1 Languages

1.1 $\lambda\backslash$

$\lambda\backslash$:

- Non deterministic semantic, with branches that reduce to a value and other ending in error.
- Name error detection.
- Type error detection.
- Without any type information in the syntax of the language.
- Uses the explicit substitution. 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 $\lambda\backslash$:

- Name 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
		mlet $x = t$ in t	overloading let
v	::=		values
		b	boolean value
		n	numeric value
		op	operator
		$(\lambda x.\ t)[s]$	closure
b	::=		boolean value
		true	true value
		false	false value
op	::=		operators
		add1	sum
		not	negation
c	::=		configurations
		v	
		$t[s]$	
		$c\ c$	
		mlet $x = c$ in c	
		e	
e	::=		errors
		NameError	name error
		TypeError	type error
s	::=		explicit substitutions
		$\bullet$	empty substitution
		$x \mapsto \{\overline{v}\}, s$	variable substitution

Figure 1: Syntax of the $\lambda\backslash$.

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

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

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

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

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

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

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

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

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

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

Figure 2: Reduction rules for $\lambda\backslash$.

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

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

Figure 3: Substitution rules for $\lambda\backslash$.

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 $\lambda\backslash_S$

$\lambda\backslash_S$:

- Non deterministic semantic, with branches that reduce to a value and other ending in error.
- name error detection.
- Type error detection.
- Dispatch error detection.
- Without any type information in the syntax of the language.
- Semantic "tag driven", introducing flat tag.

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

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

$$\frac{c_2 \longrightarrow c_2'}{v_1\ c_2 \longrightarrow v_1\ c_2'} \quad (\mathsf{App2})$$

Figure 4: Congruence rules for $\lambda\backslash$.

$$\mathsf{mlet}\ x = e\ \mathsf{in}\ c_2 \longrightarrow e \quad (\mathsf{LetErr})$$

$$e\ c_2 \longrightarrow e \quad (\mathsf{App1Err})$$

$$v\ e \longrightarrow e \quad (\mathsf{App2Err})$$

Figure 5: Propagation error rules for $\lambda\backslash$.

Characterization of the errors for $\lambda\backslash_S$:

- Name and type error are detected in the same cases that the $\lambda\backslash$.
- Dispatch error is detected if the operators not or $\mathsf{add1}$ 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.

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 operator* 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 operator* tag *is defined as follows:*

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

1.3 $\lambda\backslash_S$ with ascription

1.4 $\lambda\backslash_\omega$

$\lambda\backslash_\omega$:

- Deterministic semantic. With the use of multi-values a program can reduce to a set of value.

$$x[\,] \longrightarrow \mathsf{NameError} \qquad (\mathsf{NameError})$$

$$\frac{v_1 \neq \lambda x.\ t \vee v_1 \neq op}{v_1\ v_2 \longrightarrow \mathsf{TypeError}} \qquad (\mathsf{TypeErrApp})$$

$$\frac{v \neq n}{\mathsf{add1}\ v \longrightarrow \mathsf{TypeError}} \qquad (\mathsf{TypeErrSum})$$

$$\frac{v \neq b}{\mathsf{not}\ v \longrightarrow \mathsf{TypeError}} \qquad (\mathsf{TypeErrNegation})$$

Figure 6: Error rules for $\lambda\backslash$.

S	$::=$		tags
		Int	integer tag
		Bool	boolean tag
		Fun	function tag
e	$::=$		errors
		NameError	name error
		TypeError	type error
		DispatchError	dispatch error
...			

Figure 7: Syntax of the $\lambda\backslash_S$.

- Name error detection.
- Type error detection.
- Dispatch error detection.
- Ambiguity error detection.

Characterization of the errors for $\lambda\backslash_\omega$:

- Free variable and type error are detected in the same cases that the $\lambda\backslash_S$ 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.

The only congruence rule that change is: $\dfrac{c \longrightarrow c'}{w\ c \longrightarrow w\ c'}$ (App2)

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

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

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

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

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

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

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

Figure 8: Reduction rules for $\lambda\backslash_S$.

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

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

$$\frac{c_2 \longrightarrow c_2' \qquad \mathsf{notVal}(c)}{(\lambda x.\ t_2)[s]\ c_2 \longrightarrow (\lambda x.\ t_2)[s]\ c'} \quad (\mathsf{App2})$$

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

$$\frac{c_2 \longrightarrow c_2' \qquad \mathsf{notVal\_Var}(c_2)}{op\ c_2 \longrightarrow op\ c_2'} \quad (\mathsf{App4})$$

Figure 9: Congruence rules for $\lambda\backslash_S$.

1.5 $\lambda\backslash_T$

- 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 $\lambda\backslash_\omega$, with the use structural tags.
- Do not support context-dependent overloading.
- Since it is not verified the type information, the evaluation it is ...

Definition 7 (tag). *The operator* tag *is defined as follows:*

$$\mathsf{tag}(v) = \begin{cases} \mathsf{Int} & v = n \\ \mathsf{Bool} & v = b \\ (\mathsf{Bool} \to \mathsf{Bool}) & v = \mathsf{not} \\ (\mathsf{Int} \to \mathsf{Int}) & v = \mathsf{add1} \\ T_1 \to T_2 & v = ((\lambda x.\ t_2)^{T_1 \to T_2})[s] \end{cases}$$

$$\frac{\ldots\quad \mathsf{noIsIn}(x_1, [s_1])}{x_1[s_1]\ v_2 \longrightarrow \mathsf{NameError}} \quad (\mathsf{NameErrApp})$$

$$\frac{\mathsf{noIsIn}(x, [s])}{\mathsf{add1}\ x[s] \longrightarrow \mathsf{NameError}} \quad (\mathsf{NameErrSum})$$

$$\frac{\mathsf{noIsIn}(x, [s])}{\mathsf{not}\ x[s] \longrightarrow \mathsf{NameError}} \quad (\mathsf{NameErrNegation})$$

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

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

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

Figure 10: Error rules for $\lambda\backslash_S$.

$$\begin{array}{lllr}
\ldots & & & \\
t & ::= & & \mathsf{terms} \\
 & & \ldots & \\
 & & t :: S & \mathsf{ascription} \\
c & ::= & & \mathsf{configurations} \\
 & & \ldots & \\
 & & c :: S &
\end{array}$$

Figure 11: Syntax of the $\lambda\backslash_S$ with ascriptions.

Definition 8 ($\mathsf{filter}(\cdot,\cdot)$)**.** *The operator* filter *is defined as follows:*

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

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

$$\mathsf{lookup}(w_1, w_2) = \{(v_1, v_2) \mid v_1 \in w_1 \wedge v_2 \in w_2 \wedge \mathsf{tag}(v_1) = T_1 \to T_2 \wedge \mathsf{tag}(v_2) = T_1\}$$

1.6 Static semantic for $\lambda\backslash_T$

Definition 10 ($\oplus$)**.** *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 11 (lookup)**.** *The operator* lookup *is defined as follows:*

$$\mathsf{lookup}(T_1^*, T_2^*) = \{T_1 \to T_2 \mid T_1 \to T_2 \in T_1^* \ \wedge \ T_1 \in T_2^* \ \}$$

Definition 12 ($\Gamma(s)$)**.** *The typing context built from a substitution s, writing $\Gamma(s)$, it is defined as follows:*

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

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

$$\ldots$$

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

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

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

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

$$error :: S \longrightarrow error \quad \text{(AscErr)}$$

$$\frac{\mathsf{noIsIn}(x, [s])}{x[s] :: S \longrightarrow \mathsf{NameError}} \quad \text{(NameErrAsc)}$$

$$\frac{\mathsf{tag}(v) \neq S}{v :: S \longrightarrow \mathsf{TypeError}} \quad \text{(TypeErrAsc)}$$

$$\frac{\neg\mathsf{lookup}(x, [s], S, v)}{x[s] :: S \longrightarrow v} \quad \text{(DisErrAsc)}$$

Figure 12: Rules for $\lambda\backslash_S$ with ascriptions.

		$\ldots$	
w	$::=$		multi − value
		v	value
		$\{\overline{v}\}$	set of values
c	$::=$		configurations
		w	
		$t[s]$	
		$c\ c$	
		mlet $x = c$ in c	
		$c :: S$	
		e	error
s	$::=$		explicit substitutions
		$\bullet$	empty substitution
		$x \mapsto w, s$	variable substitution
e	$::=$		errors
		$\ldots$	
		AmbiguityError	ambiguity error

Figure 13: Syntax of the $\lambda\backslash_\omega$.

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

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

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

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

$$\{\overline{v}\}[s] \longrightarrow \{\overline{v}\} \qquad \textsf{(MultiValue)}$$

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

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

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

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

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

$$\frac{\textsf{filter}(\{\overline{v}\}, \textsf{Fun}) = \{v_1\}}{\{\overline{v}\}\ w_2 \longrightarrow v_1\ w_2} \qquad \textsf{(AppMultValue)}$$

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

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

Figure 14: Reduction rules for $\lambda\backslash_\omega$.

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

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

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

Figure 15: Reduction rules for $\lambda\backslash_1$.

$$\cdots$$

$$\frac{\mathsf{filter}(\{\overline{v}\}, S) = \{\ \}}{\{\overline{v}\} :: S \longrightarrow \mathsf{DispatchError}} \quad (\mathsf{DisErrAsc})$$

$$\frac{\mathsf{filter}(\{\overline{v}\}, \mathsf{Fun}) = \{\ \}}{\{\overline{v}\}\ w_2 \longrightarrow \mathsf{DispatchError}} \quad (\mathsf{DisErrApp})$$

$$\frac{\mathsf{filter}(\{\overline{v}\}, \mathsf{Int}) = \{\ \}}{\mathsf{add1}\ \{\overline{v}\} \longrightarrow \mathsf{DispatchError}} \quad (\mathsf{DisErrSum})$$

$$\frac{\mathsf{filter}(\{\overline{v}\}, \mathsf{Bool}) = \{\ \}}{\mathsf{not}\ \{\overline{v}\} \longrightarrow \mathsf{DispatchError}} \quad (\mathsf{DisErrNegation})$$

$$\frac{\mathsf{filter}(\{\overline{v}\}, S) = w\ \wedge \|w\| > 1}{\{\overline{v}\} :: S \longrightarrow \mathsf{AmbiguityError}} \quad (\mathsf{AmbErrAsc})$$

$$\frac{\mathsf{filter}(\{\overline{v}\}, \mathsf{Fun}) = w\ \wedge \|w\| > 1}{\{\overline{v}\}\ w_2 \longrightarrow \mathsf{AmbiguityError}} \quad (\mathsf{AmbErrApp})$$

$$\frac{\mathsf{filter}(\{\overline{v}\}, \mathsf{Int}) = w\ \wedge \|w\| > 1}{\mathsf{add1}\ \{\overline{v}\} \longrightarrow \mathsf{AmbiguityError}} \quad (\mathsf{AmbErrSum})$$

$$\frac{\mathsf{filter}(\{\overline{v}\}, \mathsf{Bool}) = w\ \wedge \|w\| > 1}{\mathsf{not}\ \{\overline{v}\} \longrightarrow \mathsf{AmbiguityError}} \quad (\mathsf{AmbErrNegation})$$

Figure 16: Error rules for $\lambda\backslash_1$.

t	$::=$		terms
		b	boolean value
		n	numeric value
		op	operator
		$(\lambda x.\ t)^{T \to T}$	abstraction
		x	variable
		$t\ t$	application
		mlet $x = t$ in t	overloading let
		$t :: T$	ascription
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
c	$::=$		configurations
		w	
		$t[s]$	
		$c\ c$	
		mlet $x : T = c$ in c	
		$c :: T$	
		e	error
e	$::=$		errors
		NameError	name error
		TypeError	type error
		DispatchError	dispatch error
		AmbiguityError	ambiguity error
s	$::=$		explicit substitutions
		$\bullet$	empty substitution
		$x \mapsto w, s$	variable substitution

Figure 17: Syntax of the $\lambda\backslash_T$.

$$\cdots$$

$$(\lambda x.\ t_2)^{T_1 \to T_2})[s]\ w \longrightarrow (t_2[x \mapsto w, s]) :: T_2 \qquad (\mathsf{App})$$

$$\frac{\{(v_1, v_2)\} = \mathsf{lookup}(\{\overline{v}\}, w_2)}{\{\overline{v}\}\ w_2 \longrightarrow v_1\ v_2} \qquad (\mathsf{AppMultValue})$$

Figure 18: Reduction rules for $\lambda\backslash_T$.

	$\cdots$		
T^*	$::=$		multi − types
		$\{\overline{T}\}$	multi − type
Γ	$::=$		typing contexts
		$\varnothing$	empty context
		$\Gamma, x : T$	term variable binding
ϕ	$::=$		multi − typing contexts
		$\varnothing$	empty context
		$\phi, x : T^*$	term variable binding

Figure 19: Syntax for $\lambda\backslash_T$ with static semantic.

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

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

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

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

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

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

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

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

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

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

$$\frac{\begin{array}{c} \Gamma; \phi \vdash t_1 : T_1^* \quad \Gamma; \phi \vdash t_2 : T_2^* \\ \mathsf{lookup}(T_1^*, T_2^*) = \{T_1 \to T_2\} \end{array}}{\Gamma; \phi \vdash t_1\ t_2 : \{T_2\}} \qquad (\mathsf{TApp})$$

Figure 20: Term typing rules.

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

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

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

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

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

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

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

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

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

$$\frac{\Gamma \vdash_c v_i : \{T_i\}}{\Gamma \vdash_c \{\overline{v}\} : \{\overline{T}\}} \qquad (\mathsf{CTMulTVal})$$

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

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

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

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

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

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

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

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

$$\frac{\begin{array}{c} \Gamma \vdash_c c_1 : T_1^* \qquad \Gamma \vdash_c c_2 : T_2^* \\ \mathsf{lookup}(T_1^*, T_2^*) = \{T_1 \to T_2\} \end{array}}{\Gamma \vdash_c c_1\ c_2 : \{T_2\}} \qquad (\mathsf{CSTApp})$$

Figure 21: Configuration typing rules.