{Int} & v = n \\ \mathsf{Bool} & v = b \\ T_1 \to T_2 & v = ((\lambda x.\ t_2)^{T_1 \to T_2})[s] \end{cases}$$

Definition 10 ($\mathsf{filter}(\cdot, \cdot)$). *The function* filter *is defined as follows:*

$$\mathsf{filter}(w, T) = \begin{cases} w & w = v \\ v' & !\exists v' \in w \mid \mathsf{tag}(v') = T \end{cases}$$

Definition 11 (lookup). *The function* lookup *is defined as follows:*

$$\mathsf{lookup}(w_1, w_2) = \begin{cases} (w_1, w_2) & w_1 = v_1 \wedge w_2 = v_2 \\ (w_1, v_2) & w_1 = v_1 \wedge \mathsf{tag}(w_1) = T_1 \to T_2\ \wedge\ !\exists v_2 \in w_2 \mid \mathsf{tag}(v_2) = T_1 \\ (v_1, w_2) & w_2 = v_2 \wedge \mathsf{tag}(w_2) = T_1\ \wedge\ !\exists v_1 \in w_1 \mid \mathsf{dom}(\mathsf{tag}(v_1)) = T_1 \\ (v_1, v_2) & !\exists v_1 \in w_1\ \wedge\ !\exists v_2 \in w_2 \mid \mathsf{tag}(v_1) = T_1 \to T_2 \wedge \mathsf{tag}(v_2) = T_1 \end{cases}$$

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

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

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

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

s

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

$$(\mathsf{mlet}\ x : T_1 = t_1\ \mathsf{in}\ t_2)[s] \longrightarrow \mathsf{mlet}\ x : T_1 = t_1[s]\ \mathsf{in}\ t_2[s] \qquad \text{(LetSub)}$$

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

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

$$\frac{\mathsf{filter}(w, T_1) = v}{\mathsf{mlet}\ x : T_1 = w\ \mathsf{in}\ t_2[s] \longrightarrow t_2[x \mapsto v \oplus s]} \qquad \text{(Let)}$$

$$\frac{(((\lambda x.\ t_2)^{T_1 \to T_2})[s], v_2) = \mathsf{lookup}(w_1, w_2)}{w_1\ w_2 \longrightarrow ([x \mapsto v_2]t_2)[s]} \qquad \text{(App)}$$

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

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

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

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

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

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

Figure 10: Reduction rules for Overloading Language.

$$
\begin{array}{lllr}
 & \ldots & & \\
T^* & ::= & & \text{multi} - \text{types} \\
 & & \{\overline{T}\} & \text{multi} - \text{type} \\
 & & & \\
\Gamma & ::= & & \text{typing contexts} \\
 & & \varnothing & \text{empty context} \\
 & & \Gamma, x : T & \text{term variable binding} \\
 & & & \\
\phi & ::= & & \text{multi} - \text{typing contexts} \\
 & & \varnothing & \text{empty context} \\
 & & \phi, x : T^* & \text{term variable binding}
\end{array}
$$

Figure 11: Syntax for Overloading Language with static semantic.

$$\boxed{\Gamma; \phi \vdash t : T}$$

$$\Gamma; \phi \vdash b : \{\mathsf{Bool}\} \qquad (\mathsf{TBool})$$

$$\Gamma; \phi \vdash n : \{\mathsf{Int}\} \qquad (\mathsf{TInt})$$

$$\Gamma; \phi \vdash \mathsf{not} \; : \{\mathsf{Bool} \to \mathsf{Bool}\} \qquad (\mathsf{TNegation})$$

$$\Gamma; \phi \vdash \mathsf{add1} \; : \{\mathsf{Int} \to \mathsf{Int}\} \qquad (\mathsf{TSum})$$

$$\frac{x : T \in \Gamma}{\Gamma; \phi \vdash x : \{T\}} \qquad (\mathsf{TVar}\Gamma)$$

$$\frac{x : T^* \in \phi}{\Gamma; \phi \vdash x : T^*} \qquad (\mathsf{TVar}\phi)$$

$$\frac{\begin{array}{c} x \notin dom(\Gamma \cup \phi) \\ \Gamma, x : T_1; \phi \vdash t_2 : T_2^* \qquad T_2 \in T_2^* \end{array}}{\Gamma \vdash_c (\lambda x.\ t_2)^{T_1 \to T_2} : \{T_1 \to T_2\}} \qquad (\mathsf{TAbs})$$

$$\frac{\Gamma; \phi \vdash t : T^* \qquad T \in T^*}{\Gamma; \phi \vdash t :: T : \{T\}} \qquad (\mathsf{TAsc})$$

$$\frac{\begin{array}{c} x \notin dom(\Gamma) \qquad \Gamma; \phi \vdash t_1 : T_1^* \\ T_1 \in T_1^* \qquad \Gamma; \phi \oplus (x : T_1) \vdash t_2 : T_2^* \end{array}}{\Gamma; \phi \vdash \mathsf{mlet}\ x : T_1 = t_1 \ \mathsf{in}\ t_2 : T_2^*} \qquad (\mathsf{TLet})$$

$$\frac{\begin{array}{c} \Gamma; \phi \vdash t_1 : T_1^* \qquad \Gamma; \phi \vdash t_2 : T_2^* \\ !\exists (T_1 \to T_2) \in T_1^* \; | !\exists (T_1) \in T_2^* \end{array}}{\Gamma; \phi \vdash t_1 \ t_2 : \{T_2\}} \qquad (\mathsf{TApp})$$

Figure 12: Term typing rules.

$$\boxed{\Gamma;\phi \vdash t : T}$$

$$\Gamma \vdash_c b : \{\mathsf{Bool}\} \qquad (\mathsf{CTBool})$$

$$\Gamma \vdash_c b[s] : \{\mathsf{Bool}\} \qquad (\mathsf{CSTBool})$$

$$\Gamma \vdash_c n : \{\mathsf{Int}\} \qquad (\mathsf{CTInt})$$

$$\Gamma \vdash_c n[s] : \{\mathsf{Int}\} \qquad (\mathsf{CSTInt})$$

$$\Gamma \vdash_c \mathsf{not} \ : \{\mathsf{Bool} \to \mathsf{Bool}\} \qquad (\mathsf{CTNegation})$$

$$\Gamma \vdash_c \mathsf{not}\ [s] : \{\mathsf{Bool} \to \mathsf{Bool}\} \qquad (\mathsf{CSTNegation})$$

$$\Gamma \vdash_c \mathsf{add1} \ : \{\mathsf{Int} \to \mathsf{Int}\} \qquad (\mathsf{CTSum})$$

$$\Gamma \vdash_c \mathsf{add1}\ [s] : \{\mathsf{Int} \to \mathsf{Int}\} \qquad (\mathsf{CSTSum})$$

$$\frac{x : T \in \Gamma}{\Gamma \vdash_c x[s] : \{T\}} \qquad (\mathsf{CTVar}\Gamma)$$

$$\frac{x : T^* \in \phi(s)}{\Gamma \vdash_c x[s] : T^*} \qquad (\mathsf{CTVar}\phi)$$

$$\frac{\begin{array}{c} x \notin dom(\Gamma \cup \phi(s)) \\ \Gamma, x : T_1 \vdash_c t_2[s] : T_2^* \qquad T_2 \in T_2^* \end{array}}{\Gamma \vdash_c ((\lambda x.\ t_2)^{T_1 \to T_2})[s] : \{T_1 \to T_2\}} \qquad (\mathsf{CTAbs})$$

$$\frac{\Gamma \vdash_c t[s] :: T : T^*}{\Gamma \vdash_c (t :: T)[s] : T^*} \qquad (\mathsf{CTAsc})$$

$$\frac{\Gamma \vdash_c c : T^* \qquad T \in T^*}{\Gamma \vdash_c c :: T : \{T\}} \qquad (\mathsf{CSTAsc})$$

$$\frac{\Gamma \vdash_c \mathsf{mlet}\ x : T_1 = t_1[s]\ \mathsf{in}\ t_2[s] : T^*}{\Gamma \vdash_c (\mathsf{mlet}\ x : T_1 = t_1\ \mathsf{in}\ t_2)[s] : T^*} \qquad (\mathsf{CTLet})$$

$$\frac{\begin{array}{c} x \notin dom(\Gamma) \qquad \Gamma \vdash_c c_1 : T_1^* \\ T_1 \in T_1^* \qquad \Gamma;\phi(s) \oplus (x : T_1) \vdash t_2 : T_2^* \end{array}}{\Gamma \vdash_c \mathsf{mlet}\ x : T_1 = c_1\ \mathsf{in}\ t_2[s] : T_2^*} \qquad (\mathsf{CSTLet})$$

$$\frac{\Gamma \vdash_c t_1[s]\ t_2[s] : T^*}{\Gamma \vdash_c (t_1\ t_2)[s] : T^*} \qquad (\mathsf{CTCApp})$$

$$\frac{\begin{array}{c} \Gamma \vdash_c c_1 : T_1^* \qquad \Gamma \vdash_c c_2 : T_2^* \\ !\exists (T_1 \to T_2) \in T_1^* \ |!\exists (T_1) \in T_2^* \end{array}}{\Gamma \vdash_c c_1\ c_2 : \{T_2\}} \qquad (\mathsf{CSTApp})$$

Figure 13: Configuration typing rules.

1.6 Static semantic for Overloading Language

Definition 12 ($\oplus$)**.** *Given a multi-type context ϕ and a pair $(x : T)$, the operator $\oplus$ is defined as follows:*

$$\phi \oplus (x : T) = \begin{cases} x : \{T\} & \phi = \varnothing \\ \phi', x : (T^* \cup \{T\}) & \phi = \phi', x : T^* \\ \phi' \oplus (x : T), y : T^* & \phi = \phi', y : T^* \end{cases}$$

Definition 13 ($\Gamma(s)$)**.** *The typing context built from a substitution s, writing $\Gamma(s)$, it is defined as follows:*

$$\Gamma(s) = \begin{cases} \varnothing & s = \bullet \\ \Gamma(s'), x : T & s = (x, v) : s' \ \wedge \ \Gamma \vdash_c v : T \end{cases}$$