1 Syntax

1.1 Source Syntax

$$
\begin{array}{lrcl}
\text{Types} & T & ::= & \alpha \mid \top \mid \tau_1 \to \tau_2 \mid \forall\alpha.\,\tau \mid \tau_1 \,\&\, \tau_2 \mid \{l{:}\tau\} \\
\text{Expressions} & E & ::= & x \mid \top \mid \lambda(x{:}\tau).\,e \mid e_1\ e_2 \mid \Lambda\alpha.\,e \mid e\ \tau \mid e_1,,e_2 \mid \{l = e\} \mid e.l \mid e \setminus l \\
 & & \mid & \texttt{sig}\ s[\overline{\alpha}]\ \texttt{where}\ \overline{l : \tau}\ \texttt{in}\ e \\
 & & \mid & \texttt{sig}\ s_1[\overline{\alpha_1}]\ \texttt{extends}\ \overline{s_2[\overline{\alpha}]}\ \texttt{where}\ \overline{l : \tau}\ \texttt{in}\ e \\
 & & \mid & \texttt{algebra}\ x\ \texttt{implements}\ \overline{s[\overline{\tau}]}\ \texttt{where}\ \overline{l@(l_1\ \overline{x_1}) = e_1}\ \texttt{in}\ e \\
 & & \mid & \texttt{algebra}\ x\ \texttt{extends}\ \overline{x_0}\ \texttt{implements}\ \overline{s[\overline{\tau}]}\ \texttt{where}\ \overline{l@(l_1\ \overline{x_1}) = e_1}\ \texttt{in}\ e \\
 & & \mid & \texttt{data}\ d\ \texttt{from}\ s[\overline{\alpha_0}].\alpha_1\ \texttt{in}\ e \\
 & & \mid & \texttt{let}\ x\ (\overline{x_1 : \tau_1})\ (\overline{x_2 : d[\overline{\tau}]}) : d[\overline{\tau}] = e_1\ \texttt{in}\ e \\
 & & \mid & \texttt{<}\overline{x}\texttt{>} \\
\text{Contexts} & \Gamma & ::= & \epsilon \mid \Gamma, \alpha \mid \Gamma, x{:}\tau \\
 & & \mid & \Gamma, s[\overline{\alpha}] \to \overline{l : \tau} \\
 & & \mid & \Gamma, x \multimap \overline{s[\overline{\tau}]} \\
 & & \mid & \Gamma, d \rightsquigarrow s[\overline{\alpha_0}].\alpha_1 : \tau \\
\text{Labels} & l & & \text{(fields)} \\
 & s & & \text{(interfaces)} \\
 & d & & \text{(datatypes)} \\
\text{Syntactic sugars} & \circ & ::= & s[\overline{\tau_0}] \\
 & \bullet & ::= & [\overline{\tau_0}/\overline{\alpha}]\Gamma(s) \\
 & \circ & ::= & d(\overline{\tau_0}) \\
 & \bullet & ::= & [\overline{\tau_0}/(\overline{\alpha_0} \setminus \alpha_1)]\Gamma(d)
\end{array}
$$

1.2 Target Syntax

$$
\begin{array}{lrcl}
\text{Types} & T & ::= & \alpha \mid \top \mid \tau_1 \to \tau_2 \mid \forall\alpha.\,\tau \mid \tau_1 \,\&\, \tau_2 \mid \{l{:}\tau\} \\
\text{Expressions} & E & ::= & x \mid \top \mid \lambda(x{:}\tau).\,e \mid e_1\ e_2 \mid \Lambda\alpha.\,e \mid e\ \tau \mid e_1,,e_2 \mid \{l = e\} \mid e.l \mid e \setminus l \\
\text{Contexts} & \Gamma & ::= & \epsilon \mid \Gamma, \alpha \mid \Gamma, x{:}\tau \\
\text{Labels} & l & & \\
\text{Syntactic sugars} & \circ & ::= & \texttt{let}\ x : \tau = e_1\ \texttt{in}\ e_2 \\
 & \bullet & ::= & (\lambda(x{:}\tau).\,e_2)\ e_1
\end{array}
$$

2 Translation Rules

$\boxed{\Gamma \vdash e : \tau \Rightarrow E}$

$$
\frac{\Gamma, s[\overline{\alpha}] \to \overline{l : \tau} \ \vdash\ e \ :\ \tau_* \ \Rightarrow\ E}{\Gamma \ \vdash\ \texttt{sig}\ s[\overline{\alpha}]\ \texttt{where}\ \overline{l : \tau}\ \texttt{in}\ e \ :\ \tau_* \ \Rightarrow\ \texttt{let}\ merge_s : \ldots = \ldots\ \texttt{in}\ E}
$$

$$
\frac{\overline{\Gamma \vdash s_2[\overline{\alpha_2}]} \qquad \Gamma, s_1[\overline{\alpha_1}] \to \mathbf{U}_{\varnothing}\overline{[\overline{\alpha}/\overline{\alpha_2}]\Gamma(s_2)}\ \mathbf{U}_{\leftarrow}\overline{l : \tau} \ \vdash\ e \ :\ \tau_* \ \Rightarrow\ E}{\Gamma \ \vdash\ \texttt{sig}\ s_1[\overline{\alpha_1}]\ \texttt{extends}\ \overline{s_2[\overline{\alpha}]}\ \texttt{where}\ \overline{l : \tau}\ \texttt{in}\ e \ :\ \tau_* \ \Rightarrow\ \texttt{let}\ merge_{s_1} : \ldots = \ldots\ \texttt{in}\ E}
$$

$$
\frac{\overline{\Gamma \vdash s[\overline{\alpha}]} \qquad \overline{\Gamma, \overline{x_1} : [\overline{\tau}/\overline{\alpha}]gen2_A(l_1) \ \vdash\ e_1 \ :\ \tau_1 \ \Rightarrow\ E_1} \qquad \Gamma, x : \&\overline{[\overline{\tau}/\overline{\alpha}]\Gamma(s)}, x \multimap \overline{s[\overline{\tau}]} \ \vdash\ e \ :\ \tau_* \ \Rightarrow\ E}{\begin{array}{c}\Gamma \ \vdash\ \texttt{algebra}\ x\ \texttt{implements}\ \overline{s[\overline{\tau}]}\ \texttt{where}\ \overline{l@(l_1\ \overline{x_1}) = e_1}\ \texttt{in}\ e \ :\ \tau_* \ \Rightarrow \\ \texttt{let}\ x : \&\overline{[\overline{\tau}/\overline{\alpha}]\Gamma(s)} = \{\overline{l_1 = \lambda(\overline{x_1}{:}[\overline{\tau}/\overline{\alpha}]gen2_A(l_1)).\{l = E_1\}}\}\ \texttt{in}\ E\end{array}}
$$

$$
\frac{\overline{\Gamma \vdash s[\overline{\alpha}]} \qquad \overline{\Gamma \vdash x_0} \qquad \overline{\Gamma, \overline{x_1} : [\overline{\tau}/\overline{\alpha}]gen2_A(l_1) \ \vdash\ e_1 \ :\ \tau_1 \ \Rightarrow\ E_1} \qquad \Gamma, x : \&\overline{[\overline{\tau}/\overline{\alpha}]\Gamma(s)}, x \multimap \overline{s[\overline{\tau}]} \ \vdash\ e \ :\ \tau_* \ \Rightarrow\ E}{\begin{array}{c}\Gamma \ \vdash\ \texttt{algebra}\ x\ \texttt{extends}\ \overline{x_0}\ \texttt{implements}\ \overline{s[\overline{\tau}]}\ \texttt{where}\ \overline{l@(l_1\ \overline{x_1}) = e_1}\ \texttt{in}\ e \ :\ \tau_* \ \Rightarrow \\ \texttt{let}\ x : \&\overline{[\overline{\tau}/\overline{\alpha}]\Gamma(s)} = \overline{x_0},,\{\overline{l_1 = \lambda(\overline{x_1}{:}[\overline{\tau}/\overline{\alpha}]gen2_A(l_1)).\{l = E_1\}}\}\ \texttt{in}\ E\end{array}}
$$

$$
\frac{\Gamma \vdash s[\overline{\alpha}] \to \overline{l : \tau} \qquad \Gamma, d \rightsquigarrow s[\overline{\alpha_0}].\alpha_1 : \{accept : \forall\alpha_1.\,s[\overline{\alpha_0}] \to \alpha_1\} \ \vdash\ e \ :\ \tau_* \ \Rightarrow\ E}{\Gamma \ \vdash\ \texttt{data}\ d\ \texttt{from}\ s[\overline{\alpha_0}].\alpha_1\ \texttt{in}\ e \ :\ \tau_* \ \Rightarrow\ \texttt{let}\ \overline{gen3(l)} = \ldots\ \texttt{in}\ E} \quad \text{NEED TCHECK}
$$

$$
\frac{\Gamma \vdash d \qquad \Gamma, \overline{x_1 : \tau_1}, \overline{x_2 : d[\overline{\tau}]} \ \vdash\ e_1 \ :\ d[\overline{\tau}] \ \Rightarrow\ E_1 \qquad \Gamma, x : \overline{\tau_1} \to \overline{d[\overline{\tau}]} \to d[\overline{\tau}] \ \vdash\ e \ :\ \tau_* \ \Rightarrow\ E}{\Gamma \ \vdash\ \texttt{let}\ x\ (\overline{x_1 : \tau_1})\ (\overline{x_2 : d[\overline{\tau}]}) : d[\overline{\tau}] = e_1\ \texttt{in}\ e \ :\ \tau_* \ \Rightarrow\ \texttt{let}\ x = [gen4(d)]E_1\ \texttt{in}\ E} \quad \text{NEED TCHECK}
$$

$$
\frac{\overline{\Gamma \vdash x \multimap s[\overline{\tau}]} \qquad \Gamma \vdash s[\overline{\alpha}]}{\Gamma \ \vdash\ \texttt{<}\overline{x}\texttt{>} \ :\ s[\&\overline{\tau}] \ \Rightarrow\ merge_s\,\overline{[\overline{\tau}]}\ \overline{x}}
$$

$merge_s$: the merge algebra for object algebra interface s.

$$
merge_s : \forall\overline{\alpha_A}.\,\forall\overline{\alpha_B}.\,s[\overline{\alpha_A}] \to s[\overline{\alpha_B}] \to s[\overline{\alpha_A \,\&\, \alpha_B}] = \Lambda\overline{\alpha_A}.\,\Lambda\overline{\alpha_B}.\,\lambda(alg_1{:}s[\overline{\alpha_A}]).\,\lambda(alg_2{:}s[\overline{\alpha_B}]).\{\overline{l = [\overline{\alpha_A \,\&\, \alpha_B}/\overline{\alpha}]gen(l)}\}
$$

$\mathit{merge}_s \overline{[\tau]}\ \overline{x}$: generalizing $\mathit{merge}_s [\overline{\tau_i}][\overline{\tau_j}]\ x_i\ x_j$.

$\mathit{gen}(l)$: $\lambda(\overline{x}{:}\mathit{gen2}_A(l)).\ \mathit{alg}_1.l\ \overline{x},, \mathit{alg}_2.l\ \overline{x}$.

$\mathit{gen2}(l)$: get the type from context $\Gamma(s).l$. $\mathit{gen2}_A(l)$ derives the type of arguments in field l, and $\mathit{gen2}_B(l)$ gets the return type.

$\mathit{gen3}(l)$: for each case l, generate an auxiliary function for building structures. Only consider those with return type $[\overline{\alpha_0}/\overline{\alpha}]\mathit{gen2}_B(l) = \alpha_1$, where $\overline{\alpha_0}, \alpha_1$ are the ones from $\texttt{data}\ d\ \texttt{from}\ s[\overline{\alpha_0}].\alpha_1\ \texttt{in}\ e$.

$[\mathit{gen4}(d)]$: $[\overline{l[\overline{\tau}]/l}]$. Only when $d \rightsquigarrow s[\overline{\alpha_0}].\alpha_1$, l is a constructor in s, and $\mathit{gen3}(l)$ exists.

$\mathbf{U}_{\varnothing}$ denotes the disjoint union on records, and $\mathbf{U}_{\leftarrow}$ also means the union, but the fields on the right side will replace the left ones with same names.

3 Auxiliary Rules for Expanding Types

$$\boxed{\Gamma \vdash \tau \Rightarrow T} \qquad \frac{\Gamma \vdash s[\overline{\alpha}] \to \tau_*}{\Gamma \vdash s[\overline{\tau_0}] \Rightarrow [\overline{\tau_0}/\overline{\alpha}]\tau_*} \qquad \frac{\Gamma \vdash d \rightsquigarrow s[\overline{\alpha_0}].\alpha_1 : \tau_* \qquad \Gamma \vdash s[\overline{\alpha}] \to \overline{l : \tau}}{\Gamma \vdash d[\overline{\tau_0}] \Rightarrow [\overline{\tau_0}/(\overline{\alpha_0}\backslash\alpha_1)]\tau_*}$$

The rules here are consistent with the syntactic sugars before.

4 Notes

The rules should support both special syntax for algebras and common syntax.

- **sig:** (1) in the environment; (2) as a type synonym.
- **alg:** (1) in the environment; (2) as a function.
- **data:** (1) in the environment; (2) as a type synonym.

Each datatype has only one sort. And instantiation only works for datatypes.

Type and consistency check need. The type-check has already been highlighted in translation rules. For consistency, like in the declaration of an algebra, the label l should be consistent. And in the instantiation $<\overline{x}>$, it requires $\overline{x \multimap s[\overline{\tau}]}$ with the same s.

5 Example: ListAlg

Declaration of ListAlg:

$$\mathsf{sig}\ \mathsf{ListAlg}[A,\ L]\ \mathit{where}\ \mathsf{nil} : L,\ \mathsf{cons} : A \to L \to L;$$

Get the types:

$$\begin{array}{rcl@{\qquad\qquad}rcl} \mathit{gen2}_A(\mathsf{nil}) & = & - & \mathit{gen2}_A(\mathsf{cons}) & = & A,\ L \\ \mathit{gen2}_B(\mathsf{nil}) & = & L & \mathit{gen2}_B(\mathsf{cons}) & = & L \end{array}$$

The merge algebra:

$$\begin{array}{rcl} \mathit{mergeListAlg} & = & \Lambda(A1,\ L1).\ \Lambda(A2,\ L2).\ \lambda(\mathit{alg}1{:}\mathsf{ListAlg}[A1,\ L1]).\ \lambda(\mathit{alg}2{:}\mathsf{ListAlg}[A2,\ L2]). \\ & & \{\ \mathsf{nil} \ \ = \mathit{alg}1.\mathsf{nil}\ ,,\ \mathit{alg}2.\mathsf{nil}, \\ & & \ \ \mathsf{cons}\ = \lambda(x{:}A1\ \&\ A2).\ \lambda(y{:}L1\ \&\ L2).\ \mathit{alg}1.\mathsf{cons}\ x\ y\ ,,\ \mathit{alg}2.\mathsf{cons}\ x\ y\ \} \end{array}$$

Declaration of List:

$$\mathsf{data}\ \mathsf{List}\ \mathsf{from}\ \mathsf{ListAlg}[A,\ L].L;$$

Generate auxiliary constructors:

$$\begin{array}{rcl} \mathit{gen3}(\mathsf{nil}) & : & \forall A.\ \mathsf{List}[A] \\ & = & \Lambda A.\{\mathsf{accept}\ =\ \Lambda L.\ \lambda(\mathit{alg}{:}\mathsf{ListAlg}[A,\ L]).\ \mathit{alg}.\mathsf{nil}\} \\ \mathit{gen3}(\mathsf{cons}) & : & \forall A.\ A \to \mathsf{List}[A] \to \mathsf{List}[A] \\ & = & \Lambda A.\ \lambda(x{:}A).\ \lambda(y{:}\mathsf{List}[A]).\{\mathsf{accept}\ =\ \Lambda L.\ \lambda(\mathit{alg}{:}\mathsf{ListAlg}[A,\ L]).\ \mathit{alg}.\mathsf{cons}\ x\ (y.\mathsf{accept}[L]\ \mathit{alg})\} \end{array}$$