1 Syntax

1.1 Source Syntax

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

1.2 Target Syntax

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

2 Translation Rules

$$
\boxed{\Gamma \vdash e : \tau \Rightarrow E}
\qquad
\frac{\Gamma, s[\overline{\alpha}] \to \overline{l:\tau} \;\vdash\; e \;:\; \tau_* \;\Rightarrow\; E}{\Gamma \;\vdash\; \mathtt{sig}\;s[\overline{\alpha}]\;\mathtt{where}\;\overline{l:\tau}\;\mathtt{in}\;e \;:\; \tau_* \;\Rightarrow\; \mathtt{let}\;\mathit{merge}_s : \ldots = \ldots\;\mathtt{in}\;E}
$$

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

$$
\frac{\overline{\Gamma \vdash s[\overline{\alpha}]} \qquad \overline{\Gamma, \overline{x_1} : \mathit{gen2}_A(l_1) \;\vdash\; e_1 \;:\; \tau_1 \;\Rightarrow\; E_1} \qquad \overline{\tau_1 <: \mathit{gen2}_B(l_1)} \qquad \Gamma, x : \&\overline{[\overline{\tau}/\overline{\alpha}]\Gamma(s)}, x \multimap \overline{s[\overline{\tau}]} \;\vdash\; e \;:\; \tau_* \;\Rightarrow\; E}{\Gamma \;\vdash\; \mathtt{algebra}\;x\;\mathtt{implements}\;\overline{s[\overline{\tau}]}\;\mathtt{where}\;\overline{l@(l_1\;\overline{x_1}) = e_1}\;\mathtt{in}\;e \;:\; \tau_* \;\Rightarrow\; \mathtt{let}\;x : \&\overline{[\overline{\tau}/\overline{\alpha}]\Gamma(s)} = \{\overline{l_1 = \lambda(\overline{x_1}{:}\mathit{gen2}_A(l_1)).\,E_1}\}\;\mathtt{in}\;E}
$$

$$
\frac{\overline{\Gamma \vdash s[\overline{\alpha}]} \qquad \overline{\Gamma \vdash x_0 : \tau_0} \qquad \overline{\Gamma, \overline{x_1} : \mathit{gen2}(l_1) \;\vdash\; e_1 \;:\; \tau_1 \;\Rightarrow\; E_1} \qquad \Gamma, x : \&\overline{[\overline{\tau}/\overline{\alpha}]\Gamma(s)} \;\vdash\; e \;:\; \tau_* \;\Rightarrow\; E}{\begin{array}{c}\Gamma \;\vdash\; \mathtt{algebra}\;x\;\mathtt{extends}\;\overline{x_0}\;\mathtt{implements}\;\overline{s[\overline{\tau}]}\;\mathtt{where}\;\overline{l@(l_1\;\overline{x_1}) = e_1}\;\mathtt{in}\;e \;:\; \tau_* \;\Rightarrow \\ \mathtt{let}\;x : \&\overline{[\overline{\tau}/\overline{\alpha}]\Gamma(s)} = \overline{x_0},,\{\overline{l_1 = \lambda(\overline{x_1}{:}\mathit{gen2}(l_1)).\,E_1}\}\;\mathtt{in}\;E\end{array}}
$$

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

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

$$
\frac{\Gamma \vdash s[\overline{\alpha}] \qquad \Gamma \;\vdash\; e \;:\; \tau_0 \;\Rightarrow\; E}{\Gamma \;\vdash\; e[\overline{\tau}]\texttt{<}\overline{x}\texttt{>} \;:\; \tau_* \;\Rightarrow\; E.\mathit{accept}[\&\overline{\tau}]\;(\mathit{merge}_s\;\overline{x})}
$$

merge_s: the merge algebra for object algebra interface s.

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

$gen(l)$: $\lambda(\overline{x}:...).\, alg1.\overline{x},, alg2.\overline{x}$.

$gen2(l)$: get the type from context $\Gamma(s).l$.

$gen3(l)$: for each case l, generate an auxiliary function for building structures. Only consider those with return type α_1 in `data` d `from` $s[\overline{\alpha_0}].\alpha_1$ `in` e.

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

3 Auxiliary Rules for Expanding Types

4 Amendment

4.1 In translation: alg

Question: Type-check for $\tau_1 <:$ the return type in $\Gamma(s)$?

4.2 In translation: algext

Question: Type-check for $\overline{s[\overline{\tau}]} = \overline{\tau_0} + ...$?

4.3 In translation: datatype

Question: Check if $\alpha_1 \in \overline{\alpha_0}$? Not sure if something like ... makes sense.

4.4 In translation: insta

Question: Check if e has the field "accept"? And the relationship between τ_0 and τ_*?

4.5 Critical: algebras in context?

Question: Put algebras into context? Need to check more for types in that case. But instantiation becomes more concise. Currently the translation rule for instantiation doesn't really work (it doesn't know which interface to use, since merge algebras are namespaced). The merge algebra is limited. And some constructors potentially cannot be generated automatically.

4.6 Extension: sig as type? structure building?

Question: Currently a structure can only be built from datatype. With signatures as types, the code could be more flexible.

4.7 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.