A Appendix

A.1 LEMMA 1 and Proof

LEMMA 1. For any expression e under an interface table $\mathcal{I}$ IT where $\Gamma \vdash e \in I^{\mathcal{I}}$, $\mathcal{I}$ has `@ObjOf` annotation and $[\![\mathcal{I}]\!] = \mathcal{I}'$, then under the interface table $\mathcal{I}'$ IT, $\Gamma \vdash e \in I^{\mathcal{I}}$.

Proof. By induction on the typing rules: by the grammar shown in Figure 4, there are 6 cases for an arbitrary expression e:

- Variables are typed in the same exact way.
- Field update. The type preservation is ensured by induction.
- A method call (normal, static or super). The corresponding method declaration won't be "removed" by the translation, also the types remain unchanged. The only work `@ObjOf` does is adding a static method `of` to the interface, however, a pre-condition of the translation is $\texttt{of} \notin \mathsf{dom}(I^{\mathcal{I}})$, so adding `of` method has no way to affect any formerly well typed method call.
- An object creation. Adding the `of` method doesn't introduce unimplemented methods to an interface, moreover, the static method is not inheritable, hence after translation such an object creation still type checks and has the right type by induction.

◀

LEMMA 1b. For any expression e under an interface table $\mathcal{I}$ IT where there is no heir of $I^{\mathcal{I}}$, $\Gamma \vdash e \in I^{\mathcal{I}}$, $\mathcal{I}$ has `@Obj` annotation and $[\![\mathcal{I}]\!] = \mathcal{I}'$, then under the interface table $\mathcal{I}'$ IT, $\Gamma \vdash e \in \_ <: I^{\mathcal{I}}$.

Proof. The proof follows the same scheme of the Lemma1, but for the case of method call the return type may be refined with a subtype. This is still ok since we require $\_ <: I^{\mathcal{I}}$. On the other side, this weaker result still allows the application on the method call typing rules, since in the premises the types of the actual parameter are required to be a subtype of the formal one. ◀

A.2 LEMMA 2 and Proof

LEMMA 2. If $\mathcal{I}$ has `@ObjOf` annotation and $I^{\mathcal{I}}$ OK in $\mathcal{I}$ IT, then $[\![\mathcal{I}]\!]$ OK in $[\![\mathcal{I}]\!]$ IT.

Proof. By the rule (T-INTF) in Figure 5, we divide the proof into two parts.
Part I. For each default or static method in the domain of $[\![I^{\mathcal{I}}]\!]$, the type of the return value is compatible with the method's return type.

Since $\mathcal{I}$ OK, and by **LEMMA 1**, all the existing default and static methods are well typed in $[\![\mathcal{I}]\!]$, except for the new method `of`. It suffices to prove that it still holds for $\mathsf{ofMethod}(I)$.

By the definition of $\mathsf{ofMethod}(I)$, the return value is an object

$$\texttt{return new } I^{\mathcal{I}}\texttt{()\{...\}}$$

To prove it is of type $I^{\mathcal{I}}$, we use the typing rule (T-OBJ).

- All field initializations are type correct. By the definition of $\mathsf{ofMethod}(I^{\mathcal{I}})$ in Section 5.1, the fields $m_1, \ldots, m_n$ are initialized by `of`'s arguments, and types are compatible.
- All method bodies are well-typed.

- Typing of the i-th getter m_i.

  $\Gamma, m_i : I_i, \texttt{this} : I^{\mathcal{I}} \vdash m_i \in I_i$

  We know that $I_i = I^{mh_i}$ since the i-th getter has its return type the same as the corresponding field m_i.
- Typing of the `with-` method of an arbitrary field m_i. By Section 5.1, if the `with-` method of m_i is well-defined, it has the form

  $I^{\mathcal{I}}$ `with#`m_i`(`I_i `_val){return` $I^{\mathcal{I}}$`.of(`$\overline{e}_i$`);}`

  $\overline{e}_i$ is obtained by replacing m_i with `_val` in the list of fields, and since they have the same type I_i, the arguments $\overline{e}_i$ are compatible with $I^{\mathcal{I}}$`.of` method. Hence

  $\Gamma, m_1 : I_1 \ldots m_n : I_n, \texttt{this} : I^{\mathcal{I}}, \texttt{\_val} : I_i \vdash I^{\mathcal{I}}\texttt{.of(}\overline{e}_i\texttt{)} \in I^{\mathcal{I}}$

  We know that $I^{\mathcal{I}} = I^{mh_i}$ by the return type of `with#`m_i shown as above.
- Typing of the i-th setter $\_m_i$. If the $\_m_i$ method is well-defined, it has the form

  $I^{\mathcal{I}}$ $\_m_i$`(`I_i `_val){` m_i`= _val;return this;}`

  By (T-UPDATE), the assignment "m_i`= _val;`" is correct since m_i and `_val` have the same type I_i, and the return type is decided by **`this`**.

  $\Gamma, \texttt{this} : I^{\mathcal{I}}, \texttt{\_val} : I_i \vdash \texttt{this} \in I^{\mathcal{I}}$

  We know that $I^{\mathcal{I}} = I^{mh_i}$ by the return type of $\_m_i$ shown as above.

- All method headers are valid with respect to the domain of $I^{\mathcal{I}}$. Namely

$$\mathsf{sigvalid}(mh_1 \ldots mh_n, I)$$

For convenience, we use "$meth$ in $\mathsf{ofMethod}(I^{\mathcal{I}})$" to denote that $meth$ is one of the implemented methods in the return expression of $\mathsf{ofMethod}(I^{\mathcal{I}})$, namely **`new`** $I^{\mathcal{I}}$`(){...}`.

- For the i-th getter m_i,

$$\begin{array}{rl} & I_i\ m_i\texttt{()\{...\}}\ \text{in}\ \mathsf{ofMethod}(I^{\mathcal{I}}) \\ \text{implies} & I_i\ m_i\texttt{();} \in \mathsf{fields}(I^{\mathcal{I}}) \\ \text{implies} & I_i\ m_i\texttt{();} = \mathsf{mbody}(m_i, I^{\mathcal{I}}) \\ \text{implies} & I_i\ m_i\texttt{();} <: \mathsf{mbody}(m_i, I^{\mathcal{I}}) \end{array}$$

- For the `with#`m_i method,

$$\begin{array}{rl} & I^{\mathcal{I}}\ \texttt{with\#}m_i\texttt{(}I_i\ \texttt{\_val)\{...\}}\ \text{in}\ \mathsf{ofMethod}(I^{\mathcal{I}}) \\ \text{implies} & \mathsf{mbody}(\texttt{with\#}m_i, I^{\mathcal{I}})\ \text{is of form}\ mh; \\ \text{with} & \mathsf{valid}(I^{\mathcal{I}}) \\ \text{implies} & \mathsf{isWith}(\mathsf{mbody}(\texttt{with\#}m_i, I^{\mathcal{I}}), I^{\mathcal{I}}) \\ \text{implies} & I^{\mathcal{I}}\ \texttt{with\#}m_i\texttt{(}I_i\ \texttt{\_val);} <: \mathsf{mbody}(\texttt{with\#}m_i, I^{\mathcal{I}}) \end{array}$$

- For the i-th setter $\_m_i$,

$$\begin{array}{rl} & I^{\mathcal{I}}\ \_m_i\texttt{(}I_i\ \texttt{\_val)\{...\}}\ \text{in}\ \mathsf{ofMethod}(I^{\mathcal{I}}) \\ \text{implies} & \mathsf{mbody}(\_m_i, I^{\mathcal{I}})\ \text{is of form}\ mh; \\ \text{with} & \mathsf{valid}(I^{\mathcal{I}}) \\ \text{implies} & \mathsf{isSetter}(\mathsf{mbody}(\_m_i, I^{\mathcal{I}}), I^{\mathcal{I}}) \\ \text{implies} & I^{\mathcal{I}}\ \_m_i\texttt{(}I_i\ \texttt{\_val);} <: \mathsf{mbody}(\_m_i, I^{\mathcal{I}}) \end{array}$$

- All abstract methods in the domain of $I^{\mathcal{I}}$ have been implemented. Namely

$$\mathsf{alldefined}(mh_1 \dots mh_n, I)$$

Here we simply refer to $\mathsf{valid}(I^{\mathcal{I}})$, since it guarantees each abstract method to satisfy $\mathsf{isField}$, isWith or $\mathsf{isSetter}$. But that object includes all implementations for those cases. A getter m_i is generated if it satisfies $\mathsf{isField}$; a `with-` method is generated for the case isWith, by the definition of $\mathsf{withMethod}$; a setter for $\mathsf{isSetter}$, similarly, by the definition of $\mathsf{setterMethod}$. Hence it is of type $I^{\mathcal{I}}$ by (T-OBJ).

Part II. Next we check that in $[\![\mathcal{I}]\!]$,

$$\mathsf{dom}([\![\mathcal{I}]\!]) = \mathsf{dom}(I_1) \cup \dots \cup \mathsf{dom}(I_n) \cup \mathsf{dom}(\overline{meth}) \cup \mathsf{dom}(meth')$$

Since $\mathcal{I}$ OK, we have $\mathsf{dom}(\mathcal{I}) = \mathsf{dom}(I_1) \cup \dots \cup \mathsf{dom}(I_n) \cup \mathsf{dom}(\overline{meth})$, and hence it is equivalent to prove

$$\mathsf{dom}([\![\mathcal{I}]\!]) = \mathsf{dom}(I^{\mathcal{I}}) \cup \mathsf{dom}(meth')$$

This is obvious since a pre-condition of the translation is `of` $\notin \mathsf{dom}(I^{\mathcal{I}})$, so $meth'$ doesn't overlap with $\mathsf{dom}(I^{\mathcal{I}})$. The definition of dom is based on mbody, and here the new domain $\mathsf{dom}([\![\mathcal{I}]\!])$ is only an extension to $\mathsf{dom}(I)$ with the `of` method, namely $meth'$. Also note that after translation, there are still no methods with conflicted names, since the `of` method was previously not in the domain, hence $[\![\mathcal{I}]\!]$ is well-formed, which finishes our proof. ◀

LEMMA 2b. If $\mathcal{I}$ has `@Obj` annotation $I^{\mathcal{I}}$ OK in $\mathcal{I}$ IT and there is no heir of $I^{\mathcal{I}}$, then $[\![\mathcal{I}]\!]$ OK in $[\![\mathcal{I}]\!]$ IT.

Proof. Part I. Similarly to what already argued for **Lemma 2**, since $\mathcal{I}$ OK, and by **LEMMA 1b**, all the existing default and static methods are well typed in $[\![\mathcal{I}]\!]$ IT. The translation function delegates its work to `@ObjOf` in such way that we can refer to **Lemma 2** to complete this part. Note that all the methods added (directly) by `@Obj` are abstract, and thus there is no body to typecheck.
Part II. Similar to what we already argued for **Lemma 2**, but we need to notice that the newly added methods are valid refinements for already present methods in $\mathsf{dom}(I^{\mathcal{I}})$ before the translation. Thus by last clause of the definition of $\mathsf{override}(\_)$, $\mathsf{mbody}(\_)$ is defined on the same method names. ◀

A.3 THEOREM and Proof

THEOREM `@ObjOf` tuning
If a given interface table $\mathcal{I}$ IT is OK where $\mathcal{I}$ has `@ObjOf`, $\mathsf{valid}(I^{\mathcal{I}})$ and `of` $\notin \mathsf{dom}(I^{\mathcal{I}})$, then the interface table $[\![\mathcal{I}]\!]$ IT is OK.

Proof. LEMMA 2 already proves that $[\![\mathcal{I}]\!]$ is OK. On the other hand, for any $\mathcal{I}' \in \text{IT}\backslash\mathcal{I}$, by **LEMMA 1**, we know that all its methods are still well-typed, and the generated code in translation of `@ObjOf` is only a static method `of`, which has no way to affect the domain of $\mathcal{I}'$, so after translation rule (T-INTF) can still be applied, which finishes our proof. ◀

THEOREM `@Obj` tuning
If a given interface table $\mathcal{I}$ IT is OK where $\mathcal{I}$ has `@Obj`, $\mathsf{valid}(I^{\mathcal{I}})$ and `of` $\notin \mathsf{dom}(I^{\mathcal{I}})$, and there is no heir of $I^{\mathcal{I}}$, then the interface table $[\![\mathcal{I}]\!]$ IT is OK.

Proof. Similar to what already argued for **THEOREM** `@ObjOf` tuning we can apply **LEMMA 2b** and **LEMMA 1b**. Then we finish by **THEOREM** `@ObjOf` tuning. ◀