Backpack: Retrofitting Haskell with Interfaces
Technical Appendix

Scott Kilpatrick
MPI-SWS
skilpat@mpi-sws.org

Derek Dreyer
MPI-SWS
dreyer@mpi-sws.org

Simon Peyton Jones
Microsoft Research
simonpj@microsoft.com

Simon Marlow
Facebook
marlowsd@gmail.com

Friday 12th July, 2013

Contents

1 Introduction

This appendix expands on the POPL'14 submission of the same name ("the paper"). It presents the full definition and metatheory for Backpack. The version of Backpack presented in the paper ("Paper Backpack") makes various simplifications of the system in this appendix ("Backpack") for clarity and readability. We describe those differences here:

- Paper Backpack eliminates the syntactic category of expressions, E, and folds the functionality of this category (and its relevant judgments) into straightforward "duplications" of the $p = E$ binding. Instead of $p = E$ and a separate category of E, Paper Backpack defines $p = [M]$, $p :: [S]$, and $p = p'$. Backpack has an additional semantic object $\mathcal{M}$ which is like an atomic version of $\mathcal{L}$ in that it stores an identity and a view type τ for a single module. Additionally, the type of expressions, Σ, can either mention a single module (in the case of module and signature expressions and atomic names) or multiple modules (in the case of subnamespace projection). Context shapes (in the typing rules) do not require any projection in Paper Backpack as they do in Backpack.

- Logical path names in Paper Backpack are straightforward tokens drawn from the sort *ModPaths*, whereas in Backpack they are dotted lists ("path projections") composed of individual module component names X drawn from sort *ModCompts*; these allow programmers to select subnamespaces. For example, if $\mathsf{A.X}$ and $\mathsf{A.Y}$ are bound in the context, then the binding $\mathsf{B} = \mathsf{A}$ introduces two logical bindings for $\mathsf{B.X}$ and $\mathsf{B.Y}$ that point to those modules.

  Paper Backpack has only a single (binding) rule for paths (*i.e.*, aliases), whereas Backpack has two: one for full path names to modules and another for subnamespaces. We feel that Paper Backpack's treatment of paths as mere names is simpler and sufficiently expressive.

- Thinning is not defined in Paper Backpack, although it is described in the tour (Paper Section 2).

- Paper Backpack simplifies some of the semantic objects in harmless, cosmetic ways. Its definition of *espc* as an identity and a *dent* is a minor technical simplification that facilitates pattern matching in the rules;

moreover, *dent* in Paper Backpack is $\tilde{dspc}$ in Backpack. The `data` part of *dent* is omitted in Paper Backpack.

- Paper Backpack simplifies some semantic objects in more significant ways. There, module types (and shapes) have only two components, whereas in Backpack they have a third component—a list of imported module identities. This is needed in the definition of thinning, specifically, in the definition of one identity depending on another, $\mathsf{depends}_\Phi(-)$.
- The multiple-bindings shaping judgment also synthesizes a substitution in Backpack; in Paper Backpack it does not.

Part I
Backpack definition

2 Package Syntax

(package names)	P	$\in$	*PkgNames*
(module variables)	X	$\in$	*ModCompts*
(paths)	p, q	::=	$\epsilon \mid p.X$
(modules)	M	::=	$\ldots$
(signatures)	S	::=	$\ldots$
(expressions)	E	::=	$p \mid [M] \mid [:: S]$
(bindings)	B	::=	$p = E \mid$ **include** P t r
(thinning)	t	::=	$(\overline{p})$
(renaming)	r	::=	$\langle \overline{p \mapsto p} \rangle$
(package definitions)	D	::=	**package** P t **where** $\overline{B}$
(top-level bindings)	T	::=	$\overline{D}$

$$
\begin{array}{lll}
\text{(paths)} & X & \stackrel{\text{def}}{=} \epsilon.X \\
& X.\epsilon & \stackrel{\text{def}}{=} \epsilon.X \\
& X.(p'.X') & \stackrel{\text{def}}{=} (X.p').X'
\end{array}
$$

3 Miscellaneous Examples

- It might seem strange that logical type (and shape) contexts keep a "view type" (τ) on each mapped identity ν, even though the corresponding physical type context already contains a module type for ν. Consider the following example:

```
package view-type where
  A :: [x :: Int]
  C = A
  C = [x = 1; y = 2]
  B = [import A; export (y); y = x + 5]
```

The import A within the B module resolves to the identity of the A binding which is ν_C, due to the alias C = A. This alias ties together C and A so

that they will have the same module identity, thus when we bind C to some module in the third line, we are actually implementing the hole specified by the first, A binding.

With that in mind, which exact entity named y is exported by B? $[\nu_C]$y or $[\nu_B]$y? In Backpack it is the latter. Why? We specifically want imports to path p to see the interface of what p has been bound to. In this example, we did not actually bind A to anything after the first binding; therefore its interface only exposes an x value. That means that B only sees the x value, and the y that it exports unambiguously refers to its own local value.

In order to support this semantics (in which the "logical" interface for p comes from the bindings of name p only), we need to store a module type in the logical context *in addition to* the real, physical type in the physical context. (In the example above, the real physical type for A is that of ν_C; it exposes both x and y.)

4 Semantic Objects

4.1 Stamps

Stamps represent module identity. Each (non-variable) stamp uniquely identifies a module and its dependencies. Like a closure in the λ-calculus, stamps contain the code and, recursively, the closures (stamps) of its free variables (imports, resp.).

(stamp variables)	α, β	$\in$	$\mathit{StampVars}$
(stamp constructors)	$\mathcal{K}$	$\in$	$\mathit{StampCtors}$
(stamps)	ν	$::=$	$\alpha \mid \mu\alpha.\mathcal{K}\ \overline{\nu}$
(stamp substitutions)	ϕ, θ	$::=$	$\{\overline{\alpha \coloneqq \nu}\}$

- Stamp variables α correspond to identities of signatures without corresponding implementations, and to the provenances of code entities specified in signatures.
- Each application $\mu\alpha.\mathcal{K}\ \overline{\nu}$ represents the closure of a particular module $\mathcal{K}$ with the identities of all its imported modules $\overline{\nu}$. We assume a bijection between stamp constructors $\mathcal{K}$ and module source code M. The μ-binder is used for recursive modules that (transitively) import themselves. When α does not appear in $\overline{\nu}$ we leave off the binder:

$$\text{(stamps)} \quad \mathcal{K}\ \overline{\nu} \stackrel{\text{def}}{=} \mu\alpha.\mathcal{K}\ \overline{\nu} \quad \text{for some } \alpha \notin \mathsf{fv}(\overline{\nu})$$

Because stamps are recursively we cannot simply compare them for syntactic equality. Instead we need a coinductive interpretation of equivalence:

$$\frac{}{\vdash \alpha \equiv_\alpha \alpha} \qquad \frac{\vdash \{\alpha_1 \coloneqq \mu\alpha_1.\mathcal{K}\ \overline{\nu}_1\}\overline{\nu}_1 \equiv_\alpha \{\alpha_2 \coloneqq \mu\alpha_2.\mathcal{K}\ \overline{\nu}_2\}\overline{\nu}_2}{\vdash \mu\alpha_1.\mathcal{K}\ \overline{\nu}_1 \equiv_\alpha \mu\alpha_2.\mathcal{K}\ \overline{\nu}_2}$$

Clearly this doesn't lend itself naturally to an algorithm, so we appeal to unification of first-order, recursive terms to decide equivalence. Due to a well-known

correspondence of recursive types to DFAs and DFA minimization,[1][2] there exists a normalization function

$$\mathsf{norm}(-) : \mathit{Stamps} \to \mathit{Stamps}$$

on stamps, which is used for the elaboration.

For the elaboration into the IL, we assume an injection from stamps (up to $\equiv_\alpha$ equivalence) to plain Haskell module names,

$$(-)^\star \quad : \quad \mathit{Stamps}/{\equiv_\alpha} \rightarrowtail \mathit{ILModNames}$$

We lift this to an injection from module types τ to plain Haskell interfaces *ftyp* which translates each stamp occurring in τ to its injected module name,

$$(-)^\star \quad : \quad \mathit{ModTypes}/{\equiv_\alpha} \rightarrowtail \mathit{ILModTyps}$$

4.2 Semantic Signatures

In the tradition of *The Definition of Standard ML* we classify package expressions with "semantic objects." Well-typed package expressions are classified with Σ and package definitions with Φ, defined below.

(module polarity)	m	::=	$+ \mid -$
(physical type contexts)	Φ	::=	$\overline{\nu{:}\tau^m}$
(logical atomic types)	$\mathcal{M}$	::=	$\nu@\tau$
(logical type contexts)	$\mathcal{L}$	::=	$\overline{p \mapsto \nu@\tau}$
(package types)	Ξ, Γ	::=	$(\Phi; \mathcal{L})$
(expression types)	Σ	::=	$(\Phi; \mathcal{L}) \mid (\Phi; \mathcal{M})$
(package environments)	Δ	::=	$\cdot \mid \Delta,\ P = \lambda\overline{\alpha}.\mathit{dexp}{:}\forall\overline{\alpha}.\Xi$

- Coercing typey things into shapey things:

$$\begin{array}{lcl}
\mathsf{shape}((\Phi; \mathcal{L})) & \stackrel{\text{def}}{=} & (\mathsf{shape}(\Phi); \mathsf{shape}(\mathcal{L})) \\
\mathsf{shape}((\Phi; \mathcal{M})) & \stackrel{\text{def}}{=} & (\mathsf{shape}(\Phi); \mathsf{shape}(\mathcal{M})) \\
\\
\mathsf{shape}(\overline{\nu{:}\tau^m}) & \stackrel{\text{def}}{=} & \overline{\nu{:}\mathsf{shape}(\tau)^m} \\
\\
\mathsf{shape}(\overline{p : \mathcal{M}}) & \stackrel{\text{def}}{=} & \overline{p : \mathsf{shape}(\mathcal{M})} \\
\\
\mathsf{shape}(\nu@\tau) & \stackrel{\text{def}}{=} & \nu@\mathsf{shape}(\tau) \\
\\
\mathsf{shape}(\langle\!|\ \overline{\mathit{dspc}}\ ;\ \overline{\mathit{espc}}\ ;\ \overline{\nu}\ |\!\rangle) & \stackrel{\text{def}}{=} & \langle\!|\overline{\mathsf{shape}(\mathit{dspc})}\ ;\ \overline{\mathit{espc}}\ ;\ \overline{\nu}|\!\rangle \\
\\
\mathsf{shape}(x :: \mathit{typ}) & \stackrel{\text{def}}{=} & x \\
\mathsf{shape}(\mathtt{data}\ T\ \mathit{kenv}) & \stackrel{\text{def}}{=} & \mathtt{data}\ T \\
\mathsf{shape}(\mathtt{data}\ T\ \mathit{kenv} = \overline{K\ \overline{\mathit{typ}}}) & \stackrel{\text{def}}{=} & \mathtt{data}\ T(\overline{K})
\end{array}$$

[1] `http://dcommon.bu.edu/xmlui/handle/2144/1800`

[2] `http://gallium.inria.fr/~fpottier/publis/gauthier-fpottier-icfp04.pdf`

- The domain of a binding signature or context, and the restriction to a given prefix:

$$\begin{array}{lcl} \mathsf{dom}(\overline{p \mapsto \nu@\tau}) & \stackrel{\text{def}}{=} & \{\overline{p}\} \\ \mathsf{dom}((\Phi;\mathcal{L})) & \stackrel{\text{def}}{=} & \mathsf{dom}(\mathcal{L}) \end{array}$$

$$\begin{array}{lcl} \mathsf{dom}_p(\mathcal{L}) & \stackrel{\text{def}}{=} & \{p' \mid p' = p.p'',\ p' \in \mathsf{dom}(\mathcal{L})\} \\ \mathsf{dom}_p((\Phi;\mathcal{L})) & \stackrel{\text{def}}{=} & \mathsf{dom}_p(\mathcal{L}) \end{array}$$

defined similarly for shapes and contexts

- Projection from a binding signature and context, forming an expression signature:

$$\begin{array}{rcl} \mathcal{L}.p & \stackrel{\text{def}}{=} & \begin{cases} \mathcal{M} & \text{if } (p \mapsto \mathcal{M}) \in \mathcal{L} \\ (p'' \mapsto \mathcal{M} \mid p' = p.p'',\ (p' \mapsto \mathcal{M}) \in \mathcal{L}) & \text{if } p \notin \mathsf{dom}(\mathcal{L}) \text{ and } \mathsf{dom}_p(\mathcal{L}) \neq \emptyset \end{cases} \\ (\Phi;\mathcal{L}).p & \stackrel{\text{def}}{=} & (\Phi;\mathcal{L}.p) \end{array}$$

defined similarly for shapes

This is defined when p is directly bound to a module signature in Ξ, or when p is a prefix of a path (or paths) in Ξ. Similarly for Γ.

- Removal of a path from a signature:

$$\begin{array}{rcl} \mathcal{L}\backslash p & \stackrel{\text{def}}{=} & (p' \mapsto \mathcal{M} \in \mathcal{L} \mid p' \neq p.p'') \\ (\Phi;\mathcal{L})\backslash p & \stackrel{\text{def}}{=} & (\Phi;\mathcal{L}\backslash p) \end{array}$$

defined similarly for shapes

- Consing a path onto an expression signature, forming a binding signature:

$$\begin{array}{lcl} \mathsf{cons}(p;\nu@\tau) & \stackrel{\text{def}}{=} & p \mapsto \nu@\tau \\ \mathsf{cons}(p;\overline{p' \mapsto \nu@\tau}) & \stackrel{\text{def}}{=} & \overline{p.p' \mapsto \nu@\tau} \end{array}$$

$$\begin{array}{lcl} \mathsf{cons}(p;(\Phi;\mathcal{M})) & \stackrel{\text{def}}{=} & (\Phi;\mathsf{cons}(p;\mathcal{M})) \\ \mathsf{cons}(p;(\Phi;\mathcal{L})) & \stackrel{\text{def}}{=} & (\Phi;\mathsf{cons}(p;\mathcal{L})) \end{array}$$

defined similarly for shapes

- Applying a renaming to a signature:

$$\begin{array}{lcl} \mathsf{rename}(\langle\cdot\rangle;\mathcal{L}) & \stackrel{\text{def}}{=} & \mathcal{L} \\ \mathsf{rename}(\langle p_s \mapsto p_t, \overline{p'_s \mapsto p'_t}\rangle;\mathcal{L}) & \stackrel{\text{def}}{=} & \mathsf{rename}(\langle\overline{p'_s \mapsto p'_t}\rangle;\mathcal{L}') \\ & & \text{where } \begin{cases} \mathsf{dom}_{p_s}(\mathcal{L}) \neq \emptyset,\ \mathsf{dom}_{p_t}(\mathcal{L}) = \emptyset \\ \mathcal{L}' = \mathcal{L}\backslash p_s, \mathsf{cons}(p_t;\mathcal{L}.p_s) \end{cases} \\ \mathsf{rename}(r;(\Phi;\mathcal{L})) & \stackrel{\text{def}}{=} & (\Phi;\mathsf{rename}(r;\mathcal{L})) \end{array}$$

defined similarly for shapes

- Creating a stamp from a context of imported module (shapes):

$$\mathsf{mkident}(M;(\tilde{\Phi};\tilde{\mathcal{L}})) \stackrel{\text{def}}{=} \mathcal{K}\ \nu_1\ \ldots\ \nu_n \quad \text{where} \quad \begin{cases} \mathcal{K} \text{ encodes } M \\ M \text{ imports } p_1,\ldots,p_n \\ \forall i \in [1..n]:\ \tilde{\mathcal{L}}(p_i) = \nu_i@\tilde{\tau}_i \end{cases}$$

5 Syntax and Semantics of Haskell

The core fragment of our external language, Haskell modules M (and signatures S), roughly looks just like the core fragment of the internal language. The only difference lies in the different sorts of names and the lack of a named module header in M.

Module names in external Haskell comprise logical path names p, whereas in internal Haskell they're module/file names f. The syntax of imports, exports, and definitions therefore differ only in their respective sorts of names.

(entity names)	χ	::=	$x \mid T \mid K$
(module names)	m	::=	$p \mid \mathsf{Local}$
(entity refs)	*eref*	::=	$\chi \mid m.\chi$
(entity imports)	*import*	::=	$\chi \mid \chi\texttt{(}\overline{\chi}\texttt{)} \mid \chi\texttt{(..)}$
(entity exports)	*export*	::=	$\mathit{eref} \mid \mathit{eref}\,\texttt{(}\overline{\chi}\texttt{)} \mid \mathit{eref}\,\texttt{(..)} \mid \texttt{module}\ m$
(import declarations)	*impdecl*	::=	$\texttt{import}\ [\texttt{qualified}]\ p\ [[\texttt{as}\ p]]\ \mathit{impspec}$
(imported module specs)	*impspec*	::=	$\emptyset \mid \texttt{(}\overline{\mathit{import}}\texttt{)}$
(imports)	*impdecls*	::=	$\overline{\mathit{impdecl}}$
(export lists)	*expdecl*	::=	$\emptyset \mid \texttt{export (}\overline{\mathit{export}}\texttt{)}$
(entity definitions)	*defs*	::=	$\texttt{data}\ T\ \mathit{kenv} = \overline{K\ \overline{\mathit{utyp}}}$
		$\mid$	$x\ [:: \mathit{utyp}] = \mathit{uexp}$
(entity declarations)	*decls*	::=	$\texttt{data}\ T\ \mathit{kenv}$
		$\mid$	$\texttt{data}\ T\ \mathit{kenv} = \overline{K\ \overline{\mathit{utyp}}}$
		$\mid$	$x :: \mathit{utyp}$
(IL entity refs)	$\check{\mathit{eref}}$	::=	$\chi \mid f.\chi$
(IL entity exports)	$\check{\mathit{export}}$	::=	$\check{\mathit{eref}} \mid \check{\mathit{eref}}\,\texttt{(}\overline{\chi}\texttt{)}$
(IL entity imports)	$\check{\mathit{import}}$	::=	$\chi \mid \chi\texttt{(}\overline{\chi}\texttt{)}$
(IL imported module specs)	$\check{\mathit{impspec}}$	::=	$\texttt{(}\overline{\check{\mathit{import}}}\texttt{)}$
(IL entity exports)	$\check{\mathit{expdecl}}$	::=	$\emptyset \mid \texttt{(}\overline{\check{\mathit{export}}}\texttt{)}$

The syntax of modules differ primarily in their arrangements of the imports, exports, and definitions, and with the additional module name in the IL. The new signature form S resembles M but leaves out the exports and swaps definitions for mere declarations.

(modules)	M	::=	$\mathit{impdecls}; \mathit{expdecl}; \mathit{defs}$
(signatures)	S	::=	$\mathit{impdecls}; \mathit{decls}$
(IL modules)	*hsmod*	::=	$\texttt{module}\ f\ \check{\mathit{expdecl}}\ \texttt{where}\ \check{\mathit{impdecls}}; \check{\mathit{defs}}$

(physical names)	$phnm$	::=	$[\nu]\chi$
(types)	typ	::=	$a \mid [\nu]T\ \overline{typ} \mid \texttt{forall}\ kenv\texttt{.}\ typ$
(kinds)	knd	::=	$\texttt{*} \mid knd\ \texttt{->}\ knd$
(kind environments)	$kenv$	::=	$\overline{a :: knd}$
(entity specs)	$dspc$	::=	$\texttt{data}\ T\ kenv$
		$\mid$	$\texttt{data}\ T\ kenv\ \texttt{=}\ \overline{K\ \overline{typ}}$
		$\mid$	$x :: typ$
(entity spec shapes)	$\tilde{dspc}$	::=	$\texttt{data}\ T$
		$\mid$	$\texttt{data}\ T(\overline{K})$
		$\mid$	x
(export specs)	$espc$	::=	$[\nu]\chi \mid [\nu]\chi(\overline{\chi})$
(entity spec sets)	$dspcs$	::=	$\overline{dspc}$
(export spec sets)	$espcs$	::=	$\overline{espc}$
(module types)	τ	::=	$\langle\!\langle\ dspcs\ \texttt{;}\ espcs\ \texttt{;}\ \overline{\nu}\ \rangle\!\rangle$
(module shapes)	$\tilde{\tau}$	::=	$\langle\!\langle \tilde{dspcs}\ \texttt{;}\ espcs\ \texttt{;}\ \overline{\nu} \rangle\!\rangle$
(signature types)	σ	::=	$\langle\!\langle\ \emptyset\ \texttt{;}\ espcs\ \texttt{;}\ \overline{\nu}\ \rangle\!\rangle$
(signature shapes)	$\tilde{\sigma}$	::=	$\langle\!\langle\ \emptyset\ \texttt{;}\ espcs\ \texttt{;}\ \overline{\nu}\ \rangle\!\rangle$
(entity environments)	$eenv$	::=	$\{\overline{eref \mapsto phnm}\}; \overline{espc}$

(augmented entity environments)	$aenv$	::=	$aenv^+ \mid aenv^-$
(augmented mod entity environments)	$aenv^+$	::=	$(\tilde{\Phi}; \nu_0; defs)$
(augmented sig entity environments)	$aenv^-$	::=	$(\tilde{\Phi}; \tilde{\tau}; decls)$
(IL augmented entity environments)	$\check{aenv}$	::=	$(fenv; f; \check{defs})$

Note: We treat export spec sets *espcs* synonymously with vectors of export specs, $\overline{espc}$. In both cases they act as finite maps from $[\nu]\chi$ to the full export specs. In this way, substitution on *espcs* is partial. For example,

$$\{\alpha := \nu\}\ ([\nu]\chi, [\alpha]\chi(\overline{\chi}'))$$

is undefined since the merged result would be invalid. *dspcs* and $\overline{dspc}$ are similarly synonymous.

Typing both levels of Haskell modules occurs in three phases: first, build an environment of entity names gotten from the imports and the locally defined entities; second, type check the definitions; and third, expose only the exported entities. We define the internal level of Haskell type checking first:

$$\frac{\begin{array}{c} fenv; f_0 \vdash^{\text{IL}}_{\text{c}} \check{impdecls}; \check{defs} \rightsquigarrow \check{eenv} \\ fenv; f_0; \check{eenv} \vdash^{\text{IL}}_{\text{c}} \check{defs} : \overline{\check{dspc}} \qquad f_0; \check{eenv} \vdash^{\text{IL}}_{\text{c}} \check{expdecl} \rightsquigarrow \overline{\check{espc}} \end{array}}{fenv \vdash^{\text{IL}}_{\text{c}} (\texttt{module}\ f_0\ \check{expdecl}\ \texttt{where}\ \check{impdecls}; \check{defs}) : \langle\!\langle\ \overline{\check{dspc}}\ \texttt{;}\ \overline{\check{espc}}\ \texttt{;}\ \mathsf{imps}(\check{impdecls})\ \rangle\!\rangle} \quad (\textsc{IlCoreMod})$$

- The first premise/judgment constructs an entity environment, $\check{eenv}$, which maps imported and defined entity names to their physical names annotated with stamps. The *fenv* is used to resolve imported module names to their imported entities, while the current module name, f_0, is used to

resolve those entities defined by the current definitions, $\check{defs}$. In GHC this is the environment constructed and used by the *Renamer* just before type checking.

- The second premise/judgment performs the actual type/kind checking of the definitions, $\check{defs}$, which produces the static specifications $\overline{\check{dspc}}$.
- The third premise/judgment parses the module's export list, $\check{expdecl}$, into a list of exported physical names, $\overline{\check{espc}}$. The *fenv* is needed for a corner case of Haskell module semantics—resolving exports of entire, imported modules.

5.1 Semantics of (External) Haskell

5.1.1 Module and Signature Judgments

For the external level of Haskell, we need four separate judgments: each combination of shaping vs. typing, and signatures vs. modules.

The external module typing judgment directly corresponds to the internal typing judgment above. Since the syntax and semantics of imports and exports is different in the EL (in that they are slightly more permissive), we need new import and export resolution judgments for the EL module typing. On the other hand, definition checking has the same semantics as in the IL, so we reuse that judgment by translating all stamps ν into file names $\nu^\star$ in the "inputs" and "outputs" of this judgment. We know the specs will all be of the form $dspc^\star$ because the only file names they can refer to—from the environment "inputs"—are themselves in the image of the $(-)^\star$ translation.

$$\frac{\begin{array}{c}\mathsf{shape}(\mathcal{L}); \nu_0 \Vdash_{\mathrm{c}} impdecls; defs \rightsquigarrow eenv \\ \Phi^\star; \nu_0^\star; \mathsf{refs}^\star_{\nu_0}(eenv) \uplus^{\mathrm{IL}}_{\mathrm{c}} \mathsf{refs}^\star_{\nu_0}(defs) : dspcs^\star \\ \nu_0; eenv \Vdash_{\mathrm{c}} expdecl \rightsquigarrow espcs \\ N = \{\mathcal{L}(\mathsf{imp}(impdecl)) \mid impdecl \in impdecls\} \\ hsmod = \mathsf{mkmod}(\mathcal{L}; \nu_0; eenv; impdecls; defs; espcs)\end{array}}{(\Phi; \mathcal{L}); \nu_0 \vdash_{\mathrm{c}} (impdecls; expdecl; defs) : \langle\!| \ dspcs \ ; \ espcs \ ; \ N \ |\!\rangle \rightsquigarrow hsmod} \ (\textsc{CoreMod})$$

The remaining judgments for typing and shaping are given below:

- Module shaping:

$$\frac{\begin{array}{c}\tilde{\mathcal{L}}; \nu_0 \Vdash_{\mathrm{c}} impdecls; defs \rightsquigarrow eenv \qquad \nu_0; eenv \Vdash_{\mathrm{c}} expdecl \rightsquigarrow espcs \\ defs \sqsubseteq \tilde{dspcs} \qquad N = \{\tilde{\mathcal{L}}(\mathsf{imp}(impdecl)) \mid impdecl \in impdecls\}\end{array}}{(\tilde{\Phi}; \tilde{\mathcal{L}}); \nu_0 \Vdash_{\mathrm{c}} (impdecls; expdecl; defs) : \langle\!| \tilde{dspcs} \ ; \ espcs \ ; \ N |\!\rangle}$$

- Signature typing:

$$\frac{\mathsf{shape}(\mathcal{L}); \tilde{\tau} \Vdash_{\mathrm{c}} impdecls; decls \rightsquigarrow eenv \qquad \Phi; eenv \vdash_{\mathrm{c}} decls \rightsquigarrow \overline{dspc} \qquad \tilde{\tau} \Vdash_{\mathrm{c}} decls \rightsquigarrow \overline{espc}}{(\Phi; \mathcal{L}); \tilde{\tau} \vdash_{\mathrm{c}} (impdecls; decls) \rightsquigarrow \langle\!| \ \emptyset \ \ ; \overline{espc} \ ; \emptyset \ |\!\rangle; \mathsf{mksigenv}(\overline{dspc}; \overline{espc})}$$

- Signature shaping:

$$\frac{\tilde{\tau} \Vdash_{\mathrm{c}} decls \rightsquigarrow \overline{espc} \qquad \overline{decl \sqsubseteq \tilde{dspc}} \qquad decls = \overline{decl}}{(\tilde{\Phi}; \tilde{\mathcal{L}}); \tilde{\tau} \Vdash_{\mathrm{c}} (impdecls; decls) \rightsquigarrow \langle\!|\emptyset \ \ ; \overline{espc} \ ; \ \emptyset|\!\rangle; \mathsf{mksigenv}(\overline{\tilde{dspc}}; \overline{espc})}$$

$$\mathsf{mksigenv}(\overline{dspc}; \overline{espc}) \stackrel{\text{def}}{=} \bigoplus_{\substack{dspc \in \overline{dspc} \\ espc \in \overline{espc} \\ dspc \sqsubseteq espc}} (\mathsf{ident}(espc) : \langle\!|\ dspc\ ;\ espc\ ;\ \emptyset\ |\!\rangle^-)$$

$$\mathsf{mksigenv}(\overline{\tilde{dspc}}; \overline{espc}) \stackrel{\text{def}}{=} \bigoplus_{\substack{\tilde{dspc} \in \overline{\tilde{dspc}} \\ espc \in \overline{espc} \\ \tilde{dspc} \sqsubseteq espc}} (\mathsf{ident}(espc) : \langle\!|\ \tilde{dspc}\ ;\ espc\ ;\ \emptyset\ |\!\rangle^-)$$

5.1.2 Checking, Import Resolution, and Export Resolution

- $\boxed{\Phi; eenv; \tilde{\tau} \vdash_{\mathsf{c}} decls \rightsquigarrow \overline{dspc}; \overline{espc}}$ Axiom!

- $\boxed{\tilde{\mathcal{L}} \Vdash_{\mathsf{c}} impdecl \rightsquigarrow eenv}$ Resolution of import declarations into an entity environment.

$$\frac{\tilde{\mathcal{L}}; p \Vdash_{\mathsf{c}} impspec \rightsquigarrow \overline{espc} \qquad eenv_{\mathsf{base}} = \mathsf{mkeenv}(\overline{espc}) \qquad eenv_{\mathsf{qual}} = \mathsf{qualify}(p['']; eenv_{\mathsf{base}})}{\tilde{\mathcal{L}} \Vdash_{\mathsf{c}} (\texttt{import}\ p\ [\texttt{as}\ p']\ impspec) \rightsquigarrow (eenv_{\mathsf{base}} \oplus eenv_{\mathsf{qual}})}$$

$$\frac{\tilde{\mathcal{L}}; p \Vdash_{\mathsf{c}} impspec \rightsquigarrow \overline{espc} \qquad eenv_{\mathsf{base}} = \mathsf{mkeenv}(\overline{espc}) \qquad eenv_{\mathsf{qual}} = \mathsf{qualify}(p['']; eenv_{\mathsf{base}})}{\tilde{\mathcal{L}} \Vdash_{\mathsf{c}} (\texttt{import qualified}\ p\ [\texttt{as}\ p']\ impspec) \rightsquigarrow eenv_{\mathsf{qual}}}$$

  - First resolve the individual entity imports for module p to get an environment $eenv_{\mathsf{base}}$.
  - $eenv_{\mathsf{qual}}$ contains only the qualified versions of entities imported from p; $eenv$ contains both qualified and unequalified versions of the entities.
  - The optional substitution δ_{as} renames the imported module name p to the aliased name p' if an alias is given in the import; the default substitution δ does nothing.
  - The final environment $eenv_{\mathsf{final}}$ applies either δ or δ_{as} to the entity references in the environment—this environment will be the full one for p, $eenv$, or only the qualified one, $eenv_{\mathsf{qual}}$, depending on the optional `qualified` keyword. The environment from the rest of the import statements is joined onto this one.

$$\mathsf{qualify}(m; eenv) \stackrel{\text{def}}{=} \{m.\chi \mapsto phnm \mid \chi \mapsto phnm \in eenv\}; \mathsf{locals}(eenv) \quad \text{where } \forall eref \in \mathsf{dom}(eenv) : \exists \chi : eref = \chi$$

- $\boxed{\tilde{\mathcal{L}}; p \Vdash_{\mathsf{c}} impspec \rightsquigarrow \overline{espc}}$ Resolution of a single imported module's specification.

$$\frac{\overline{espc} \in \tilde{\mathcal{L}}(p)}{\tilde{\mathcal{L}}; p \Vdash_{\mathsf{c}} \emptyset \rightsquigarrow \overline{espc}}$$

  - Simply gather up all the export specs in this module.

$$\frac{\overline{\tilde{\mathcal{L}}; p \Vdash_{\mathsf{c}} import \rightsquigarrow espc}}{\tilde{\mathcal{L}}; p \Vdash_{\mathsf{c}} (\overline{import}) \rightsquigarrow \bigoplus_{espc \in \overline{espc}} \{espc\}}$$

  - Resolve each imported entity into an environment, and then join them all together.

- $\boxed{\tilde{\mathcal{L}}; p \Vdash_{\mathrm{c}} \mathit{import} \rightsquigarrow \mathit{espc}}$ Resolution of a single imported module's imported entity.

$$\frac{\mathit{espc} \in \tilde{\mathcal{L}}(p) \qquad \mathit{espc} = [\nu]\chi}{\tilde{\mathcal{L}}; p \Vdash_{\mathrm{c}} \chi \rightsquigarrow \mathit{espc}} \qquad \frac{\mathit{espc} \in \tilde{\mathcal{L}}(p) \qquad \mathit{espc} \leq \mathit{espc}' = [\nu]\chi()}{\tilde{\mathcal{L}}; p \Vdash_{\mathrm{c}} \chi \rightsquigarrow \mathit{espc}'}$$

$$\frac{\mathit{espc} \in \tilde{\mathcal{L}}(p) \qquad \mathit{espc} \leq \mathit{espc}' = [\nu]\chi(\overline{\chi}')}{\tilde{\mathcal{L}}; p \Vdash_{\mathrm{c}} \chi(\overline{\chi}') \rightsquigarrow \mathit{espc}'}$$

  - First make sure that χ is actually exported by the module at path p. Then check that the imported subordinate names $\overline{\chi}'$ are among those exported.
  - Both χ and the subordinate names $\overline{\chi}'$ are added to the entity environment.

$$\frac{\mathit{espc} \in \tilde{\mathcal{L}}(p) \qquad \mathit{espc} = [\nu]\chi(\overline{\chi}')}{\tilde{\mathcal{L}}; p \Vdash_{\mathrm{c}} \chi(..) \rightsquigarrow \mathit{espc}}$$

  - Like the previous case, but adds all subordinate names to the environment.

- $\boxed{\tilde{\mathcal{L}}; \nu_0 \Vdash_{\mathrm{c}} \mathit{impdecls}; \mathit{defs} \rightsquigarrow \mathit{eenv}}$ Resolution of all the imports of a module, along with the naming of the locally defined entities.

$$\frac{\forall i \in [1..n] : \ \tilde{\mathcal{L}} \Vdash_{\mathrm{c}} \mathit{impdecl}_i \rightsquigarrow \mathit{eenv}_i \qquad \mathit{eenv}_{\mathsf{imp}} = \bigoplus_{i \in [1..n]} \mathit{eenv}_i}{\tilde{\mathcal{L}}; \nu_0 \Vdash_{\mathrm{c}} \mathit{impdecl}_1, \ldots, \mathit{impdecl}_n; \mathit{defs} \rightsquigarrow \mathit{eenv}_{\mathsf{imp}} \oplus \mathsf{mklocaleenv}(\nu_0; \mathit{defs})} \ (\textsc{EnvRes})$$

  - We don't have a path name for "this" module—only a semantic identity. Instead of allowing qualified names for the locally defined entities using this module's name, we allow a distinguished path Local.

- $\boxed{\tilde{\mathcal{L}}; \tilde{\tau} \Vdash_{\mathrm{c}} \mathit{impdecls}; \mathit{decls} \rightsquigarrow \mathit{eenv}}$ Resolution of all the imports of a signature, along with the naming of the locally defined entities.

$$\frac{\forall i \in [1..n] : \ \tilde{\mathcal{L}} \Vdash_{\mathrm{c}} \mathit{impdecl}_i \rightsquigarrow \mathit{eenv}_i \qquad \mathit{eenv}_{\mathsf{imp}} = \bigoplus_{i \in [1..n]} \mathit{eenv}_i}{\tilde{\mathcal{L}}; \tilde{\tau} \Vdash_{\mathrm{c}} \mathit{impdecl}_1, \ldots, \mathit{impdecl}_n; \mathit{decls} \rightsquigarrow \mathit{eenv}_{\mathsf{imp}} \oplus \mathsf{mklocaleenv}(\tilde{\tau}; \mathit{decls})}$$

- $\boxed{\nu_0; \mathit{eenv} \Vdash_{\mathrm{c}} \mathit{expdecl} \rightsquigarrow \overline{\mathit{espc}}}$ Resolution of module exports.

$$\frac{\overline{\mathit{phnm}} = ([\nu]\chi \mid \mathsf{Local}.\chi \mapsto [\nu]\chi \in \mathit{eenv})}{\nu_0; \mathit{eenv} \Vdash_{\mathrm{c}} \emptyset \rightsquigarrow \mathsf{filterespcs}(\mathsf{locals}(\mathit{eenv}); \overline{\mathit{phnm}})} \ (\textsc{ExpLocal})$$

$$\frac{\begin{array}{c}\forall i \in [1..n] : \ \nu_0; \mathit{eenv} \Vdash_{\mathrm{c}} \mathit{export}_i \rightsquigarrow \overline{\mathit{espc}_i} \\ \overline{\mathit{espc}} = \bigoplus_{i \in [1..n]} \overline{\mathit{espc}_i} \qquad \mathsf{nooverlap}(\overline{\mathit{espc}})\end{array}}{\nu_0; \mathit{eenv} \Vdash_{\mathrm{c}} (\mathit{export}_1, \ldots, \mathit{export}_n) \rightsquigarrow \overline{\mathit{espc}}} \ (\textsc{ExpList})$$

- $\boxed{\nu_0; \mathit{eenv} \Vdash_{\mathrm{c}} \mathit{export} \rightsquigarrow \overline{\mathit{espc}}}$ Resolution of a single entity export. *Haskell permits export of subordinate names without their owning names in the case of class methods. We don't currently allow this because every exported entity must have its own espc, but a class method does not! We could perhaps add such subordinate names as their own espcs (without subordinate lists).*

$$\frac{\mathit{eenv}(\mathit{eref}) = [\nu]\chi : [\nu]\chi}{\nu_0; \mathit{eenv} \Vdash_{\mathrm{c}} \mathit{eref} \rightsquigarrow [\nu]\chi} \ (\textsc{ExpSimple})$$

  - If the exported entity has no subordinate names, then it gets a normal export spec.

$$\frac{\mathit{eenv}(\mathit{eref}) = [\nu]\chi : [\nu]\chi(\overline{\chi}')}{\nu_0; \mathit{eenv} \Vdash_{\mathrm{c}} \mathit{eref} \rightsquigarrow [\nu]\chi()} \ (\textsc{ExpSimpleEmpty})$$

  - If the exported entity *does* have subordinate names, then it gets a subordinate export spec with zero subordinates.

$$\frac{\mathit{eenv}(\mathit{eref}) = [\nu]\chi : [\nu]\chi(\overline{\chi}', \overline{\chi}'')}{\nu_0; \mathit{eenv} \Vdash_{\mathrm{c}} \mathit{eref}\,\texttt{(}\overline{\chi}'\texttt{)} \rightsquigarrow [\nu]\chi(\overline{\chi}')} \ (\textsc{ExpSubList})$$

$$\frac{\mathit{eenv}(\mathit{eref}) = [\nu]\chi : [\nu]\chi(\overline{\chi}')}{\nu_0; \mathit{eenv} \Vdash_{\mathrm{c}} \mathit{eref}\,\texttt{(..)} \rightsquigarrow [\nu]\chi(\overline{\chi}')} \ (\textsc{ExpSubAll})$$

  - The full $\mathit{espc} \in \mathit{eenv}$ that matches this entity should contain *all* subordinate names that are available qualified or unqualified. With this definition, it's less clear that this is the case, but it should be true!

$$\frac{\begin{array}{l}\overline{\mathit{phnm}} = (\overline{[\nu]\chi \mid \chi \mapsto [\nu]\chi \in \mathit{eenv},\ m.\chi \mapsto [\nu]\chi \in \mathit{eenv}}) \\ \forall [\nu]\chi \in \overline{\mathit{phnm}} :\ \mathit{eenv}(\chi) = [\nu]\chi \ \wedge\ \mathit{eenv}(m.\chi) = [\nu]\chi\end{array}}{\nu_0; \mathit{eenv} \Vdash_{\mathrm{c}} \texttt{module}\ m \rightsquigarrow \mathsf{filterespcs}(\mathsf{locals}(\mathit{eenv}); \overline{\mathit{phnm}})} \ (\textsc{ExpModAll})$$

  - Exactly those entities which are accessible both unqualified and qualified (with m) are exported; these are the physical names in $\overline{\mathit{phnm}}$.
  - The resulting export specs are those whose name is in $\overline{\mathit{phnm}}$, and in the case of specs with subordinates, only the subordinate names in $\overline{\mathit{phnm}}$ are exported.
  - The final premise checks that every name included in this set is unambiguously identified by both the qualified and unqualified names.

$$\begin{array}{lcl}
\mathsf{filterespc}([\nu]\chi; \overline{\mathit{phnm}}) & \stackrel{\text{def}}{=} & [\nu]\chi \\
\mathsf{filterespc}([\nu]\chi(\overline{\chi}'); \overline{\mathit{phnm}}) & \stackrel{\text{def}}{=} & [\nu]\chi(\overline{\chi}'') \quad \text{where } \{\overline{[\nu]\chi''}\} = \{\overline{[\nu]\chi'}\} \cap \{\overline{\mathit{phnm}}\} \\
\mathsf{filterespcs}(\overline{\mathit{espc}}; \overline{\mathit{phnm}}) & \stackrel{\text{def}}{=} & \left\{ \mathsf{filterespc}(\mathit{espc}; \overline{\mathit{phnm}}) \;\middle|\; \begin{array}{l} \mathit{espc} \in \overline{\mathit{espc}}, \\ \mathit{phnm} \in \overline{\mathit{phnm}}, \\ \mathit{espc} \sqsubseteq \mathit{phnm} \end{array} \right\}
\end{array}$$

- $\boxed{\tilde{\tau} \Vdash_{\mathrm{c}} \mathit{decls} \rightsquigarrow \overline{\mathit{espc}}}$ Translation of signature declarations into export specifications.

$$\frac{\tilde{\tau} = \langle\!|\, \overline{\tilde{dspc}} \;;\; \overline{\mathit{espc}_0}, \overline{\mathit{espc}} \;;\; \overline{\nu'} \,|\!\rangle \qquad \overline{\mathit{decl} \sqsubseteq \mathit{espc}'_0 \leq \mathit{espc}_0}}{\tilde{\tau} \Vdash_{\mathrm{c}} \overline{\mathit{decl}} \rightsquigarrow \overline{\mathit{espc}'_0}}$$

- $\boxed{\mathit{aenv} \Vdash_{\mathrm{c}} \mathit{eenv}\ \mathsf{wf}}$ Determine whether the physical names in the range of the entity environment are indeed exported from their defining modules, or from a local definition. Also check that the locally available export specs all make sense.

$$\mathit{aenv} \Vdash_{\mathrm{c}} \mathit{eenv}\ \mathsf{wf} \quad \overset{\text{def}}{\Leftrightarrow} \quad \begin{array}{l} \forall \mathit{espc} \in \mathit{eenv} : \begin{cases} \mathit{aenv} \Vdash_{\mathrm{c}} \mathit{espc}\ \mathsf{wf} \\ \mathsf{mkphnms}(\mathit{espc}) \subseteq \mathsf{rng}(\mathit{eenv}) \end{cases} \\ \forall \mathit{eref} \mapsto \mathit{phnm} \in \mathit{eenv} : \begin{cases} \mathit{aenv} \Vdash_{\mathrm{c}} \mathit{eref} \mapsto \mathit{phnm}\ \mathsf{wf} \\ \mathit{phnm} \in \mathsf{mkphnms}(\mathsf{locals}(\mathit{eenv})) \end{cases} \end{array}$$

- $\boxed{\mathit{aenv} \Vdash_{\mathrm{c}} \mathit{eref} \mapsto \mathit{phnm}\ \mathsf{wf}}$ Determine whether the physical name in the range of the mapping is indeed exported from its defining module, or from a local definition.

$$\frac{\begin{array}{c} \mathit{eref} = \chi \text{ or } m.\chi \\ (\mathit{eref} = \mathsf{Local}.\chi) \;\Rightarrow\; \mathsf{islocal}(\mathit{aenv}; [\nu]\chi) \end{array}}{\mathit{aenv} \Vdash_{\mathrm{c}} \mathit{eref} \mapsto [\nu]\chi\ \mathsf{wf}}$$

- $\boxed{\mathit{aenv} \Vdash_{\mathrm{c}} \mathit{espc}\ \mathsf{wf}}$ $\boxed{\mathit{aenv} \Vdash_{\mathrm{c}} \overline{\mathit{espc}}\ \mathsf{wf}}$ Check that a locally available export spec is well formed with respect to a local definition and/or a spec in the context.

$$\frac{\mathit{espc}_0 = \mathsf{locmatch}(\mathit{aenv}; \mathit{espc}) \qquad \mathit{espc}_1 = \mathsf{ctxmatch}(\mathit{aenv}; \mathit{espc}) \qquad \mathit{espc}_0 \oplus \mathit{espc}_1 \leq \mathit{espc}}{\mathit{aenv} \Vdash_{\mathrm{c}} \mathit{espc}\ \mathsf{wf}}$$

$$\frac{\mathit{espc}_0 = \mathsf{locmatch}(\mathit{aenv}; \mathit{espc}) \qquad \mathsf{noctxmatch}(\mathit{aenv}; \mathit{espc}) \qquad \mathit{espc}_0 \leq \mathit{espc}}{\mathit{aenv} \Vdash_{\mathrm{c}} \mathit{espc}\ \mathsf{wf}}$$

$$\frac{\mathsf{nolocmatch}(\mathit{aenv}; \mathit{espc}) \qquad \mathit{espc}_1 = \mathsf{ctxmatch}(\mathit{aenv}; \mathit{espc}) \qquad \mathit{espc}_1 \leq \mathit{espc}}{\mathit{aenv} \Vdash_{\mathrm{c}} \mathit{espc}\ \mathsf{wf}}$$

$$\mathit{aenv} \Vdash_{\mathrm{c}} \overline{\mathit{espc}}\ \mathsf{wf} \quad \overset{\text{def}}{\Leftrightarrow} \quad \forall \mathit{espc} \in \overline{\mathit{espc}} :\ \mathit{aenv} \Vdash_{\mathrm{c}} \mathit{espc}\ \mathsf{wf}$$

5.1.3 Auxiliary Definitions

The following auxiliary definitions are needed in much of the formalism of the core language.

- $\boxed{\mathsf{islocal}(\mathit{aenv}; \mathit{espc})}$ $\boxed{\mathsf{islocal}(\mathit{aenv}; \mathit{phnm})}$ Is the given entity's identity the same as the local module? Or, in the case of signatures, is this entity

one of the locally specified ones? (This does *not* check for a matching definition/declaration; it merely looks at the identity.)

$$\begin{array}{lcl}
\mathsf{islocal}((\tilde{\Phi}; \nu_0; \mathit{defs}); \mathit{espc}) & \stackrel{\text{def}}{\Leftrightarrow} & \mathsf{ident}(\mathit{espc}) = \nu_0 \\
\mathsf{islocal}((\tilde{\Phi}; \tilde{\tau}; \mathit{decls}); \mathit{espc}) & \stackrel{\text{def}}{\Leftrightarrow} & \exists \mathit{espc}' \in \tilde{\tau} : \ \mathit{espc}' \perp \mathit{espc} \\
\mathsf{islocal}((\tilde{\Phi}; \nu_0; \mathit{defs}); [\nu]\chi) & \stackrel{\text{def}}{\Leftrightarrow} & \nu = \nu_0 \\
\mathsf{islocal}((\tilde{\Phi}; \tilde{\tau}; \mathit{decls}); [\nu]\chi) & \stackrel{\text{def}}{\Leftrightarrow} & \exists \mathit{espc}' \in \tilde{\tau} : \ \mathit{espc}' \sqsubseteq [\nu]\chi
\end{array}$$

- $\boxed{\mathsf{locmatch}(\mathit{aenv}; \mathit{espc})}$ $\boxed{\mathsf{ctxmatch}(\mathit{aenv}; \mathit{espc})}$ $\boxed{\mathsf{nolocmatch}(\mathit{aenv}; \mathit{espc})}$ $\boxed{\mathsf{noctxmatch}(\mathit{aenv}; \mathit{espc})}$
Find a matching export spec in the context or the local environment.

$$\begin{array}{lcll}
\mathsf{locmatch}(\mathit{aenv}; \mathit{espc}) & \stackrel{\text{def}}{=} & \mathit{espc}' & \text{if} \ \begin{cases} \mathsf{islocal}(\mathit{aenv}; \mathit{espc}), \\ \exists d \in \mathit{aenv}.\mathit{ds} : \ d \sqsubseteq \mathit{espc}', \\ \mathit{espc}' \perp \mathit{espc} \end{cases} \\
\mathsf{ctxmatch}(\mathit{aenv}; \mathit{espc}) & \stackrel{\text{def}}{=} & \mathit{espc}' & \text{if} \ \begin{cases} \mathit{espc}' \in \mathit{aenv}.\tilde{\Phi}(\mathsf{ident}(\mathit{espc})), \\ \mathit{espc}' \perp \mathit{espc} \end{cases} \\
\mathsf{nolocmatch}(\mathit{aenv}; \mathit{espc}) & \stackrel{\text{def}}{\Leftrightarrow} & \multicolumn{2}{l}{\mathsf{islocal}(\mathit{aenv}; \mathit{espc}) \ \Rightarrow} \\
 & & \multicolumn{2}{l}{\quad \forall \mathit{espc}', d \in \mathit{aenv}.\mathit{ds} : \ d \sqsubseteq \mathit{espc}' \Rightarrow \mathit{espc}' \not\perp \mathit{espc}} \\
\mathsf{noctxmatch}(\mathit{aenv}; \mathit{espc}) & \stackrel{\text{def}}{\Leftrightarrow} & \multicolumn{2}{l}{\forall \mathit{espc}' \in \mathit{aenv}.\tilde{\Phi}(\mathsf{ident}(\mathit{espc})) : \mathit{espc}' \not\perp \mathit{espc}}
\end{array}$$

- $\boxed{\mathsf{mkeenv}(\mathit{espc})}$ $\boxed{\mathsf{mkeenv}(\overline{\mathit{espc}})}$ Create the entity environment that maps every unqualified name given in the export spec(s).

$$\begin{array}{lcl}
\mathsf{mkeenv}([\nu]\chi) & \stackrel{\text{def}}{=} & \{\chi \mapsto [\nu]\chi\}; [\nu]\chi \\
\mathsf{mkeenv}([\nu]\chi(\overline{\chi}')) & \stackrel{\text{def}}{=} & (\{\chi \mapsto [\nu]\chi\} \cup \{\chi' \mapsto [\nu]\chi' \mid \chi' \in \overline{\chi}'\}); [\nu]\chi(\overline{\chi}') \\
\mathsf{mkeenv}(\overline{\mathit{espc}}) & \stackrel{\text{def}}{=} & \bigoplus_{\mathit{espc} \in \overline{\mathit{espc}}} \mathsf{mkeenv}(\mathit{espc})
\end{array}$$

- $\boxed{\mathsf{localespcs}(\nu_0; \mathit{defs})}$ $\boxed{\mathsf{localespcs}(\tilde{\tau}; \mathit{decls})}$ Construct the export specs of all locally defined/declared entities.

$$\begin{array}{lcl}
\mathsf{localespcs}(\nu_0; \mathit{defs}) & \stackrel{\text{def}}{=} & (\mathsf{mkespc}(\nu_0; d) \mid d \in \mathit{defs}) \\
\mathsf{localespcs}(\tilde{\tau}; \mathit{decls}) & \stackrel{\text{def}}{=} & (\mathit{espc}' \mid \mathit{espc} \in \tilde{\tau}, \ \exists d \in \mathit{decls} : \ d \sqsubseteq \mathit{espc}' \leq \mathit{espc})
\end{array}$$

- $\boxed{\mathsf{mklocaleenv}(\nu_0; \overline{\mathit{espc}})}$ $\boxed{\mathsf{mklocaleenv}(\nu_0; \mathit{defs})}$ $\boxed{\mathsf{mklocaleenv}(\tilde{\tau}; \mathit{decls})}$ Create the entity environment that maps all local entity names, qualified and unqualified.

$$\begin{array}{lcl}
\mathsf{mklocaleenv}(\overline{\mathit{espc}}) & \stackrel{\text{def}}{=} & \mathsf{mkeenv}(\overline{\mathit{espc}}) \oplus \mathsf{qualify}(\mathsf{mkeenv}(\overline{\mathit{espc}}); \mathsf{Local}) \\
\mathsf{mklocaleenv}(\nu_0; \mathit{defs}) & \stackrel{\text{def}}{=} & \mathsf{mklocaleenv}(\mathsf{localespcs}(\nu_0; \mathit{defs})) \\
\mathsf{mklocaleenv}(\tilde{\tau}; \mathit{decls}) & \stackrel{\text{def}}{=} & \mathsf{mklocaleenv}(\mathsf{localespcs}(\tilde{\tau}; \mathit{decls}))
\end{array}$$

- $\boxed{\mathsf{haslocaleenv}(\mathit{eenv}; \nu_0; \mathit{defs})}$ $\boxed{\mathsf{haslocaleenv}(\mathit{eenv}; \tilde{\tau}; \mathit{decls})}$ Specifies that *eenv* contains all the locally defined entities and, moreover, that this subset

is disjoint from the remainder of the environment.

$$\mathsf{haslocaleenv}(\mathit{eenv}; \nu_0; \mathit{defs}) \stackrel{\text{def}}{\Leftrightarrow} \exists \mathit{eenv}' : \begin{cases} \mathit{eenv} = \mathsf{mklocaleenv}(\nu_0; \mathit{defs}) \oplus \mathit{eenv}' \\ \forall \mathit{eref} \in \mathsf{dom}(\mathit{eenv}') : \ \mathit{eref} \neq \mathsf{Local}.\chi \end{cases}$$

$$\mathsf{haslocaleenv}(\mathit{eenv}; \tilde{\tau}; \mathit{decls}) \stackrel{\text{def}}{\Leftrightarrow} \exists \mathit{eenv}' : \begin{cases} \mathit{eenv} = \mathsf{mklocaleenv}(\tilde{\tau}; \mathit{decls}) \oplus \mathit{eenv}' \\ \forall \mathit{eref} \in \mathsf{dom}(\mathit{eenv}') : \ \mathit{eref} \neq \mathsf{Local}.\chi \end{cases}$$

- $\boxed{\mathsf{mkphnms}(\mathit{espc})}$ Get the set of all physical names mentioned by the *espc*.

$$\begin{array}{rcl} \mathsf{mkphnms}(\mathit{espc}) & \stackrel{\text{def}}{=} & \{[\mathsf{ident}(\mathit{espc})]\chi \mid \chi \in \mathsf{allnames}(\mathit{espc})\} \\ \mathsf{mkphnms}(\overline{\mathit{espc}}) & \stackrel{\text{def}}{=} & \bigcup_{\mathit{espc} \in \overline{\mathit{espc}}} \mathsf{mkphnms}(\mathit{espc}) \end{array}$$

- $\boxed{\mathit{eenv} \oplus \mathit{eenv}}$ $\boxed{\mathit{eenv} \perp \mathit{eenv}}$ Merge two entity environments.

$$\begin{array}{rcl} \mathit{eenv}_1 \oplus \mathit{eenv}_2 & \stackrel{\text{def}}{=} & \{\mathit{eref} \mapsto \mathit{phnm} \mid \mathit{eref} \mapsto \mathit{phnm} \in \mathit{eenv}_1 \text{ or } \mathit{eenv}_2\}; \\ & & \langle\!|\mathsf{locals}(\mathit{eenv}_1) \oplus \mathsf{locals}(\mathit{eenv}_2)|\!\rangle \\ \mathit{eenv}_1 \perp \mathit{eenv}_2 & \stackrel{\text{def}}{\Leftrightarrow} & \mathsf{locals}(\mathit{eenv}_1) \perp \mathsf{locals}(\mathit{eenv}_2) \end{array}$$

- $\boxed{\mathit{eenv}(\mathit{eref})}$ Look up the (single!) physical name referred to by *eref* in the environment.

$$\mathit{eenv}(\mathit{eref}) \stackrel{\text{def}}{=} \mathit{phnm} \quad \text{where} \quad \begin{cases} \mathit{eenv} = \{\mathit{eref} \mapsto \mathit{phnm}, \overline{\mathit{eref}' \mapsto \mathit{phnm}'}\}; \overline{\mathit{espc}}, \\ \mathit{eref} \notin \overline{\mathit{eref}'} \end{cases}$$

- $\boxed{\mathit{eenv}(\mathit{eref}) = \mathit{phnm} : \mathit{espc}}$ Look up the (single!) physical name referred to by *eref* in the environment, and also extract the relevant local *espc* from the *eenv*.

$$\mathit{eenv}(\mathit{eref}) = \mathit{phnm} : \mathit{espc} \stackrel{\text{def}}{\Leftrightarrow} \begin{cases} \mathit{espc} \in \mathsf{locals}(\mathit{eenv}), \\ \mathit{phnm} = \mathit{eenv}(\mathit{eref}), \\ \mathit{phnm} \in \mathsf{mkphnms}(\mathit{espc}) \end{cases}$$

- $\boxed{\mathsf{rng}(\mathit{eenv})}$ Get the set of all physical names that entities in the environment refer to.

$$\mathsf{rng}(\mathit{eenv}) \stackrel{\text{def}}{=} \{\mathit{phnm} \mid \mathit{eref} \mapsto \mathit{phnm} \in \mathit{eenv}\}$$

- $\boxed{\mathit{espc} \in \tilde{\mathcal{L}}(p)}$ $\boxed{\overline{\mathit{espc}} \in \tilde{\mathcal{L}}(p)}$ Checks that the given export specs are provided and visible in a logical context. The latter checks that those are *all* the export specs provided.

$$\begin{array}{rcll} \mathit{espc} \in \mathcal{L}(p) & \stackrel{\text{def}}{\Leftrightarrow} & \tilde{\tau} = \langle\!|\overline{\chi_1} \ ; \ \mathit{espc}, \overline{\mathit{espc}'} \ ; \ \overline{\nu'}|\!\rangle & \text{where } (p \mapsto \nu @ \tilde{\tau}) \in \mathcal{L} \\ \overline{\mathit{espc}} \in \mathcal{L}(p) & \stackrel{\text{def}}{\Leftrightarrow} & \tilde{\tau} = \langle\!|\overline{\chi_1} \ ; \ \overline{\mathit{espc}} \ ; \ \overline{\nu'}|\!\rangle & \text{where } (p \mapsto \nu @ \tilde{\tau}) \in \mathcal{L} \end{array}$$

- $\boxed{\mathsf{name}(\mathit{dspc})}$ $\boxed{\mathsf{name}(\mathit{espc})}$ $\boxed{\mathsf{name}(\mathit{def})}$ $\boxed{\mathsf{name}(\mathit{decl})}$ Get the name of some entity.

$$\begin{aligned} \mathsf{name}(\texttt{data}\ T\ \mathit{kenv}) &\stackrel{\text{def}}{=} T \\ \mathsf{name}(\texttt{data}\ T\ \mathit{kenv} = \overline{K\ \overline{\mathit{typ}}}) &\stackrel{\text{def}}{=} T \\ \mathsf{name}(x :: \mathit{typ}) &\stackrel{\text{def}}{=} x \end{aligned} \qquad \begin{aligned} \mathsf{name}([\nu]\chi(\overline{\chi}')) &\stackrel{\text{def}}{=} \chi \\ \mathsf{name}([\nu]\chi) &\stackrel{\text{def}}{=} \chi \end{aligned}$$

$$\begin{aligned} \mathsf{name}(\texttt{data}\ T\ \mathit{kenv} = \overline{K\ \overline{\mathit{utyp}}}) &\stackrel{\text{def}}{=} T \\ \mathsf{name}(x\ [:: \mathit{utyp}] = \mathit{uexp}) &\stackrel{\text{def}}{=} x \end{aligned} \qquad \begin{aligned} \mathsf{name}(\texttt{data}\ T\ \mathit{kenv}) &\stackrel{\text{def}}{=} T \\ \mathsf{name}(\texttt{data}\ T\ \mathit{kenv} = \overline{K\ \overline{\mathit{utyp}}}) &\stackrel{\text{def}}{=} T \\ \mathsf{name}(x :: \mathit{utyp}) &\stackrel{\text{def}}{=} x \end{aligned}$$

- $\boxed{\mathsf{names}(\mathit{dspc})}$ $\boxed{\mathsf{names}(\mathit{espc})}$ $\boxed{\mathsf{names}(\mathit{def})}$ $\boxed{\mathsf{names}(\mathit{decl})}$ Get the subordinate names of some entity, such as data constructors or class methods.

$$\begin{aligned} \mathsf{names}(\texttt{data}\ T\ \mathit{kenv}) &\stackrel{\text{def}}{=} \emptyset \\ \mathsf{names}(\texttt{data}\ T\ \mathit{kenv} = \overline{K\ \overline{\mathit{typ}}}) &\stackrel{\text{def}}{=} \{\overline{K}\} \\ \mathsf{names}(x :: \mathit{typ}) &\ \text{undefined} \end{aligned} \qquad \begin{aligned} \mathsf{names}([\nu]\chi(\overline{\chi}')) &\stackrel{\text{def}}{=} \{\overline{\chi}'\} \\ \mathsf{names}([\nu]\chi) &\ \text{undefined} \end{aligned}$$

$$\begin{aligned} \mathsf{names}(\texttt{data}\ T\ \mathit{kenv} = \overline{K\ \overline{\mathit{utyp}}}) &\stackrel{\text{def}}{=} \{\overline{K}\} \\ \mathsf{names}(x\ [:: \mathit{utyp}] = \mathit{uexp}) &\ \text{undefined} \end{aligned} \qquad \begin{aligned} \mathsf{names}(\texttt{data}\ T\ \mathit{kenv}) &\stackrel{\text{def}}{=} \emptyset \\ \mathsf{names}(\texttt{data}\ T\ \mathit{kenv} = \overline{K\ \overline{\mathit{utyp}}}) &\stackrel{\text{def}}{=} \{\overline{K}\} \\ \mathsf{names}(x :: \mathit{utyp}) &\ \text{undefined} \end{aligned}$$

- $\boxed{\mathsf{hasSubs}(\mathit{dspc})}$ $\boxed{\mathsf{hasSubs}(\mathit{espc})}$ $\boxed{\mathsf{hasSubs}(\mathit{def})}$ $\boxed{\mathsf{hasSubs}(\mathit{decl})}$ Determine whether an entity has subordinate names.

$$\begin{aligned} \mathsf{hasSubs}(\texttt{data}\ T\ \mathit{kenv}) &\stackrel{\text{def}}{=} \mathsf{true} \\ \mathsf{hasSubs}(\texttt{data}\ T\ \mathit{kenv} = \overline{K\ \overline{\mathit{typ}}}) &\stackrel{\text{def}}{=} \mathsf{true} \\ \mathsf{hasSubs}(x :: \mathit{typ}) &\stackrel{\text{def}}{=} \mathsf{false} \end{aligned} \qquad \begin{aligned} \mathsf{hasSubs}([\nu]\chi(\overline{\chi}')) &\stackrel{\text{def}}{=} \mathsf{true} \\ \mathsf{hasSubs}([\nu]\chi) &\stackrel{\text{def}}{=} \mathsf{false} \end{aligned}$$

$$\begin{aligned} \mathsf{hasSubs}(\texttt{data}\ T\ \mathit{kenv} = \overline{K\ \overline{\mathit{utyp}}}) &\stackrel{\text{def}}{=} \mathsf{true} \\ \mathsf{hasSubs}(x\ [:: \mathit{utyp}] = \mathit{uexp}) &\stackrel{\text{def}}{=} \mathsf{false} \end{aligned} \qquad \begin{aligned} \mathsf{hasSubs}(\texttt{data}\ T\ \mathit{kenv}) &\stackrel{\text{def}}{=} \mathsf{true} \\ \mathsf{hasSubs}(\texttt{data}\ T\ \mathit{kenv} = \overline{K\ \overline{\mathit{utyp}}}) &\stackrel{\text{def}}{=} \mathsf{true} \\ \mathsf{hasSubs}(x :: \mathit{utyp}) &\stackrel{\text{def}}{=} \mathsf{false} \end{aligned}$$

- $\boxed{\mathsf{allnames}(\mathit{dspc})}$ $\boxed{\mathsf{allnames}(\mathit{espc})}$ $\boxed{\mathsf{allnames}(\mathit{def})}$ $\boxed{\mathsf{allnames}(\mathit{decl})}$ Get the set containing the name and any subordinate names of the given entity.

$$\mathsf{allnames}(A) \stackrel{\text{def}}{=} \begin{cases} \{\mathsf{name}(A)\} \cup \mathsf{names}(A) & \text{if } \mathsf{hasSubs}(A) \\ \{\mathsf{name}(A)\} & \text{otherwise} \end{cases} \quad \text{for } A = \mathit{dspc}, \mathit{espc}, \mathit{def}, \mathit{decl}$$

- $\boxed{\mathsf{nooverlap}(\overline{\mathit{dspc}})}$ $\boxed{\mathsf{nooverlap}(\overline{\mathit{espc}})}$ $\boxed{\mathsf{nooverlap}(\overline{\mathit{def}})}$ $\boxed{\mathsf{nooverlap}(\overline{\mathit{decl}})}$ Get the set containing the name and any subordinate names of the given entity.

$$\mathsf{nooverlap}(A_1, \ldots, A_n) \stackrel{\text{def}}{\Leftrightarrow} \forall i, j \in [1..n] \text{ s.t. } i \neq j : \mathsf{allnames}(A_i) \# \mathsf{allnames}(A_j)$$

$$\text{for } A = \mathit{dspc}, \mathit{espc}, \mathit{def}, \mathit{decl}$$

- $\boxed{def \sqsubseteq dspc}$ $\boxed{def \sqsubseteq espc}$ $\boxed{dspc \sqsubseteq espc}$ $\boxed{espc \sqsubseteq phnm}$ This relation specifies that two core entities are syntactically similar; that a definition syntactically matches a specification, that a specification syntactically matches an export specification, and that an export specification matches a physical name. This is a very weak statement; for example, the relation includes the pair

$$x :: \texttt{Int} = \texttt{5} \;\sqsubseteq\; x :: [\nu_0]\texttt{Bool}$$

despite its obvious differences.

$\boxed{def \sqsubseteq decl}$

$$\begin{array}{rcl} \texttt{data}\ T\ kenv = \overline{K\ \overline{utyp}} & \sqsubseteq & \texttt{data}\ T\ kenv = \overline{K\ \overline{utyp}} \\ x\ [:: utyp] = uexp & \sqsubseteq & x :: utyp \end{array}$$

$\boxed{decl \sqsubseteq dspc}$

$$\begin{array}{rcl} \texttt{data}\ T\ kenv & \sqsubseteq & \texttt{data}\ T\ kenv \\ \texttt{data}\ T\ kenv = \overline{K\ \overline{utyp}} & \sqsubseteq & \texttt{data}\ T\ kenv = \overline{K\ \overline{typ}} \\ x :: utyp & \sqsubseteq & x :: typ \end{array}$$

$\boxed{dspc \sqsubseteq espc}$

$$\begin{array}{rcl} \texttt{data}\ T\ kenv & \sqsubseteq & [\nu]T() \\ \texttt{data}\ T\ kenv = \overline{K\ \overline{typ}} & \sqsubseteq & [\nu]T(\overline{K}) \\ x :: typ & \sqsubseteq & [\nu]x \end{array}$$

$\boxed{espc \sqsubseteq phnm}$

$$\begin{array}{rcl} [\nu]\chi(\overline{\chi'}) & \sqsubseteq & [\nu]\chi \\ [\nu]\chi & \sqsubseteq & [\nu]\chi \end{array}$$

$$\begin{array}{rcl} def \sqsubseteq espc & \overset{\text{def}}{\Leftrightarrow} & \exists decl, dspc :\ def \sqsubseteq decl \sqsubseteq dspc \sqsubseteq espc \\ decl \sqsubseteq espc & \overset{\text{def}}{\Leftrightarrow} & \exists dspc :\ decl \sqsubseteq dspc \sqsubseteq espc \end{array}$$

- $\boxed{\mathsf{validspc}(dspc; m)}$ States whether this specification is valid in a module type with the given polarity. This is required because not all kinds of specs are valid in both modules and signatures.

$$\mathsf{validspc}(dspc; m) \overset{\text{def}}{=} \begin{cases} \mathsf{false} & \text{if } m = + \text{ and } dspc = (\texttt{data}\ T\ kenv) \\ \mathsf{true} & \text{otherwise} \end{cases}$$

6 Semantics

The static semantics are split into two distinct phases (per-package), shaping and typing.

- Shaping characterizes structural well-formedness of a package: do the module-level imports make sense, what are the names of the code entities that each module defines, what are the module bindings in each package, and, most importantly, what are the identities of the concerning modules and the provenances of the concerning code entities. No Haskell-level type- or kind-checking is performed.
- Typing characterizes traditional well-typedness of the code entities defined in a package. Given a shape that describes the identities of everything, type-check the Haskell modules.

6.1 Shaping

$$\boxed{\tilde{\Gamma} \Vdash E \Rightarrow \tilde{\Sigma}}$$

$$\frac{p \mapsto \tilde{\mathcal{M}} \in \tilde{\Gamma}}{\tilde{\Gamma} \Vdash p \Rightarrow (\emptyset; \tilde{\mathcal{M}})}\ (\textsc{ShPath1}) \qquad \frac{p \notin \mathsf{dom}(\tilde{\Gamma}) \quad \mathsf{dom}_p(\tilde{\Gamma}) \neq \emptyset}{\tilde{\Gamma} \Vdash p \Rightarrow (\emptyset; \tilde{\Gamma}.p)}\ (\textsc{ShPath2})$$

- If p is directly bound in the context, use that. Otherwise, assemble a signature for the subnamespace.

$$\frac{\tilde{\Gamma}; \nu \Vdash_{\mathrm{c}} M : \tilde{\tau} \quad \nu = \mathsf{mkstamp}(M; \tilde{\Gamma})}{\tilde{\Gamma} \Vdash [M] \Rightarrow (\nu{:}\tilde{\tau}^{+}; \nu @ \tilde{\tau})}\ (\textsc{ShMod})$$

$$\frac{\alpha, \overline{\beta}\ \mathsf{fresh} \quad \tilde{\Gamma}; \tilde{\tau} \Vdash_{\mathrm{c}} S \rightsquigarrow \tilde{\sigma}; \tilde{\Phi}_{\tilde{\sigma}} \quad \overline{\mathit{espc}} \in \tilde{\tau} \quad \overline{\mathsf{ident}(\mathit{espc}) = \beta}}{\tilde{\Gamma} \Vdash [:: S] \Rightarrow ((\alpha{:}\tilde{\sigma}^{-}, \tilde{\Phi}_{\tilde{\sigma}}); \alpha @ \tilde{\sigma})}\ (\textsc{ShSig})$$

$$\boxed{\Delta; \tilde{\Gamma} \Vdash B \Rightarrow \tilde{\Xi}}$$

$$\frac{\tilde{\Gamma} \Vdash E \Rightarrow \tilde{\Sigma}}{\Delta; \tilde{\Gamma} \Vdash p = E \Rightarrow \mathsf{cons}(p; \tilde{\Sigma})}\ (\textsc{ShBind})$$

$$\frac{\overline{\alpha}\ \mathsf{fresh} \quad (P : \forall \overline{\alpha}.\Xi_1) \in \Delta \quad \vdash \Xi_1 \xrightarrow{t} \Xi_2 \quad \tilde{\Xi}_3 = \mathsf{shape}(\mathsf{rename}(r; \Xi_2))}{\Delta; \tilde{\Gamma} \Vdash \textbf{include}\ \mathsf{P}\ t\ r \Rightarrow \tilde{\Xi}_3}\ (\textsc{ShInc})$$

$$\boxed{\Delta \Vdash \overline{B} \Rightarrow \tilde{\Xi};\ \phi}$$

$$\frac{}{\Delta \Vdash \emptyset \Rightarrow \emptyset;\ \mathsf{id}}\ (\textsc{ShNil})$$

$$\frac{\Delta \Vdash \overline{B_1} \Rightarrow \tilde{\Xi}_1;\ \phi \quad \Delta; \tilde{\Xi}_1 \Vdash B_2 \Rightarrow \tilde{\Xi}_2 \quad \Vdash \tilde{\Xi}_1 + \tilde{\Xi}_2 \Rightarrow \tilde{\Xi};\ \phi'}{\Delta \Vdash \overline{B_1}, B_2 \Rightarrow \tilde{\Xi};\ \phi'\phi}\ (\textsc{ShSeq})$$

6.2 Typing

$$\boxed{\Gamma; \tilde{\Sigma}_{\mathsf{pkg}} \vdash E : \Sigma \rightsquigarrow \mathit{dexp}}$$

$$\frac{p \mapsto \mathcal{M} \in \Gamma}{\Gamma; \tilde{\Sigma}_{\mathsf{pkg}} \vdash p : (\emptyset; \mathcal{M}) \rightsquigarrow \{\}}\ (\textsc{TyPath1}) \qquad \frac{p \notin \mathsf{dom}(\Gamma) \quad \mathsf{dom}_p(\Gamma) \neq \emptyset}{\Gamma; \tilde{\Sigma}_{\mathsf{pkg}} \vdash p : (\emptyset; \Gamma.p) \rightsquigarrow \{\}}\ (\textsc{TyPath2})$$

$$\frac{\Gamma; \nu_0 \vdash_{\mathrm{c}} M : \tau \rightsquigarrow \mathit{hsmod}}{\Gamma; (\tilde{\Phi}; \nu_0 @ \tilde{\tau}) \vdash [M] : (\nu_0{:}\tau^{+}; \nu_0 @ \tau) \rightsquigarrow \{\nu_0^{\star} \mapsto \mathit{hsmod}\}}\ (\textsc{TyMod})$$

$$\frac{\Gamma;\tilde{\tau} \vdash_{\mathrm{c}} S \rightsquigarrow \sigma;\Phi_\sigma \qquad \nu_0{:}\sigma^- \perp \Phi_\sigma}{\Gamma;(\tilde{\Phi};\nu_0@\tilde{\tau}) \vdash [::S] : ((\nu_0{:}\sigma^- \oplus \Phi_\sigma);\nu_0@\sigma) \rightsquigarrow \mathsf{mkstubs}(\nu_0;\sigma;\Phi_\sigma)}\ (\textsc{TySig})$$

$$\boxed{\Delta;\Gamma;\tilde{\Xi}_{\mathsf{pkg}} \vdash B : \Xi \rightsquigarrow \mathit{dexp}}$$

$$\frac{\Gamma;\tilde{\Xi}_{\mathsf{pkg}}.p \vdash E : \Sigma \rightsquigarrow \mathit{dexp}}{\Delta;\Gamma;\tilde{\Xi}_{\mathsf{pkg}} \vdash p = E : \mathsf{cons}(p;\Sigma) \rightsquigarrow \mathit{dexp}}\ (\textsc{TyBind})$$

$$\frac{\begin{array}{c}\overline{\alpha}\ \mathsf{fresh} \qquad (P = \lambda\overline{\alpha}.\mathit{dexp}\ :\ \forall\overline{\alpha}.\Xi) \in \Delta \qquad \vdash \Xi \xrightarrow{t} \Xi' \\ \Xi'' = \mathsf{rename}(r;\Xi') \qquad \vdash \tilde{\Xi}_{\mathsf{pkg}} \leq_{\overline{\alpha}'} \Xi'' \rightsquigarrow \phi \qquad \overline{\alpha}' = \overline{\alpha} \cap \mathsf{dom}(\Xi'.\Phi)\end{array}}{\Delta;\Gamma;\tilde{\Xi}_{\mathsf{pkg}} \vdash \textbf{include}\ \mathsf{P}\ t\ r : \mathsf{apply}(\phi;\Xi'') \rightsquigarrow \phi^\star(\mathsf{filter}(\mathit{dexp};\Xi'))}\ (\textsc{TyInc})$$

- The first premise looks up the type of P in the context, Ξ.
- The second applies thinning to Ξ, resulting in the type Ξ' (with unnecessary stamps and bindings dropped).
- The fifth premise matches the context shape $\tilde{\Xi}_{\mathsf{pkg}}$ against the thinned and renamed shape Ξ'', producing a unifier for the variables $\overline{\alpha}$, ϕ.
- The sixth premise applies the substitution ϕ, which maps the variables $\overline{\alpha}$ into their expected identities from the context shape, to the thinned and renamed type Ξ'', and then simplifies/merges it into the final type Ξ'''. This is necessary since the substitution might result in duplicate typings in the stamp context for various stamps.
- The elaboration filters the contents of the package, *dexp*, to only those stamps still in use after thinning.

$$\boxed{\Delta;\tilde{\Xi}_{\mathsf{pkg}} \vdash \overline{B} : \Xi \rightsquigarrow \mathit{dexp}}$$

$$\frac{}{\Delta;\tilde{\Xi}_{\mathsf{pkg}} \vdash \emptyset\colon (\emptyset;\emptyset) \rightsquigarrow \{\}}\ (\textsc{TyNil})$$

$$\frac{\Delta;\tilde{\Xi}_{\mathsf{pkg}} \vdash \overline{B_1} : \Xi_1 \rightsquigarrow \mathit{dexp}_1 \qquad \Delta;\Xi_1;\tilde{\Xi}_{\mathsf{pkg}} \vdash B_2 : \Xi_2 \rightsquigarrow \mathit{dexp}_2 \qquad \Xi_1 \perp \Xi_2}{\Delta;\tilde{\Xi}_{\mathsf{pkg}} \vdash \overline{B_1}, B_2 : \Xi_1 \oplus \Xi_2 \rightsquigarrow \mathit{dexp}_1 \oplus \mathit{dexp}_2}\ (\textsc{TySeq})$$

$$\boxed{\Delta \vdash D : \forall\overline{\alpha}.\Xi \rightsquigarrow \lambda\overline{\alpha}.\mathit{dexp}}$$

$$\frac{\begin{array}{c}\Delta \Vdash \overline{B} \Rightarrow \tilde{\Xi};\ \phi \qquad \tilde{\Xi}_{\mathsf{pkg}} = \tilde{\Xi} \qquad N = \mathsf{dom}(\Xi'.\Phi) \\ \Delta;\tilde{\Xi}_{\mathsf{pkg}} \vdash \overline{B} : \Xi \rightsquigarrow \mathit{dexp} \qquad \vdash \Xi \xrightarrow{t} \Xi' \qquad \{\overline{\alpha}\} = \{\alpha \mid \alpha \in N\}\end{array}}{\Delta \vdash \textbf{package}\ P\ t\ \textbf{where}\ \overline{B} : \forall\overline{\alpha}.\Xi' \rightsquigarrow \lambda\overline{\alpha}.(\mathit{dexp}|_{N^\star})}\ (\textsc{TyPkg})$$

- The first premise computes the shape of the contents of the package (before thinning). It produces a shape, a definite substitution, and a set of constraints that must hold (under that substitution). The third premise verifies the latter condition.
- The second premise actually types the module contents.

- The fourth premise thins the type of the contents accordingly.
- The resulting type is the thinned contents' type, generalized over the free stamp variables.
- The elaboration filters that of $\overline{B}$ to only the necessary stamps (files) $\overline{\nu}$ ($\overline{\nu^\star}$, resp.), and then generalizes over the free variables (signature files).

$\boxed{\Delta \vdash \overline{D}}$

$$\frac{}{\Delta \vdash \emptyset}\ (\textsc{TyDNil}) \qquad \frac{\Delta \vdash D : \forall\overline{\alpha}.\Xi \rightsquigarrow \lambda\overline{\alpha}.\mathit{dexp} \quad \Delta, (P = \lambda\overline{\alpha}.\mathit{dexp} : \forall\overline{\alpha}.\Xi) \vdash \overline{D'}}{\Delta \vdash D, \overline{D'}}\ (\textsc{TyDCons})$$

6.3 Auxiliary definitions

Matching Judgment

$\boxed{\vdash \tilde{\Xi}_{\mathsf{pkg}} \leq_{\overline{\alpha}} \Xi \rightsquigarrow \phi}$

$$\frac{\mathsf{dom}(\phi) = \{\overline{\alpha}\} \quad \mathsf{fv}(\phi) \# \overline{\alpha} \quad \tilde{\Xi}_{\mathsf{pkg}} \leq \mathsf{apply}(\phi; \mathsf{shape}(\Xi))}{\vdash \tilde{\Xi}_{\mathsf{pkg}} \leq_{\overline{\alpha}} \Xi \rightsquigarrow \phi}$$

- This declarative judgment is implemented algorithmically like that of F-ing Modules:

$$\mathsf{lookup}_{\overline{\alpha}}(\Xi; \tilde{\Xi}) \stackrel{\text{def}}{=} \overline{\mathsf{lookup}_{\alpha}(\Xi; \tilde{\Xi})}$$

$$\mathsf{lookup}_{\alpha}((\Phi; \mathcal{L}); (\tilde{\Phi}; \tilde{\mathcal{L}})) \stackrel{\text{def}}{=} \nu \quad \text{if } \alpha{:}\tau^- \in \Phi \text{ and } p \mapsto \alpha@\tau' \in \mathcal{L} \text{ and } p \mapsto \nu@\tilde{\tau}' \in \tilde{\mathcal{L}}$$

$$\mathsf{lookup}_{\beta}((\Phi; \mathcal{L}); (\tilde{\Phi}; \tilde{\mathcal{L}})) \stackrel{\text{def}}{=} \nu_\chi \quad \text{if } \begin{pmatrix} \alpha{:}\tau^- \in \Phi \\ p \mapsto \alpha@\tau' \in \mathcal{L} \\ \chi{:}\mathit{espc} \in \tau \\ \mathsf{ident}(\mathit{espc}) = \beta \end{pmatrix} \text{ and } \begin{pmatrix} \nu{:}\tilde{\tau}^m \in \tilde{\Phi} \\ p \mapsto \nu@\tilde{\tau}' \in \tilde{\mathcal{L}} \\ \chi{:}\tilde{\mathit{espc}} \in \tilde{\tau} \\ \mathsf{ident}(\mathit{espc}) = \nu_\chi \end{pmatrix}$$

$\boxed{\mathsf{apply}(\phi; \Phi)}$

$$\begin{aligned} \mathsf{apply}(\phi; \emptyset) &\stackrel{\text{def}}{=} \emptyset \\ \mathsf{apply}(\phi; (\nu{:}\tau^m, \Phi')) &\stackrel{\text{def}}{=} (\phi\nu){:}(\phi\tau)^m \oplus \mathsf{apply}(\phi; \Phi') \end{aligned}$$

$\boxed{\Phi \vdash \mathcal{L}\ \mathsf{wf}}$ $\boxed{\tilde{\Phi} \Vdash \tilde{\mathcal{L}}\ \mathsf{wf}}$

$$\frac{\overline{\Phi \vdash \tau\ \mathsf{wf}} \quad \overline{\Phi(\nu) \leq \tau}}{\Phi \vdash \overline{p \mapsto \nu@\tau}\ \mathsf{wf}}\ (\textsc{WfLogSig}) \qquad \frac{\overline{\tilde{\Phi} \Vdash \tilde{\tau}\ \mathsf{wf}} \quad \overline{\tilde{\Phi}(\nu) \leq \tilde{\tau}}}{\tilde{\Phi} \Vdash \overline{p \mapsto \nu@\tilde{\tau}}\ \mathsf{wf}}\ (\textsc{WfShLogSig})$$

$\boxed{\Phi \vdash \Xi\ \mathsf{wf}}$ $\boxed{\tilde{\Phi} \Vdash \tilde{\Xi}\ \mathsf{wf}}$

$$\frac{\Phi \vdash \Phi'\ \mathsf{wf} \quad \Phi \oplus \Phi' \vdash \mathcal{L}'\ \mathsf{wf}}{\Phi \vdash (\Phi'; \mathcal{L}')\ \mathsf{wf}}\ (\textsc{WfSig}) \qquad \frac{\tilde{\Phi} \Vdash \tilde{\Phi}'\ \mathsf{wf} \quad \tilde{\Phi} \oplus \tilde{\Phi}' \Vdash \tilde{\mathcal{L}}'\ \mathsf{wf}}{\tilde{\Phi} \Vdash (\tilde{\Phi}'; \tilde{\mathcal{L}}')\ \mathsf{wf}}\ (\textsc{WfShSig})$$

The definitions below are defined exactly as in IL. They're listed below for posterity.

$$\boxed{\Phi \vdash \Phi\ \mathsf{wf}}\quad\boxed{\tilde{\Phi} \Vdash \tilde{\Phi}\ \mathsf{wf}}$$

$$\frac{\Phi \perp \Phi' \qquad \Phi \oplus \Phi' \vdash \Phi'\ \mathsf{specs\text{-}wf} \qquad \Phi \oplus \Phi' \vdash \Phi'\ \mathsf{exports\text{-}wf} \qquad \Phi \oplus \Phi' \vdash \Phi'\ \mathsf{deps\text{-}wf}}{\Phi \vdash \Phi'\ \mathsf{wf}}\ (\textsc{WfPhCtx})$$

$$\frac{\tilde{\Phi} \perp \tilde{\Phi}' \qquad \tilde{\Phi} \oplus \tilde{\Phi}' \Vdash \tilde{\Phi}'\ \mathsf{specs\text{-}wf} \qquad \tilde{\Phi} \oplus \tilde{\Phi}' \Vdash \tilde{\Phi}'\ \mathsf{exports\text{-}wf}}{\tilde{\Phi} \Vdash \tilde{\Phi}'\ \mathsf{wf}}\ (\textsc{WfShPhCtx})$$

$$\boxed{\Phi \vdash \Phi\ \mathsf{specs\text{-}wf}}\quad\boxed{\tilde{\Phi} \Vdash \tilde{\Phi}\ \mathsf{specs\text{-}wf}}\quad\boxed{\Phi \vdash \tau^m\ \mathsf{specs\text{-}wf}}\quad\boxed{\tilde{\Phi} \Vdash \tilde{\tau}^m\ \mathsf{specs\text{-}wf}}$$

$$\begin{array}{rcl}\Phi \vdash \Phi'\ \mathsf{specs\text{-}wf} & \stackrel{\text{def}}{\Leftrightarrow} & \forall \nu{:}\tau^m \in \Phi' :\ \Phi \vdash \tau^m\ \mathsf{specs\text{-}wf} \\ \tilde{\Phi} \Vdash \Phi'\ \mathsf{specs\text{-}wf} & \stackrel{\text{def}}{\Leftrightarrow} & \forall \nu{:}\tilde{\tau}^m \in \tilde{\Phi}' :\ \tilde{\Phi} \Vdash \tilde{\tau}^m\ \mathsf{specs\text{-}wf}\end{array}$$

$$\frac{\mathsf{nooverlap}(\overline{dspc}) \qquad \forall dspc \in \overline{dspc} :\quad \Phi \vdash dspc\ \mathsf{wf} \wedge \mathsf{validspc}(dspc; m)}{\Phi \vdash \langle\!|\ \overline{dspc}\ ;\ \overline{espc}\ ;\ \overline{\nu}\ |\!\rangle^m\ \mathsf{specs\text{-}wf}}\ (\textsc{WfSpecs})$$

$$\frac{\mathsf{nooverlap}(\overline{\tilde{dspc}}) \qquad \forall dspc \in \overline{dspc} :\quad \tilde{\Phi} \Vdash \tilde{dspc}\ \mathsf{wf} \wedge \mathsf{validspc}(\tilde{dspc}; m)}{\tilde{\Phi} \Vdash \langle\!|\overline{\tilde{dspc}}\ ;\ \overline{espc}\ ;\ \overline{\nu}|\!\rangle^m\ \mathsf{specs\text{-}wf}}\ (\textsc{WfShSpecs})$$

$$\boxed{\Phi \vdash \Phi\ \mathsf{exports\text{-}wf}}\quad\boxed{\tilde{\Phi} \Vdash \tilde{\Phi}\ \mathsf{exports\text{-}wf}}\quad\boxed{\Phi \vdash \tau\ \mathsf{exports\text{-}wf}}\quad\boxed{\tilde{\Phi} \Vdash \tilde{\tau}\ \mathsf{exports\text{-}wf}}$$

$$\begin{array}{rcl}\Phi \vdash \Phi'\ \mathsf{exports\text{-}wf} & \stackrel{\text{def}}{\Leftrightarrow} & \forall \nu{:}\tau^m \in \Phi' :\ \Phi \vdash \tau\ \mathsf{exports\text{-}wf} \\ \tilde{\Phi} \Vdash \tilde{\Phi}'\ \mathsf{exports\text{-}wf} & \stackrel{\text{def}}{\Leftrightarrow} & \forall \nu{:}\tilde{\tau}^m \in \tilde{\Phi}' :\ \tilde{\Phi} \Vdash \tilde{\tau}\ \mathsf{exports\text{-}wf}\end{array}$$

$$\frac{\mathsf{nooverlap}(\overline{espc}) \qquad \forall espc \in \overline{espc} :\ \Phi \vdash espc\ \mathsf{wf}}{\Phi \vdash \langle\!|\ \overline{dspc}\ ;\ \overline{espc}\ ;\ \overline{\nu}\ |\!\rangle\ \mathsf{exports\text{-}wf}}\ (\textsc{WfExports})$$

$$\frac{\mathsf{nooverlap}(\overline{espc}) \qquad \forall espc \in \overline{espc} :\ \tilde{\Phi} \Vdash espc\ \mathsf{wf}}{\tilde{\Phi} \Vdash \langle\!|\overline{\tilde{dspc}}\ ;\ \overline{espc}\ ;\ \overline{\nu}|\!\rangle\ \mathsf{exports\text{-}wf}}\ (\textsc{WfShExports})$$

$$\boxed{\Phi \vdash \Phi\ \mathsf{deps\text{-}wf}}\quad\boxed{\Phi \vdash \nu{:}\tau^m\ \mathsf{deps\text{-}wf}}$$

$$\Phi \vdash \Phi'\ \mathsf{deps\text{-}wf} \quad\stackrel{\text{def}}{\Leftrightarrow}\quad \forall \nu{:}\tau^m \in \Phi' :\ \Phi \vdash \nu{:}\tau^m\ \mathsf{deps\text{-}wf}$$

$$\frac{\mathsf{imps}(\tau) \subseteq \mathsf{dom}(\Phi) \qquad \mathsf{provs}(\tau) \subseteq \{\nu\} \cup \left(\bigcup_{\nu' \in \mathsf{imps}(\tau)} \mathsf{depends}_\Phi(\nu') \cup \{\nu'\}\right)}{\Phi \vdash \nu{:}\tau^+\ \mathsf{deps\text{-}wf}}\ (\textsc{WfDepsMod})$$

$$\frac{\mathsf{imps}(\tau) = \emptyset}{\Phi \vdash \nu{:}\tau^-\ \mathsf{deps\text{-}wf}}\ (\textsc{WfDepsSig})$$

$$\boxed{\Phi \vdash \tau\ \mathsf{wf}}\quad\boxed{\tilde{\Phi} \Vdash \tilde{\tau}\ \mathsf{wf}}$$

$$\frac{\begin{array}{c}\mathsf{nooverlap}(dspcs) \qquad \forall dspc \in dspcs :\ \Phi \vdash dspc\ \mathsf{wf} \\ \mathsf{nooverlap}(espcs) \qquad \forall espc \in espcs :\ \Phi \vdash espc\ \mathsf{wf} \\ \{\overline{\nu}\} \subseteq \mathsf{dom}(\Phi)\end{array}}{\Phi \vdash \langle\!|\ dspcs\ ;\ espcs\ ;\ \overline{\nu}\ |\!\rangle\ \mathsf{wf}}\ (\textsc{WfModTyp})$$

$$\frac{\mathsf{nooverlap}(\tilde{dspcs}) \quad \forall \tilde{dspc} \in \tilde{dspcs} : \tilde{\Phi} \Vdash dspc\ \mathsf{wf} \quad \mathsf{nooverlap}(espcs) \quad \forall espc \in espcs : \tilde{\Phi} \Vdash espc\ \mathsf{wf} \quad \{\overline{\nu}\} \subseteq \mathsf{dom}(\tilde{\Phi})}{\tilde{\Phi} \Vdash \langle\!| \tilde{dspcs}\ ;\ espcs\ ;\ \overline{\nu} |\!\rangle\ \mathsf{wf}}\ (\textsc{WfModSh})$$

$$\boxed{\Phi \vdash dspc\ \mathsf{wf}} \quad \boxed{\tilde{\Phi} \Vdash \tilde{dspc}\ \mathsf{wf}}$$

$$\frac{\Phi; \emptyset \vdash_{\mathrm{c}} typ :: *}{\Phi \vdash (x :: typ)\ \mathsf{wf}}\ (\textsc{WfVal}) \qquad \frac{}{\tilde{\Phi} \Vdash x\ \mathsf{wf}}\ (\textsc{WfValSh})$$

$$\frac{}{\Phi \vdash \mathtt{data}\ T\ kenv\ \mathsf{wf}}\ (\textsc{WfDataAbs}) \qquad \frac{}{\tilde{\Phi} \Vdash \mathtt{data}\ T\ \mathsf{wf}}\ (\textsc{WfDataAbsSh})$$

$$\frac{\overline{\overline{\Phi; kenv \vdash_{\mathrm{c}} typ :: *}}}{\Phi \vdash \mathtt{data}\ T\ kenv = \overline{K\ \overline{typ}}\ \mathsf{wf}}\ (\textsc{WfDataCon}) \qquad \frac{}{\tilde{\Phi} \Vdash \mathtt{data}\ T(\overline{K})\ \mathsf{wf}}\ (\textsc{WfDataConSh})$$

$$\boxed{\Phi \vdash espc\ \mathsf{wf}} \quad \boxed{\tilde{\Phi} \Vdash espc\ \mathsf{wf}}$$

$$\frac{dspc_0, espc_0 \in \Phi(\mathsf{ident}(espc)) \quad dspc_0 \sqsubseteq espc'_0 \leq espc_0 \leq espc}{\Phi \vdash espc\ \mathsf{wf}}\ (\textsc{WfESpc})$$

$$\frac{\tilde{dspc}_0, espc_0 \in \tilde{\Phi}(\mathsf{ident}(espc)) \quad \tilde{dspc}_0 \sqsubseteq espc'_0 \leq espc_0 \leq espc}{\tilde{\Phi} \Vdash espc\ \mathsf{wf}}\ (\textsc{WfShESpc})$$

$$\boxed{\Phi; kenv \vdash_{\mathrm{c}} typ :: knd}$$

$$\frac{(a :: knd) \in kenv}{\Phi; kenv \vdash_{\mathrm{c}} a :: knd}\ (\textsc{KndVar}) \qquad \frac{\Phi; kenv, kenv' \vdash_{\mathrm{c}} typ :: *}{\Phi; kenv \vdash_{\mathrm{c}} (\mathtt{forall}\ kenv'\mathtt{.}\ typ) :: *}\ (\textsc{KndAll})$$

$$\frac{\mathsf{mkknd}(\Phi(\nu)(T)) = \overline{knd}\ \mathtt{->}\ knd' \quad \overline{\Phi; kenv \vdash_{\mathrm{c}} typ :: knd}}{\Phi; kenv \vdash_{\mathrm{c}} ([\nu]T\ \overline{typ}) :: knd'}\ (\textsc{KndCtor})$$

$$\begin{aligned} \mathsf{mkknd}(\mathtt{data}\ T\ \overline{a :: knd}) &\stackrel{\mathrm{def}}{=} \overline{knd}\ \mathtt{->}\ * \\ \mathsf{mkknd}(\mathtt{data}\ T\ \overline{a :: knd} = \overline{K\ \overline{typ}}) &\stackrel{\mathrm{def}}{=} \overline{knd}\ \mathtt{->}\ * \end{aligned}$$

6.4 Thinning

Thinning filters out undesired and unnecessary modules from a package signature. A thinning spec ($\overline{p}$) says that the paths $\overline{p}$ are desired. All identities that the modules identified by $\overline{p}$ depend on must also be named by a logical path in $\overline{p}$.

$$\boxed{\vdash \Xi \xrightarrow{t} \Xi}$$

$$\frac{N = \bigcup_{p\in\overline{p}}(\{\mathcal{L}(p)\} \cup \mathsf{depends}_\Phi(\mathcal{L}(p))) \qquad \Phi' = (\nu{:}\tau^m \in \Phi \mid \nu \in N) \qquad \forall \nu{:}\tau^- \in \Phi : \begin{cases}\text{either} & \exists p \in \overline{p} : \ \mathsf{ident}(\mathcal{L}(p)) = \nu \\ \text{or} & \nu \notin N\end{cases}}{\vdash (\Phi; \mathcal{L}) \xrightarrow{(\overline{p})} (\Phi'; \mathcal{L}_{\overline{p}})}$$

- The first premise gathers up the transitive closure of necessary stamps, starting with the stamps indicated by paths $\overline{p}$, into the set N.
- The second premise filters the physical context Φ to include only those stamps in N; the result is Φ'.
- The final premise checks that every hole in the signature was either explicitly designated or is not depended-upon by the modules at the designated paths (and can thus be discarded).
- The resulting logical binding signature contains only the paths $\overline{p}$.
- Note that we require that the paths $\overline{p}$ all be atomic; *i.e.*, they cannot mention subnamespaces. We would not want this to be the case in a surface language.

$$\mathsf{depends}_\Phi(\nu) \quad \stackrel{\text{def}}{=} \quad \mathsf{depends}_{\Phi;\emptyset}(\nu) \qquad \text{where } \nu \in \mathsf{dom}(\Phi)$$

$$\begin{aligned}\mathsf{depends}_{\Phi;N}(\nu) &\stackrel{\text{def}}{=} \emptyset \qquad \text{where } \nu \in N,\ \nu \in \mathsf{dom}(\Phi) \\ \mathsf{depends}_{\Phi;N}(\nu) &\stackrel{\text{def}}{=} \left(\bigcup_{\nu'\in N'} \mathsf{depends}_{\Phi;(N,\nu)}(\nu')\right) \\ &\qquad \text{where } \nu \notin N,\ \begin{cases} N' = \mathsf{provs}(\tau) \cup \mathsf{imps}(\tau) & \text{if } \nu{:}\tau^+ \in \Phi \\ N' = \mathsf{provs}(\tau) & \text{if } \nu{:}\tau^- \in \Phi \end{cases}\end{aligned}$$

- The $\mathsf{depends}_\Phi(\nu)$ function merely computes the reachability set on the graph whose nodes are identities in Φ and with an edge (ν, ν') if

$$\begin{cases} \nu' \in \mathsf{provs}(\tau) \cup \mathsf{imps}(\tau) & \text{if } \nu{:}\tau^+ \in \Phi \\ \nu' \in \mathsf{provs}(\tau) & \text{if } \nu{:}\tau^- \in \Phi \end{cases}$$

$$\mathsf{provs}(\langle\!| \ \overline{dspc} \ ; \ \overline{espc} \ ; \ \overline{\nu} \ |\!\rangle) \quad \stackrel{\text{def}}{=} \quad \left(\bigcup_{dspc\in\overline{dspc}} \mathsf{provs}(dspc)\right) \cup \left\{\overline{\mathsf{ident}(espc)}\right\}$$

$$\begin{aligned}\mathsf{provs}(\texttt{data}\ T\ kenv) &\stackrel{\text{def}}{=} \emptyset \\ \mathsf{provs}(\texttt{data}\ T\ kenv = \overline{K\ \overline{typ}}) &\stackrel{\text{def}}{=} \textstyle\bigcup_{typ\in\overline{\overline{typ}}} \mathsf{provs}(typ) \\ \mathsf{provs}(x :: typ) &\stackrel{\text{def}}{=} \mathsf{provs}(typ)\end{aligned}$$

$$\begin{aligned}\mathsf{provs}(a) &\stackrel{\text{def}}{=} \emptyset \\ \mathsf{provs}(\texttt{forall}\ kenv.\ typ) &\stackrel{\text{def}}{=} \mathsf{provs}(typ) \\ \mathsf{provs}([\nu]T\ \overline{typ}) &\stackrel{\text{def}}{=} \{\nu\} \cup \textstyle\bigcup_{typ\in\overline{typ}} \mathsf{provs}(typ)\end{aligned}$$

$$\mathsf{imps}(\langle\!| \ \overline{dspc} \ ; \ \overline{espc} \ ; \ \overline{\nu} \ |\!\rangle) \quad \stackrel{\text{def}}{=} \quad \overline{\nu}$$

6.5 Algebras for semantic objects

6.5.1 Partial merge operations and their definedness

$$\Xi_1 \oplus \Xi_2 \stackrel{\text{def}}{=} (\Phi_1 \oplus \Phi_2; \mathcal{L}_1 \oplus \mathcal{L}_2)$$
$$\text{where} \quad \left\{ \begin{array}{lcl} \Xi_1 & = & (\Phi_1; \mathcal{L}_1) \\ \Xi_2 & = & (\Phi_2; \mathcal{L}_2) \end{array} \right.$$

$$(\Phi_1; \mathcal{L}_1) \perp (\Phi_2; \mathcal{L}_2) \stackrel{\text{def}}{\Leftrightarrow} \Phi_1 \perp \Phi_2 \ \wedge \ \mathcal{L}_1 \perp \mathcal{L}_2$$

$$\mathcal{L}_1 \oplus \mathcal{L}_2 \stackrel{\text{def}}{=} \overline{p \mapsto \nu@\tau}, \mathcal{L}'_1, \mathcal{L}'_2$$
$$\text{where} \quad \left\{ \begin{array}{lcl} \mathcal{L}_1 & = & \overline{p \mapsto \nu@\tau_1}, \mathcal{L}'_1 \\ \mathcal{L}_2 & = & \overline{p \mapsto \nu@\tau_2}, \mathcal{L}'_2 \\ \overline{\tau} & = & \overline{\tau_1 \oplus \tau_2} \\ \mathsf{dom}(\mathcal{L}'_1) \# \mathsf{dom}(\mathcal{L}'_2) \end{array} \right.$$

$$\mathcal{L}_1 \perp \mathcal{L}_2 \stackrel{\text{def}}{\Leftrightarrow} \forall p, \nu, \tau_1, \tau_2 : \left(\begin{array}{l} p \mapsto \nu@\tau_1 \in \mathcal{L}_1 \\ p \mapsto \nu@\tau_2 \in \mathcal{L}_2 \end{array} \right) \Rightarrow \tau_1 \perp \tau_2$$

$$\Phi_1 \oplus \Phi_2 \stackrel{\text{def}}{=} \overline{\nu{:}\tau^m}, \Phi'_1, \Phi'_2$$
$$\text{where} \quad \left\{ \begin{array}{lcl} \Phi_1 & = & \overline{\nu{:}\tau_1^{m_1}}, \Phi'_1 \\ \Phi_2 & = & \overline{\nu{:}\tau_2^{m_2}}, \Phi'_2 \\ \overline{\tau^m} & = & \overline{\tau_1^{m_1} \oplus \tau_2^{m_2}} \\ \mathsf{dom}(\Phi'_1) \# \mathsf{dom}(\Phi'_2) \end{array} \right.$$

$$\Phi_1 \perp \Phi_2 \stackrel{\text{def}}{\Leftrightarrow} \forall \nu, \tau_1, \tau_2, m_1, m_2 : \left(\begin{array}{l} \nu{:}{\tau_1}^{m_1} \in \Phi_1 \\ \nu{:}{\tau_2}^{m_2} \in \Phi_2 \end{array} \right) \Rightarrow \tau_1^{m_1} \perp \tau_2^{m_2}$$

$$\begin{array}{rcll} \tau_1^- \oplus \tau_2^- & \stackrel{\text{def}}{=} & (\tau_1 \oplus \tau_2)^- & \\ \tau_1^- \oplus \tau_2^+ & \stackrel{\text{def}}{=} & \tau_2^+ & \text{if } \tau_2 \leq \tau_1 \\ \tau_1^+ \oplus \tau_2^- & \stackrel{\text{def}}{=} & \tau_1^+ & \text{if } \tau_1 \leq \tau_2 \\ \tau_1^+ \oplus \tau_2^+ & \stackrel{\text{def}}{=} & \tau_1^+ & \text{if } \tau_1 = \tau_2 \end{array}$$

$$\tau_1^{m_1} \perp \tau_2^{m_2} \stackrel{\text{def}}{\Leftrightarrow} \left(\begin{array}{lcl} m_1, m_2 = -, - & \Rightarrow & \tau_1 \perp \tau_2 \\ m_1, m_2 = -, + & \Rightarrow & \tau_2 \leq \tau_1 \\ m_1, m_2 = +, - & \Rightarrow & \tau_1 \leq \tau_2 \\ m_1, m_2 = +, + & \Rightarrow & \tau_1 = \tau_2 \end{array} \right)$$

$$\begin{array}{rcl} - \oplus + & \stackrel{\text{def}}{=} & + \\ + \oplus - & \stackrel{\text{def}}{=} & + \\ m \oplus m & \stackrel{\text{def}}{=} & m \end{array}$$

$$\langle\!| \ \overline{dspc_1} \ ; \ \overline{espc_1} \ ; \ \overline{\nu_1} \ |\!\rangle \oplus \langle\!| \ \overline{dspc_2} \ ; \ \overline{espc_2} \ ; \ \overline{\nu_2} \ |\!\rangle \stackrel{\text{def}}{=} \langle\!| \ \overline{dspc} \ ; \ \overline{espc} \ ; \ \overline{\nu} \ |\!\rangle$$
$$\text{where} \quad \left\{ \begin{array}{l} \overline{dspc} = \overline{dspc_1} \oplus \overline{dspc_2} \\ \overline{espc} = \overline{espc_1} \oplus \overline{espc_2} \\ \mathsf{nooverlap}(\overline{dspc}) \ \wedge \ \mathsf{nooverlap}(\overline{espc}) \\ \{\overline{\nu}\} = \{\overline{\nu_1}\} \cup \{\overline{\nu_2}\} = \{\overline{\nu_1}\} \text{ or } \{\overline{\nu_2}\} \end{array} \right.$$

$$\langle\!|\ \overline{dspc_1}\ ;\ \overline{espc_1}\ ;\ \overline{\nu_1}\ |\!\rangle \perp \langle\!|\ \overline{dspc_2}\ ;\ \overline{espc_2}\ ;\ \overline{\nu_2}\ |\!\rangle \quad \overset{\text{def}}{\Leftrightarrow} \quad \begin{cases} \overline{dspc_1} \perp \overline{dspc_2} \\ \overline{espc_1} \perp \overline{espc_2} \\ \mathsf{nooverlap}(\overline{dspc_1} \oplus \overline{dspc_2}) \\ \mathsf{nooverlap}(\overline{espc_1} \oplus \overline{espc_2}) \\ \{\overline{\nu_1}\} \cup \{\overline{\nu_2}\} = \{\overline{\nu_1}\} \text{ or } \{\overline{\nu_2}\} \end{cases}$$

$$\overline{dspc_1} \oplus \overline{dspc_2} \overset{\text{def}}{=} \overline{dspc}, \overline{dspc_1''}, \overline{dspc_2''} \quad \text{where} \quad \begin{cases} \overline{dspc_1} = \overline{dspc_1'}, \overline{dspc_1''} \\ \overline{dspc_2} = \overline{dspc_2'}, \overline{dspc_2''} \\ \overline{dspc} = \overline{dspc_1' \oplus dspc_2'} \\ \overline{\mathsf{name}(dspc_1'')} \# \overline{\mathsf{name}(dspc_2'')} \end{cases}$$

$$\overline{espc_1} \oplus \overline{espc_2} \overset{\text{def}}{=} \overline{espc}, \overline{espc_1''}, \overline{espc_2''} \quad \text{where} \quad \begin{cases} \overline{espc_1} = \overline{espc_1'}, \overline{espc_1''} \\ \overline{espc_2} = \overline{espc_2'}, \overline{espc_2''} \\ \overline{espc} = \overline{espc_1' \oplus espc_2'} \\ \overline{\mathsf{mkphnm}(espc_1'')} \# \overline{\mathsf{mkphnm}(espc_2'')} \end{cases}$$

$$\overline{dspc_1} \perp \overline{dspc_2} \quad \overset{\text{def}}{\Leftrightarrow} \quad \forall dspc_1, dspc_2 : \left(\begin{array}{l} \mathsf{name}(dspc_1) = \mathsf{name}(dspc_2) \\ dspc_1 \in \overline{dspc_1} \ \wedge \ dspc_2 \in \overline{dspc_2} \end{array} \right) \Rightarrow dspc_1 \perp dspc_2$$

$$\overline{espc_1} \perp \overline{espc_2} \quad \overset{\text{def}}{\Leftrightarrow} \quad \forall espc_1, espc_2 : \left(\begin{array}{l} \mathsf{mkphnm}(espc_1) = \mathsf{mkphnm}(espc_2) \\ espc_1 \in \overline{espc_1} \ \wedge \ espc_2 \in \overline{espc_2} \end{array} \right) \Rightarrow espc_1 \perp espc_2$$

$$\begin{array}{rcl} (\mathtt{data}\ T\ kenv) \oplus (\mathtt{data}\ T\ kenv = \overline{K\ \overline{typ}}) & \overset{\text{def}}{=} & \mathtt{data}\ T\ kenv = \overline{K\ \overline{typ}} \\ (\mathtt{data}\ T\ kenv = \overline{K\ \overline{typ}}) \oplus (\mathtt{data}\ T\ kenv) & \overset{\text{def}}{=} & \mathtt{data}\ T\ kenv = \overline{K\ \overline{typ}} \\ dspc \oplus dspc & \overset{\text{def}}{=} & dspc \end{array}$$

$$dspc_1 \perp dspc_2 \quad \overset{\text{def}}{\Leftrightarrow} \quad dspc_1 = dspc_2 \vee \left(\begin{array}{l} dspc_1, dspc_2 = (\mathtt{data}\ T\ kenv = \overline{K\ \overline{typ}}), (\mathtt{data}\ T\ kenv) \vee \\ dspc_1, dspc_2 = (\mathtt{data}\ T\ kenv), (\mathtt{data}\ T\ kenv = \overline{K\ \overline{typ}}) \end{array} \right)$$

$$\begin{array}{rcll} [\nu]\chi(\overline{\chi}_1) \oplus [\nu]\chi(\overline{\chi}_2) & \overset{\text{def}}{=} & [\nu]\chi(\overline{\chi}') & \text{where } \{\overline{\chi}'\} = \{\overline{\chi}_1\} \cup \{\overline{\chi}_2\} \\ [\nu]\chi \oplus [\nu]\chi & \overset{\text{def}}{=} & [\nu]\chi & \end{array}$$

$$espc_1 \perp espc_2 \quad \overset{\text{def}}{\Leftrightarrow} \quad \left(\begin{array}{l} \mathsf{ident}(espc_1) = \mathsf{ident}(espc_2) \\ \mathsf{name}(espc_1) = \mathsf{name}(espc_2) \\ \mathsf{hasSubs}(espc_1) = \mathsf{hasSubs}(espc_2) \end{array} \right)$$

6.5.2 Partial order (induced from merge)

$$\langle\!|\ \overline{dspc_1}, \overline{dspc_1'}\ ;\ \overline{espc_1}, \overline{espc_1'}\ ;\ \overline{\nu_1}\ |\!\rangle \leq \langle\!|\ \overline{dspc_2}\ ;\ \overline{espc_2}\ ;\ \overline{\nu_2}\ |\!\rangle$$
$$\overset{\text{def}}{\Leftrightarrow}$$
$$\overline{dspc_1 \leq dspc_2} \ \wedge \ \overline{espc_1 \leq espc_2} \ \wedge \ \{\overline{\nu_1}\} \supseteq \{\overline{\nu_2}\}$$

$$\begin{array}{rcl} (\mathtt{data}\ T\ kenv = \overline{K\ \overline{typ}}) \leq (\mathtt{data}\ T\ kenv) & \overset{\text{def}}{\Leftrightarrow} & \text{always} \\ dspc \leq dspc & \overset{\text{def}}{\Leftrightarrow} & \text{always} \end{array}$$

$$\begin{array}{rcl} [\nu]\chi(\overline{\chi}_1) \leq [\nu]\chi(\overline{\chi}_2) & \overset{\text{def}}{\Leftrightarrow} & \{\overline{\chi}_1\} \supseteq \{\overline{\chi}_2\} \\ [\nu]\chi \leq [\nu]\chi & \overset{\text{def}}{\Leftrightarrow} & \text{always} \end{array}$$

6.6 Unification for semantic object shapes

- Merging of two composite signature shapes. The merging of logical contexts produces a substitution ϕ. The physical contexts are then merged after applying ϕ. $\boxed{\Vdash \tilde{\Xi} + \tilde{\Xi} \Rightarrow \tilde{\Xi};\ \phi}$

$$\frac{\Vdash \tilde{\mathcal{L}}_1 + \tilde{\mathcal{L}}_2 \Rightarrow \tilde{\mathcal{L}};\ \phi \qquad \tilde{\Phi} = \mathsf{apply}(\phi; \tilde{\Phi}_1) \oplus \mathsf{apply}(\phi; \tilde{\Phi}_2)}{\Vdash (\tilde{\Phi}_1; \tilde{\mathcal{L}}_1) + (\tilde{\Phi}_2; \tilde{\mathcal{L}}_2) \Rightarrow (\tilde{\Phi}; \tilde{\mathcal{L}});\ \phi}$$

- Merging of two logical contexts. $\boxed{\Vdash \tilde{\mathcal{L}} + \tilde{\mathcal{L}} \Rightarrow \tilde{\mathcal{L}};\ \phi}$

$$\frac{\mathsf{dom}(\tilde{\mathcal{L}}_1) \# \mathsf{dom}(\tilde{\mathcal{L}}_2)}{\Vdash \tilde{\mathcal{L}}_1 + \tilde{\mathcal{L}}_2 \Rightarrow \tilde{\mathcal{L}}_1, \tilde{\mathcal{L}}_2;\ \mathsf{id}}$$

$$\frac{\mathsf{unify}(\nu_1 \doteq \nu_2) \rightsquigarrow \phi \qquad \Vdash \phi\tilde{\tau}_1 + \phi\tilde{\tau}_2 \Rightarrow \tilde{\tau};\ \phi' \qquad \Vdash \phi'\phi\tilde{\mathcal{L}}_1 + \phi'\phi\tilde{\mathcal{L}}_2 \Rightarrow \tilde{\mathcal{L}};\ \phi''}{\Vdash (p \mapsto \nu_1 @ \tilde{\tau}_1), \tilde{\mathcal{L}}_1 + (p \mapsto \nu_2 @ \tilde{\tau}_2), \tilde{\mathcal{L}}_2 \Rightarrow (p \mapsto (\phi''\phi'\nu_1) @ (\phi''\tilde{\tau})), \tilde{\mathcal{L}};\ \phi''\phi'\phi}$$

- Merging of two module shapes. The unification occurs in the provenances of the export specs. $\boxed{\Vdash \tilde{\tau} + \tilde{\tau} \Rightarrow \tilde{\tau};\ \phi}$

$$\frac{\begin{array}{c}\widetilde{d\mathit{spc}} = \widetilde{d\mathit{spc}_1} \oplus \widetilde{d\mathit{spc}_2} \qquad \overline{\mathsf{name}(\widetilde{d\mathit{spc}_1'})} \# \overline{\mathsf{name}(\widetilde{d\mathit{spc}_2'})} \qquad \overline{\mathsf{name}(\mathit{espc}_1')} \# \overline{\mathsf{name}(\mathit{espc}_2')} \\ \Vdash \overline{\mathit{espc}_1} + \overline{\mathit{espc}_2} \Rightarrow \overline{\mathit{espc}};\ \phi \qquad \{\overline{\nu}\} = \{\overline{\phi\nu_1}\} \cup \{\overline{\phi\nu_2}\} \\ \tilde{\tau}_1 = \langle\!\langle \overline{\widetilde{d\mathit{spc}_1}}, \overline{\widetilde{d\mathit{spc}_1'}}\ ;\ \overline{\mathit{espc}_1}, \overline{\mathit{espc}_1'}\ ;\ \overline{\nu_1} \rangle\!\rangle \\ \tilde{\tau}_2 = \langle\!\langle \overline{\widetilde{d\mathit{spc}_2}}, \overline{\widetilde{d\mathit{spc}_2'}}\ ;\ \overline{\mathit{espc}_2}, \overline{\mathit{espc}_2'}\ ;\ \overline{\nu_2} \rangle\!\rangle \end{array}}{\Vdash \tilde{\tau}_1 + \tilde{\tau}_2 \Rightarrow \langle\!\langle \overline{\widetilde{d\mathit{spc}}}\ ;\ \overline{\mathit{espc}_1'}, \overline{\mathit{espc}}, \overline{\mathit{espc}_2'}\ ;\ \overline{\nu} \rangle\!\rangle;\ \phi}$$

$\boxed{\Vdash \overline{\mathit{espc}} + \overline{\mathit{espc}} \Rightarrow \overline{\mathit{espc}};\ \phi}$

$$\frac{}{\Vdash \emptyset + \emptyset \Rightarrow \emptyset;\ \mathsf{id}} \qquad \frac{\Vdash \mathit{espc}_1 + \mathit{espc}_2 \Rightarrow \mathit{espc};\ \phi \qquad \Vdash \overline{\phi\mathit{espc}_1'} + \overline{\phi\mathit{espc}_2'} \Rightarrow \overline{\mathit{espc}'};\ \phi'}{\Vdash (\mathit{espc}_1, \overline{\mathit{espc}_1'}) + (\mathit{espc}_2, \overline{\mathit{espc}_2'}) \Rightarrow (\mathit{espc}, \overline{\mathit{espc}'});\ \phi'\phi}$$

$\boxed{\Vdash \mathit{espc} + \mathit{espc} \Rightarrow \mathit{espc};\ \phi}$

$$\frac{\mathsf{unify}(\nu_1 \doteq \nu_2) \rightsquigarrow \phi}{\Vdash [\nu_1]\chi + [\nu_2]\chi \Rightarrow [\phi\nu_1]\chi;\ \phi} \qquad \frac{\mathsf{unify}(\nu_1 \doteq \nu_2) \rightsquigarrow \phi \qquad \{\overline{\chi}'\} = \{\overline{\chi}_1\} \cup \{\overline{\chi}_2\}}{\Vdash [\nu_1]\chi(\overline{\chi}_1) + [\nu_2]\chi(\overline{\chi}_2) \Rightarrow [\phi\nu_1]\chi(\overline{\chi}');\ \phi}$$

7 Internal Language

7.1 Syntax

(source names)	f, g	$\in$	$\mathit{ILModNames}$
(source expressions)	fexp	::=	$\mathit{hsmod} \mid -$
(source types)	ftyp	::=	$\langle\!\langle\ \overline{\check{d\mathit{spc}}}\ ;\ \overline{\check{e\mathit{spc}}}\ ;\ \overline{f}\ \rangle\!\rangle$
(typed source expressions)	tfexp	::=	$\mathit{fexp} : \mathit{ftyp}$
(directory expressions)	dexp	::=	$\{\overline{f \mapsto \mathit{tfexp}}\}$
(file environments)	fenv	::=	$\overline{f{:}\mathit{ftyp}^m}$
(file name replacements)	ϕ	::=	$\{\overline{f \coloneqq g}\}$

- EL physical module contexts are isomorphic to file environments whose file names all lie in the range of the translation function $(-)^\star$. Similarly with EL module types and source types, and EL identity substitutions and file name replacements. We implicitly rely on this isomorphism by not duplicating definitions which are "parametric" in the identity/file names, like all semantic object well-formedness judgments and algebraic definitions.

- Coercing a directory expression into a file environment:

$$\begin{array}{rcl} \mathsf{mkfenv}(\{f \mapsto \mathit{fexp} : \mathit{ftyp}, \overline{f' \mapsto \mathit{tfexp}'}\}) & \stackrel{\text{def}}{=} & \mathsf{mkfenv}(\{\overline{f' \mapsto \mathit{tfexp}'}\}), f{:}\mathit{ftyp} \\ \mathsf{mkfenv}(\{\}) & \stackrel{\text{def}}{=} & \emptyset \end{array}$$

- File name replacement in various syntactic forms. Replacement on file names, source modules, file types, and file expressions is straightforward as there are no binding sites and thus no concern for capture. Replacement on directory expressions is trickier since we must recursively merge each substituted file into the rest. (Consider what happens when a *dexp* containing two files/file names get substituted with the same name.)

$$\begin{array}{rcll} \{f \coloneqq f', \overline{g \coloneqq g'}\}f & \stackrel{\text{def}}{=} & f' & \text{where } f \notin \overline{g} \\ \{\overline{g \coloneqq g'}\}f & \stackrel{\text{def}}{=} & f & \text{where } f \notin \overline{g} \end{array}$$

$$\begin{array}{rcl} \phi\{\} & \stackrel{\text{def}}{=} & \{\} \\ \phi\{f \mapsto \mathit{tfexp}, \overline{f' \mapsto \mathit{tfexp}'}\} & \stackrel{\text{def}}{=} & \{\phi f \mapsto \phi \mathit{tfexp}\} \oplus \phi\{\overline{f' \mapsto \mathit{tfexp}'}\} \end{array}$$

- Note that we require this operation to respect translation:

$$\phi^\star(\nu^\star) \stackrel{\text{def}}{=} (\phi(\nu))^\star$$

- Merging of (typed) file expressions, file types, entity specs, export specs, directory expressions, directory types, and file environments.

$$\begin{array}{rcrcll} (\mathit{hsmod}_1 : \mathit{ftyp}_1) & \oplus & (\mathit{hsmod} : \mathit{ftyp}_2) & \stackrel{\text{def}}{=} & \mathit{hsmod}_1 : \mathit{ftyp}_1 & \text{where } \mathit{ftyp}_1 = \mathit{ftyp}_2 \\ (\mathit{hsmod}_1 : \mathit{ftyp}_1) & \oplus & (- : \mathit{ftyp}_2) & \stackrel{\text{def}}{=} & \mathit{hsmod}_1 : \mathit{ftyp}_1 & \text{where } \mathit{ftyp}_1 \leq \mathit{ftyp}_2 \\ (- : \mathit{ftyp}_1) & \oplus & (\mathit{hsmod}_2 : \mathit{ftyp}_2) & \stackrel{\text{def}}{=} & \mathit{hsmod}_2 : \mathit{ftyp}_2 & \text{where } \mathit{ftyp}_2 \leq \mathit{ftyp}_1 \\ (- : \mathit{ftyp}_1) & \oplus & (- : \mathit{ftyp}_2) & \stackrel{\text{def}}{=} & - : (\mathit{ftyp}_1 \oplus \mathit{ftyp}_2) & \text{where } \mathit{ftyp}_1 \perp \mathit{ftyp}_2 \end{array}$$

$$\mathit{dexp}_1 \oplus \mathit{dexp}_2 \stackrel{\text{def}}{=} \{\overline{f \mapsto (\mathit{tfexp}_1 \oplus \mathit{tfexp}_2)}, \overline{f_1 \mapsto \mathit{tfexp}'_1}, \overline{f_2 \mapsto \mathit{tfexp}'_2}\}$$

$$\text{where} \quad \begin{cases} \mathit{dexp}_1 = \overline{f \mapsto \mathit{tfexp}_1}, \overline{f_1 \mapsto \mathit{tfexp}'_1} \\ \mathit{dexp}_2 = \overline{f \mapsto \mathit{tfexp}_2}, \overline{f_2 \mapsto \mathit{tfexp}'_2} \\ \overline{f_1} \# \overline{f_2} \end{cases}$$

- $\boxed{\mathit{dexp}|_F}$ Filtering a directory expression.

$$\mathit{dexp}|_F \stackrel{\text{def}}{=} \{f \mapsto \mathit{tfexp} \in \mathit{dexp} \mid f \in F\}$$

- $\boxed{\mathsf{alias}(\mathit{impdecl})}$ $\boxed{\mathsf{aliases}(\mathit{impdecls})}$ $\boxed{\mathsf{aliases}(\mathit{hsmod})}$ The name or names of module name aliases on imports.

$$\begin{array}{lcl} \mathsf{alias}(\texttt{import}\ [\texttt{qualified}]\ f\ \texttt{as}\ f') & \stackrel{\text{def}}{=} & f' \\ \mathsf{aliases}(\mathit{impdecls}) & \stackrel{\text{def}}{=} & \{\mathsf{alias}(\mathit{impdecl}) \mid \mathit{impdecl} \in \mathit{impdecls}\} \\ \mathsf{aliases}(\texttt{module}\ f_0\ \mathit{expdecl}\ \texttt{where}\ \mathit{impdecls}; \mathit{defs}) & \stackrel{\text{def}}{=} & \mathsf{aliases}(\mathit{impdecls}) \end{array}$$

- $\boxed{\mathsf{imp}(\mathit{impdecl})}$ $\boxed{\mathsf{imps}(\mathit{impdecls})}$ $\boxed{\mathsf{imps}(\mathit{hsmod})}$ The name or names of imported modules.

$$\begin{array}{lcl} \mathsf{imp}(\texttt{import}\ [\texttt{qualified}]\ f\ \texttt{as}\ f') & \stackrel{\text{def}}{=} & f \\ \mathsf{imps}(\mathit{impdecls}) & \stackrel{\text{def}}{=} & \{\mathsf{imp}(\mathit{impdecl}) \mid \mathit{impdecl} \in \mathit{impdecls}\} \\ \mathsf{imps}(\texttt{module}\ f_0\ \mathit{expdecl}\ \texttt{where}\ \mathit{impdecls}; \mathit{defs}) & \stackrel{\text{def}}{=} & \mathsf{imps}(\mathit{impdecls}) \end{array}$$

7.2 Semantics

- $\boxed{\mathit{fenv} \vdash \mathit{dexp}}$ Directory expression typing. Truncate a directory expression into an environment by stripping away module source files, and then type check each source file in that environment. Like doing GHC type checking if all modules already had a binary interface file.

  The *dexp* resembles a recursive module and its constituent *ftyp*s its forward declaration, which might itself be a recursively dependent signature (r.d.s.). We proceed as in RMC by checking the well-formedness of the forward declaration and then checking the module contents with the forward declaration as an assumption.

$$\frac{\mathit{fenv}' = \mathsf{mkfenv}(\mathit{dexp}) \qquad \mathit{fenv} \vdash \mathit{fenv}'\ \mathsf{wf} \qquad \forall f \in \mathsf{dom}(\mathit{dexp})\ :\ \mathit{fenv} \oplus \mathit{fenv}'; f \vdash \mathit{dexp}(f)}{\mathit{fenv} \vdash \mathit{dexp}}\ (\textsc{IlTypDexp})$$

- $\boxed{\mathit{fenv}; f_0 \vdash \mathit{tfexp}}$ Typed file expression typing. For source files, resolve the imports to particular files and types in the environment, and then type check the file. The inferred type of the file (from GHC) must check against the given type.

$$\frac{\mathsf{name}(\mathit{hsmod}) = f_0 \qquad \mathit{fenv} \vdash^{\text{IL}}_{\text{c}} \mathit{hsmod} : \mathit{ftyp}}{\mathit{fenv}; f_0 \vdash (\mathit{hsmod} : \mathit{ftyp})}\ (\textsc{IlTypHsmod})$$

$$\frac{\mathit{fenv} \vdash \mathit{ftyp}\ \mathsf{wf}}{\mathit{fenv}; f_0 \vdash (- : \mathit{ftyp})}\ (\textsc{IlTypStub})$$

7.3 Core Language

- $\boxed{\mathit{fenv} \vdash^{\text{IL}}_{\text{c}} \check{\mathit{impdecl}} \rightsquigarrow \check{\mathit{eenv}}}$ Resolution of an import declaration into an entity environment.

$$\frac{\mathit{fenv}; f \vdash^{\text{IL}}_{\text{c}} \check{\mathit{impspec}} \rightsquigarrow \overline{\check{\mathit{espc}}} \qquad \check{\mathit{eenv}}_{\mathsf{base}} = \mathsf{mkeenv}(\overline{\check{\mathit{espc}}}) \qquad \check{\mathit{eenv}}_{\mathsf{qual}} = \mathsf{qualify}(f'; \check{\mathit{eenv}}_{\mathsf{base}})}{\mathit{fenv} \vdash^{\text{IL}}_{\text{c}} (\texttt{import}\ f\ \texttt{as}\ f'\ \check{\mathit{impspec}}) \rightsquigarrow (\check{\mathit{eenv}}_{\mathsf{base}} \oplus \check{\mathit{eenv}}_{\mathsf{qual}})}$$

$$\frac{\begin{array}{c}\mathit{fenv}; f \vdash^{\text{IL}}_{\text{c}} \check{\mathit{impspec}} \rightsquigarrow \overline{\check{\mathit{espc}}} \\ \check{\mathit{eenv}}_{\mathsf{base}} = \mathsf{mkeenv}(\overline{\check{\mathit{espc}}}) \qquad \check{\mathit{eenv}}_{\mathsf{qual}} = \mathsf{qualify}(f'; \check{\mathit{eenv}}_{\mathsf{base}})\end{array}}{\mathit{fenv} \vdash^{\text{IL}}_{\text{c}} (\texttt{import qualified } f \texttt{ as } f' \ \check{\mathit{impspec}}) \rightsquigarrow \check{\mathit{eenv}}_{\mathsf{qual}}}$$

- $\boxed{\mathit{fenv}; f_0 \vdash^{\text{IL}}_{\text{c}} \check{\mathit{impdecls}}; \check{\mathit{defs}} \rightsquigarrow \check{\mathit{eenv}}}$ Resolution of all the imports of a module, along with the naming of the locally defined entities.

$$\frac{\begin{array}{c}\forall i \in [1..n] : \ \mathit{fenv} \vdash^{\text{IL}}_{\text{c}} \check{\mathit{impdecl}}_i \rightsquigarrow \check{\mathit{eenv}}_i \\ \check{\mathit{eenv}}_{\mathsf{imp}} = \bigoplus_{i \in [1..n]} \check{\mathit{eenv}}_i \qquad \check{\mathit{eenv}}_{\mathsf{loc}} = \mathsf{mklocaleenv}(f_0; \check{\mathit{defs}})\end{array}}{\mathit{fenv}; f_0 \vdash^{\text{IL}}_{\text{c}} \check{\mathit{impdecl}}_1, \ldots, \check{\mathit{impdecl}}_n; \check{\mathit{defs}} \rightsquigarrow (\check{\mathit{eenv}}_{\mathsf{imp}} \oplus \check{\mathit{eenv}}_{\mathsf{loc}})} \ (\textsc{IlEenvRes})$$

- $\boxed{\mathit{fenv}; f \vdash^{\text{IL}}_{\text{c}} \check{\mathit{impspec}} \rightsquigarrow \overline{\check{\mathit{espc}}}}$ Resolution of a single module's import specification. **Note**: This is different from EL since fewer import spec forms.

$$\frac{\overline{\mathit{fenv}; f \vdash^{\text{IL}}_{\text{c}} \check{\mathit{import}} \rightsquigarrow \check{\mathit{espc}}}}{\mathit{fenv}; f \vdash^{\text{IL}}_{\text{c}} (\overline{\check{\mathit{import}}}) \rightsquigarrow \bigoplus_{\check{\mathit{espc}} \in \overline{\check{\mathit{espc}}}} \{\check{\mathit{espc}}\}}$$

- $\boxed{\mathit{fenv}; f \vdash^{\text{IL}}_{\text{c}} \check{\mathit{import}} \rightsquigarrow \check{\mathit{espc}}}$ Resolution of a single entity import.

$$\frac{\check{\mathit{espc}} \in \mathit{fenv}(f) \qquad \check{\mathit{espc}} = [f']\chi}{\mathit{fenv}; f \vdash^{\text{IL}}_{\text{c}} \chi \rightsquigarrow \check{\mathit{espc}}}$$

  - Differs from EL in that it must be an entity with no subordinate names.

$$\frac{\check{\mathit{espc}} \in \mathit{fenv}(f) \qquad \check{\mathit{espc}} \leq \check{\mathit{espc}}' = [f']\chi(\overline{\chi}')}{\mathit{fenv}; f \vdash^{\text{IL}}_{\text{c}} \chi(\overline{\chi}') \rightsquigarrow \check{\mathit{espc}}'}$$

- $\boxed{f_0; \check{\mathit{eenv}} \vdash^{\text{IL}}_{\text{c}} \check{\mathit{expdecl}} \rightsquigarrow \check{\mathit{espcs}}}$ Resolution of module exports.

$$\frac{\begin{array}{c}\forall i \in [1..n] : \ f_0; \check{\mathit{eenv}} \vdash^{\text{IL}}_{\text{c}} \check{\mathit{export}}_i \rightsquigarrow \check{\mathit{espc}}_i \\ \check{\mathit{espcs}} = \bigoplus_{i \in [1..n]} \{\check{\mathit{espc}}_i\} \qquad \mathsf{nooverlap}(\check{\mathit{espcs}})\end{array}}{f_0; \check{\mathit{eenv}} \vdash^{\text{IL}}_{\text{c}} (\check{\mathit{export}}_1, \ldots, \check{\mathit{export}}_n) \rightsquigarrow \check{\mathit{espcs}}}$$

- $\boxed{f_0; \check{\mathit{eenv}} \vdash^{\text{IL}}_{\text{c}} \check{\mathit{export}} \rightsquigarrow \check{\mathit{espc}}}$ Resolution of a single entity export. The notational use of $\check{\mathit{eenv}}$ as a mapping requires that the given $\check{\mathit{eref}}$ maps to only a single physical name. **NOTE!** In the IL, only a single $\check{\mathit{espc}}$ is gotten from this judgment.

  Haskell permits export of subordinate names without their owning names in the case of class methods. We don't currently allow this because every exported entity must have its own $\check{\mathit{espc}}$, but a class method does not! We could perhaps add such subordinate names as their own $\check{\mathit{espcs}}$ (without subordinate lists).

$$\frac{\check{\mathit{eenv}}(\check{\mathit{eref}}) = [f]\chi : [f]\chi}{f_0; \check{\mathit{eenv}} \vdash^{\text{IL}}_{\text{c}} \check{\mathit{eref}} \rightsquigarrow [f]\chi} \ (\textsc{IlExpSimple})$$

– If the exported entity has no subordinate names, then it gets a normal export spec.

$$\frac{ee\check{n}v(er\check{e}f) = [f]\chi : [f]\chi(\overline{\chi}', \overline{\chi}'')}{f_0; ee\check{n}v \vdash^{\text{IL}}_{\text{c}} er\check{e}f\,(\overline{\chi}') \rightsquigarrow [f]\chi(\overline{\chi}')} \text{ (IlExpSubList)}$$

(Deliberately removed rule for when an $er\check{e}f$ refers to a name with subordinates.)

(Deliberately removed rule for elided subordinate names.)

(Deliberately removed rule for `module` *export.)*

7.3.1 Augmented environments in IL

With the augmented entity environments there is not a direct translation from EL to IL definitions because the IL $ae\check{n}v$ contains a full $fenv$, rather than something corresponding to $\tilde{\Phi}$. The semantics are copied below, but the only other change lies in the $ee\check{n}v$ mapping judgment, which no longer requires that "self" entity references have a local identity.

- $\boxed{ae\check{n}v \vdash^{\text{IL}}_{\text{c}} ee\check{n}v\ \mathsf{wf}}$ Determine whether the physical names in the range of the entity environment are indeed exported from their defining modules, or from a local definition. Also check that the locally available export specs all make sense.

$$ae\check{n}v \vdash^{\text{IL}}_{\text{c}} ee\check{n}v\ \mathsf{wf} \overset{\text{def}}{\Leftrightarrow} \begin{array}{l} \forall es\check{p}c \in ee\check{n}v : \begin{cases} ae\check{n}v \vdash^{\text{IL}}_{\text{c}} es\check{p}c\ \mathsf{wf} \\ \mathsf{mkphnms}(es\check{p}c) \subseteq \mathsf{rng}(ee\check{n}v) \end{cases} \\ \forall er\check{e}f \mapsto ph\check{n}m \in ee\check{n}v : \begin{cases} ae\check{n}v \vdash^{\text{IL}}_{\text{c}} er\check{e}f \mapsto ph\check{n}m\ \mathsf{wf} \\ ph\check{n}m \in \mathsf{mkphnms}(\mathsf{locals}(eenv)) \end{cases} \end{array}$$

- $\boxed{ae\check{n}v \vdash^{\text{IL}}_{\text{c}} er\check{e}f \mapsto ph\check{n}m\ \mathsf{wf}}$ Determine whether the physical name in the range of the mapping is indeed exported from its defining module, or from a local definition.

$$\frac{er\check{e}f = \chi \text{ or } f.\chi}{ae\check{n}v \vdash^{\text{IL}}_{\text{c}} er\check{e}f \mapsto [f]\chi\ \mathsf{wf}}$$

- $\boxed{ae\check{n}v \vdash^{\text{IL}}_{\text{c}} es\check{p}c\ \mathsf{wf}}$ $\boxed{ae\check{n}v \vdash^{\text{IL}}_{\text{c}} \overline{es\check{p}c}\ \mathsf{wf}}$ Check that a locally available export spec is well formed with respect to a local definition and/or a spec in the context.

$$\frac{es\check{p}c_0 = \mathsf{locmatch}(ae\check{n}v; es\check{p}c) \quad es\check{p}c_1 = \mathsf{ctxmatch}(ae\check{n}v; es\check{p}c) \quad es\check{p}c_0 \oplus es\check{p}c_1 \leq es\check{p}c}{ae\check{n}v \vdash^{\text{IL}}_{\text{c}} es\check{p}c\ \mathsf{wf}}$$

$$\frac{es\check{p}c_0 = \mathsf{locmatch}(ae\check{n}v; es\check{p}c) \quad \mathsf{noctxmatch}(ae\check{n}v; es\check{p}c) \quad es\check{p}c_0 \leq es\check{p}c}{ae\check{n}v \vdash^{\text{IL}}_{\text{c}} es\check{p}c\ \mathsf{wf}}$$

$$\frac{\mathsf{nolocmatch}(ae\check{n}v; es\check{p}c) \quad es\check{p}c_1 = \mathsf{ctxmatch}(ae\check{n}v; es\check{p}c) \quad es\check{p}c_1 \leq es\check{p}c}{ae\check{n}v \vdash^{\text{IL}}_{\text{c}} es\check{p}c\ \mathsf{wf}}$$

$$aenv \Vdash_{\mathsf{c}} \overline{\check{espc}}\ \mathsf{wf} \quad \stackrel{\text{def}}{\Leftrightarrow} \quad \forall \check{espc} \in \overline{\check{espc}} :\ aenv \vdash^{\text{IL}}_{\mathsf{c}} \check{espc}\ \mathsf{wf}$$

- $\boxed{\mathsf{islocal}(\check{aenv}; \check{espc})}$ $\boxed{\mathsf{islocal}(\check{aenv}; \check{phnm})}$ Is the given entity's identity the same as the local module? Or, in the case of signatures, is this entity one of the locally specified ones? (This does *not* check for a matching definition/declaration; it merely looks at the identity.)

$$\begin{array}{lcl} \mathsf{islocal}((\check{fenv}; f_0; \check{defs}); \check{espc}) & \stackrel{\text{def}}{\Leftrightarrow} & \mathsf{ident}(\check{espc}) = f_0 \\ \mathsf{islocal}((\check{fenv}; f_0; \check{defs}); [f]\chi) & \stackrel{\text{def}}{\Leftrightarrow} & f = f_0 \end{array}$$

- $\boxed{\mathsf{locmatch}(\check{aenv}; \check{espc})}$ $\boxed{\mathsf{ctxmatch}(\check{aenv}; \check{espc})}$ $\boxed{\mathsf{nolocmatch}(\check{aenv}; \check{espc})}$ $\boxed{\mathsf{noctxmatch}(\check{aenv}; \check{espc})}$ Find a matching export spec in the context or the local environment.

$$\begin{array}{lcll} \mathsf{locmatch}(\check{aenv}; \check{espc}) & \stackrel{\text{def}}{=} & \check{espc}' & \text{if } \begin{cases} \mathsf{islocal}(\check{aenv}; \check{espc}), \\ \exists \check{def} \in \check{aenv}.\check{defs} :\ \check{def} \sqsubseteq \check{espc}', \\ \check{espc}' \perp \check{espc} \end{cases} \\ \mathsf{ctxmatch}(\check{aenv}; \check{espc}) & \stackrel{\text{def}}{=} & \check{espc}' & \text{if } \begin{cases} \check{espc}' \in \check{aenv}.\check{fenv}(\mathsf{ident}(\check{espc})), \\ \check{espc}' \perp \check{espc} \end{cases} \\ \mathsf{nolocmatch}(\check{aenv}; \check{espc}) & \stackrel{\text{def}}{\Leftrightarrow} & \mathsf{islocal}(\check{aenv}; \check{espc}) \Rightarrow & \\ & & \quad \forall \check{espc}', \check{def} \in \check{aenv}.\check{defs} :\ \check{def} \sqsubseteq \check{espc}' \Rightarrow \check{espc}' \not\perp \check{espc} & \\ \mathsf{noctxmatch}(\check{aenv}; \check{espc}) & \stackrel{\text{def}}{\Leftrightarrow} & \forall \check{espc}' \in \check{aenv}.\check{fenv}(\mathsf{ident}(\check{espc})) : \check{espc}' \not\perp \check{espc} & \end{array}$$

7.4 Elaboration definitions

- $\boxed{\mathsf{refs}^{\star}_{\nu_0}(\mathit{eref})}$ $\boxed{\mathsf{refs}^{\star}_{\nu_0}(\mathit{eenv})}$ Translation of entity references in Haskell programs. This extends to a function on any syntactic category that contains *eref*. Note that it is injective since $p \in \mathit{Paths}$ and $\nu^{\star} \notin \mathit{Paths}$ for all ν. On entity envs it acts as such on the *eref*s but as $(-)^{\star}$ on the rest.

$$\begin{array}{lcl} \mathsf{refs}^{\star}_{\nu_0}(\chi) & \stackrel{\text{def}}{=} & \chi \\ \mathsf{refs}^{\star}_{\nu_0}(p.\chi) & \stackrel{\text{def}}{=} & p.\chi \\ \mathsf{refs}^{\star}_{\nu_0}(\mathsf{Local}.\chi) & \stackrel{\text{def}}{=} & (\nu_0{}^{\star}).\chi \end{array}$$

$$\mathsf{refs}^{\star}_{\nu_0}(\mathit{eenv}) \quad \stackrel{\text{def}}{=} \quad \{\mathsf{refs}^{\star}_{\nu_0}(\mathit{eref}) \mapsto \mathit{phnm}^{\star} \mid \mathit{eref} \mapsto \mathit{phnm} \in \mathit{eenv}\}; \mathsf{locals}(\mathit{eenv})^{\star}$$

- $\boxed{\mathsf{mkmod}(\mathcal{L}; \mathit{eenv}; \nu_0; \mathit{impdecls}; \mathit{defs}; \mathit{espcs})}$ Module patching. Translates an EL module into a directory expression containing a single, plain, IL module. Rewrite the logical imports to the identities' names; limit what entities are imported from each module; and limit what entities are exported.

$$\mathsf{mkmod}(\mathcal{L}; \mathit{eenv}; \nu_0; \mathit{impdecls}; \mathit{defs}; \mathit{espcs}) \stackrel{\text{def}}{=} \left(\begin{array}{l} \texttt{module}\ \nu_0{}^{\star}\ \mathsf{mkexpdecl}(\mathit{espcs}; \mathit{eenv})\ \texttt{where} \\ \quad \mathsf{mkimpdecls}(\mathcal{L}; \mathit{impdecls}) \\ \quad \mathsf{refs}^{\star}_{\nu_0}(\mathit{defs}) \end{array} \right)$$

$$
\begin{array}{lcl}
\mathsf{mkexpdecl}(\mathit{espcs}; \mathit{eenv}) & \stackrel{\text{def}}{=} & (\!(\mathsf{mkexp}(\mathit{espc}; \mathit{eenv}) \mid \mathit{espc} \in \mathit{espcs})\!) \\
\mathsf{mkexp}([\nu]\chi; \mathit{eenv}) & \stackrel{\text{def}}{=} & \mathsf{refs}^{\star}_{\nu_0}(\mathit{eref}) \text{ where } [\nu]\chi \sqsubseteq \mathit{eenv}(\mathit{eref}) \\
\mathsf{mkexp}([\nu]\chi(\overline{\chi'}); \mathit{eenv}) & \stackrel{\text{def}}{=} & \mathsf{refs}^{\star}_{\nu_0}(\mathit{eref})(\!(\overline{\chi'})\!) \text{ where } [\nu]\chi(\overline{\chi'}) \sqsubseteq \mathit{eenv}(\mathit{eref})
\end{array}
$$

The existence of such a *eref* is proven for all *espc* resulting from export resolution.

$$
\begin{array}{l}
\mathsf{mkimpdecls}(\mathcal{L}; \mathit{impdecls}) \stackrel{\text{def}}{=} (\mathsf{mkimpdecl}(\mathcal{L}; \mathit{impdecl}) \mid \mathit{impdecl} \in \mathit{impdecls}) \\
\mathsf{mkimpdecl}(\mathcal{L}; \texttt{import}\ [\texttt{qualified}]\ p\ \mathit{impspec}) \\
\qquad \stackrel{\text{def}}{=} \texttt{import}\ [\texttt{qualified}]\ \mathcal{L}^{\star}(p)\ \texttt{as}\ p\ \mathsf{mkimpspec}(\mathcal{L}; p; \mathit{impspec}) \\
\\
\mathsf{mkimpdecl}(\mathcal{L}; \texttt{import}\ [\texttt{qualified}]\ p\ \texttt{as}\ p'\ \mathit{impspec}) \\
\qquad \stackrel{\text{def}}{=} \texttt{import}\ [\texttt{qualified}]\ \mathcal{L}^{\star}(p)\ \texttt{as}\ p'\ \mathsf{mkimpspec}(\mathcal{L}; p; \mathit{impspec})
\end{array}
$$

$$
\begin{array}{lcl}
\mathsf{mkimpspec}(\mathcal{L}; p; \emptyset) & \stackrel{\text{def}}{=} & (\!(\overline{\mathsf{mkentimp}(\mathit{espc})})\!) \quad \text{if} \quad \overline{\mathit{espc}} \in \mathcal{L}(p) \\
\mathsf{mkimpspec}(\mathcal{L}; p; (\!(\overline{\mathit{import}})\!)) & \stackrel{\text{def}}{=} & (\!(\overline{\mathsf{mkentimp}(\mathcal{L}; p; \mathit{import})})\!)
\end{array}
$$

$$
\begin{array}{lcll}
\mathsf{mkentimp}(\mathcal{L}; p; \chi) & \stackrel{\text{def}}{=} & \mathsf{mkentimp}(\mathit{espc}) & \text{if} \quad \begin{cases} \mathit{espc} \in \mathcal{L}(p) \\ \chi = \mathsf{name}(\mathit{espc}) \end{cases} \\
\mathsf{mkentimp}(\mathcal{L}; p; \chi(\!(..)\!)) & \stackrel{\text{def}}{=} & \mathsf{mkentimp}(\mathit{espc}) & \text{if} \quad \begin{cases} \mathit{espc} \in \mathcal{L}(p) \\ \chi = \mathsf{name}(\mathit{espc}) \\ \mathsf{hasSubs}(\mathit{espc}) \end{cases} \\
\mathsf{mkentimp}(\mathcal{L}; p; \chi(\!(\overline{\chi'})\!)) & \stackrel{\text{def}}{=} & \chi(\!(\overline{\chi'})\!) &
\end{array}
$$

$$
\begin{array}{rcl}
\mathsf{mkentimp}([\nu]\chi) & \stackrel{\text{def}}{=} & \chi \\
\mathsf{mkentimp}([\nu]\chi(\overline{\chi'})) & \stackrel{\text{def}}{=} & \chi(\!(\overline{\chi'})\!)
\end{array}
$$

Note that instance declarations won't have names, so we need some new GHC magic to allow us to explicitly import and export them.

- $\boxed{\mathsf{mkstubs}(\Phi)}$ Converts a module context of all signatures into a set of IL stubs.

$$\mathsf{mkstubs}(\overline{\nu{:}\tau^{-}}) \stackrel{\text{def}}{=} \{\overline{\nu^{\star} \mapsto - : \tau^{\star}}\}$$

- $\boxed{\Phi_{\mathsf{ctx}} \vdash \Phi \sim \mathit{dexp}}$ $\boxed{\vdash (\nu{:}\tau^{m}) \sim (f \mapsto \mathit{tfexp})}$ Relate the EL context Φ to the IL term *dexp*. Essentially, this shows a direct correspondence between a module type in Φ and a file in *dexp*. Due to the way imports are elaborated, this is not merely checking for direct syntactic equivalence.

$$\frac{\begin{array}{c}\Phi_{\mathsf{ctx}} \vdash \Phi\ \mathsf{wf} \quad \Phi = \nu_1{:}{\tau_1}^{m_1}, \ldots, \nu_n{:}{\tau_n}^{m_n} \\ {\Phi_{\mathsf{ctx}}}^{\star} \vdash \mathit{dexp} \quad \mathit{dexp} = \{f_1 \mapsto \mathit{tfexp}_1, \ldots, f_n \mapsto \mathit{tfexp}_n\} \\ \forall i \in [1..n] : \vdash (\nu_i{:}{\tau_i}^{m_i}) \sim (f_i \mapsto \mathit{tfexp}_i)\end{array}}{\Phi_{\mathsf{ctx}} \vdash \Phi \sim \mathit{dexp}}$$

$$\frac{\nu^\star = f \qquad \tau^\star = \mathit{ftyp}}{\vdash (\nu{:}\tau^-) \sim (f \mapsto - : \mathit{ftyp})}$$

$$\frac{\nu^\star = f \qquad \tau^\star = \mathit{ftyp} \qquad \mathit{hsmod} = \mathsf{mkmod}(\mathcal{L}; \mathit{eenv}; \nu; \mathit{impdecls}; \mathit{defs}; \mathit{espcs})}{\vdash (\nu{:}\tau^+) \sim (f \mapsto \mathit{hsmod} : \mathit{ftyp})}$$

– The last premise says that the term was the result of our elaboration. This is needed to show that any (translated) identity substitution is valid on this term.

Part II
External language metatheory

8 Hauptsätze

Theorem 1 (Regularity of EL typing)**.** Assume $\Gamma = (\Phi; \mathcal{L})$, where applicable.

(1) If $\emptyset \vdash \Gamma$ wf and $\Gamma; \nu_0 \vdash_{\mathrm{c}} M : \tau$ and $\Phi \perp \nu_0{:}\tau^+$, then $\Phi \vdash \nu_0{:}\tau^+$ wf.
(2) If $\emptyset \vdash \Gamma$ wf and $\Gamma; \tilde{\tau} \vdash_{\mathrm{c}} S \rightsquigarrow \sigma; \Phi_\sigma$ and $\Phi \perp \Phi_\sigma$ and $\Phi \perp \nu_0{:}\sigma^-$, then $\Phi \vdash \Phi_\sigma$ wf and $\Phi \oplus \Phi_\sigma \vdash \nu_0{:}\sigma^-$ wf.
(3) If $\emptyset \vdash \Gamma$ wf and $\Gamma; \tilde{\Sigma}_{\mathsf{pkg}} \vdash E : \Sigma$ and $\Phi \perp \Sigma.\Phi$, then $\Phi \vdash \Sigma$ wf.
(4) If $\vdash \Delta$ wf and $\emptyset \vdash \Gamma$ wf and $\Delta; \Gamma; \tilde{\Xi}_{\mathsf{pkg}} \vdash B : \Xi$ and $\Gamma \perp \Xi$, then $\Phi \vdash \Xi$ wf. (Proof uses reg on E; subst invariance on Ξ wf; weakening on Ξ wf; wf-preservation of thinning (Lemma 8); and wf-preservation of path manip (Lemma 11).)
(5) If $\vdash \Delta$ wf and $\Delta; \tilde{\Xi}_{\mathsf{pkg}} \vdash \overline{B} : \Xi$, then $\emptyset \vdash \Xi$ wf. (Proof, by induction, uses reg on B; itself; and cut (Lemma 7).)
(6) If $\vdash \Delta$ wf and $\Delta \vdash D : \forall \overline{\alpha}.\Xi$, then $\emptyset \vdash \Xi$ wf. (Proof uses reg on $\overline{B}$ and wf-preservation of thinning (Lemma 8).)

Theorem 2 (Soundness of the elaboration)**.** Assume $\Gamma = (\Phi; \mathcal{L})$, where applicable.

(1) If $\emptyset \vdash \Gamma$ wf and $\Gamma; \nu_0 \vdash_{\mathrm{c}} M : \tau \rightsquigarrow \mathit{hsmod}$ and $\Phi \perp \nu_0{:}\tau^+$, then $\Phi^\star \vdash^{\mathrm{IL}}_{\mathrm{c}} \mathit{hsmod} : \tau^\star$. (Proof follows from Lemma 115 and Lemma 118.)
(2) If $\emptyset \vdash \Gamma$ wf and $\Gamma; \tilde{\Sigma}_{\mathsf{pkg}} \vdash E : \Sigma \rightsquigarrow \mathit{dexp}$ and $\Phi \perp \Sigma.\Phi$, then $\Phi \vdash \Sigma.\Phi \sim \mathit{dexp}$.
(3) If $\vdash \Delta$ wf and $\emptyset \vdash \Gamma$ wf and $\Delta; \Gamma; \tilde{\Xi}_{\mathsf{pkg}} \vdash B : \Xi \rightsquigarrow \mathit{dexp}$ and $\Gamma \perp \Xi$, then $\Phi \vdash \Xi.\Phi \sim \mathit{dexp}$.
(4) If $\vdash \Delta$ wf and $\Delta; \tilde{\Xi}_{\mathsf{pkg}} \vdash \overline{B} : \Xi \rightsquigarrow \mathit{dexp}$, then $\emptyset \vdash \Xi.\Phi \sim \mathit{dexp}$.
(5) If $\vdash \Delta$ wf and $\Delta \vdash D : \forall \overline{\alpha}.\Xi \rightsquigarrow \lambda \overline{\alpha}.\mathit{dexp}$, then $\emptyset \vdash \Xi.\Phi \sim \mathit{dexp}$.

9 Judgmental properties of classifiers

Lemma 3 (Weakening physical context preserves well-formedness)**.** Suppose $\Phi \perp \Phi_{\mathsf{W}}$.

(1) If $\Phi; \mathit{kenv} \vdash_{\mathrm{c}} \mathit{typ} :: \mathit{knd}$ then $\Phi \oplus \Phi_{\mathsf{W}}; \mathit{kenv} \vdash_{\mathrm{c}} \mathit{typ} :: \mathit{knd}$.
(2) If $\Phi \vdash \mathit{dspc}$ wf then $\Phi \oplus \Phi_{\mathsf{W}} \vdash \mathit{dspc}$ wf.
(3) If $\Phi \vdash \mathit{espc}$ wf then $\Phi \oplus \Phi_{\mathsf{W}} \vdash \mathit{espc}$ wf.

(4) If $\Phi \vdash \tau$ wf then $\Phi \oplus \Phi_{\mathsf{W}} \vdash \tau$ wf.
(5) If $\Phi \vdash \Phi'$ specs-wf then $\Phi \oplus \Phi_{\mathsf{W}} \vdash \Phi'$ specs-wf.
(6) If $\Phi \vdash \Phi'$ exports-wf then $\Phi \oplus \Phi_{\mathsf{W}} \vdash \Phi'$ exports-wf.
(7) If $\Phi \vdash \Phi'$ deps-wf then $\Phi \oplus \Phi_{\mathsf{W}} \vdash \Phi'$ deps-wf. (Proof uses antitonicity of depends [Property 49] and Property 60.)
(8) If $\Phi \vdash \Phi'$ wf and $\Phi' \perp \Phi_{\mathsf{W}}$, then $\Phi \oplus \Phi_{\mathsf{W}} \vdash \Phi'$ wf.

(9) If $\Phi \vdash \mathcal{L}$ wf then $\Phi \oplus \Phi_{\mathsf{W}} \vdash \mathcal{L}$ wf.
(10) If $\Phi \vdash \Xi$ wf and $\Xi.\Phi \perp \Phi_{\mathsf{W}}$, then $\Phi \oplus \Phi_{\mathsf{W}} \vdash \Xi$ wf.

Lemma 4 (Merging preserves well-formedness)**.** Suppose $\Phi_1 \perp \Phi_2$.
(1) If validspc$(dspc_1; m_1)$ and validspc$(dspc_2; m_2)$ and $dspc_1 \perp dspc_2$, then validspc$(dspc_1 \oplus dspc_2; m_1 \oplus m_2)$.
(2) If $\Phi_1 \vdash dspc_1$ wf and $\Phi_2 \vdash dspc_2$ wf and $dspc_1 \perp dspc_2$, then $\Phi_1 \oplus \Phi_2 \vdash dspc_1 \oplus dspc_2$ wf.
(3) If $\Phi_1 \vdash espc_1$ wf and $\Phi_2 \vdash espc_2$ wf and $espc_1 \perp espc_2$, then $\Phi_1 \oplus \Phi_2 \vdash espc_1 \oplus espc_2$ wf.

(4) If $\Phi_1 \vdash \tau_1$ wf and $\Phi_2 \vdash \tau_2$ wf and $\tau_1 \perp \tau_2$, then $\Phi_1 \oplus \Phi_2 \vdash \tau_1 \oplus \tau_2$ wf. (Proof uses merge and weakening on *dspc* and *espc*.)
(5) If $\Phi_1 \vdash \Phi_1'$ specs-wf and $\Phi_2 \vdash \Phi_2'$ specs-wf and $\Phi_1' \perp \Phi_2'$, then $\Phi_1 \oplus \Phi_2 \vdash \Phi_1' \oplus \Phi_2'$ specs-wf. (Proof uses merge and weakening on *dspc*.)
(6) If $\Phi_1 \vdash \Phi_1'$ exports-wf and $\Phi_2 \vdash \Phi_2'$ exports-wf and $\Phi_1' \perp \Phi_2'$, then $\Phi_1 \oplus \Phi_2 \vdash \Phi_1' \oplus \Phi_2'$ exports-wf. (Proof uses merge and weakening on *espc*.)
(7) If $\Phi_1 \vdash \Phi_1'$ deps-wf and $\Phi_2 \vdash \Phi_2'$ deps-wf and $\Phi_1' \perp \Phi_2'$, then $\Phi_1 \oplus \Phi_2 \vdash \Phi_1' \oplus \Phi_2'$ deps-wf. (Proof uses antitonicity of depends, Property 49.)
(8) If $\Phi_1 \vdash \Phi_1'$ wf and $\Phi_2 \vdash \Phi_2'$ wf and $\Phi_1' \perp \Phi_2'$ and $\Phi_1 \perp \Phi_2'$ and $\Phi_2 \perp \Phi_1'$, then $\Phi_1 \oplus \Phi_2 \vdash \Phi_1' \oplus \Phi_2'$ wf. (Proof uses merge and weakening on *espc* and Property 71.)

(9) If $\Phi_1 \vdash \mathcal{L}_1$ wf and $\Phi_2 \vdash \mathcal{L}_2$ wf and $\mathcal{L}_1 \perp \mathcal{L}_2$, then $\Phi_1 \oplus \Phi_2 \vdash \mathcal{L}_1 \oplus \mathcal{L}_2$ wf. (Proof uses merge and weakening on τ.)
(10) If $\Phi_1 \vdash \Xi_1$ wf and $\Phi_2 \vdash \Xi_2$ wf and $\Xi_1.\Phi \perp \Xi_2.\Phi$ and $\Phi_1 \perp \Xi_2.\Phi$ and $\Phi_2 \perp \Xi_1.\Phi$, then $\Phi_1 \oplus \Phi_2 \vdash \Xi_1 \oplus \Xi_2$ wf. (Proof uses merge on $\mathcal{L}$ and Φ.)

Lemma 5 (Invariance of well-formedness under substitution)**.** Suppose $\emptyset \vdash \Phi$ wf and let $\Phi_\theta = \mathsf{apply}(\theta; \Phi)$ be defined.
(1) If $\Phi; kenv \vdash_{\mathrm{c}} typ :: knd$ then $\Phi_\theta; kenv \vdash_{\mathrm{c}} \theta typ :: knd$.
(2) If $\Phi \vdash dspc$ wf then $\Phi_\theta \vdash \theta dspc$ wf.
(3) If $\Phi \vdash espc$ wf then $\Phi_\theta \vdash \theta espc$ wf.

(4) If $\Phi \vdash \tau$ wf then $\Phi_\theta \vdash \theta\tau$ wf.
(5) If $\Phi \vdash \Phi'$ specs-wf and $\mathsf{apply}(\theta; \Phi')$ defined then $\Phi_\theta \vdash \mathsf{apply}(\theta; \Phi')$ specs-wf. (Proof uses invariace and merge on *dspc*.)
(6) If $\Phi \vdash \Phi'$ exports-wf and $\mathsf{apply}(\theta; \Phi')$ defined then $\Phi_\theta \vdash \mathsf{apply}(\theta; \Phi')$ exports-wf. (Proof uses invariace and merge on *espc*.)
(7) If $\Phi \vdash \Phi'$ deps-wf and $\mathsf{apply}(\theta; \Phi')$ defined then $\Phi_\theta \vdash \mathsf{apply}(\theta; \Phi')$ deps-wf. (Proof uses Property 50.)

(8) If $\Phi \vdash \Phi'$ wf and $\mathsf{apply}(\theta;\Phi')$ is defined and $\Phi_\theta \perp \mathsf{apply}(\theta;\Phi')$, then $\Phi_\theta \vdash \mathsf{apply}(\theta;\Phi')$ wf. (Proof uses Lemma 7 and invariance on exps/specs-wf.)

(9) If $\Phi \vdash \mathcal{L}$ wf then $\Phi_\theta \vdash \theta\mathcal{L}$ wf. (Proof uses Lemma 7.)
(10) If $\Phi \vdash \Xi$ wf and $\mathsf{apply}(\theta;\Xi.\Phi)$ is defined and $\Phi_\theta \perp \mathsf{apply}(\theta;\Xi.\Phi)$, then $\Phi_\theta \vdash \mathsf{apply}(\theta;\Xi)$ wf. (Proof uses Lemma 7 and invariance on $\mathcal{L}$ and Φ.)

Lemma 6 (Preservation of well-formedness under context strengthening)**.** Suppose $N \subseteq \mathsf{dom}(\Phi)$.
(1) If $\Phi; \mathit{kenv} \vdash_{\mathrm{c}} \mathit{typ} :: \mathit{knd}$ and $\mathsf{provs}(\mathit{typ}) \subseteq N$, then $\Phi|_N; \mathit{kenv} \vdash_{\mathrm{c}} \mathit{typ} :: \mathit{knd}$.
(2) If $\Phi \vdash \mathit{dspc}$ wf and $\mathsf{provs}(\mathit{dspc}) \subseteq N$, then $\Phi|_N \vdash \mathit{dspc}$ wf.
(3) If $\Phi \vdash \mathit{espc}$ wf and $\mathsf{ident}(\mathit{espc}) \in N$, then $\Phi|_N \vdash \mathit{espc}$ wf.

(4) If $\Phi \vdash \tau$ wf and $\mathsf{provs}(\tau) \in N$, then $\Phi|_N \vdash \tau$ wf.
(5) If $\Phi \vdash \Phi$ specs-wf and $\mathsf{depends}_\Phi(N) \subseteq N$, then $\Phi|_N \vdash \Phi|_N$ specs-wf.
(6) If $\Phi \vdash \Phi$ exports-wf and $\mathsf{depends}_\Phi(N) \subseteq N$, then $\Phi|_N \vdash \Phi|_N$ exports-wf.
(7) If $\Phi \vdash \Phi$ deps-wf and $\mathsf{depends}_\Phi(N) \subseteq N$, then $\Phi|_N \vdash \Phi|_N$ deps-wf. (Proof uses Lemma 51.)
(8) If $\emptyset\vdash \Phi$ wf and $\mathsf{depends}_\Phi(N) \subseteq N$, then $\emptyset\vdash \Phi|_N$ wf.

(9) If $\Phi \vdash \mathcal{L}$ wf and $\mathsf{depends}_\Phi(N) \subseteq N$, then $\Phi|_N \vdash \mathcal{L}|_N$ wf.
(10) If $\emptyset\vdash (\Phi;\mathcal{L})$ wf and $\mathsf{depends}_\Phi(N) \subseteq N$, then $\emptyset\vdash (\Phi;\mathcal{L})|_N$ wf.

Lemma 7 (Cut on physical contexts and signatures)**.**
(1) If $\Phi \vdash \Phi_1$ wf and $\Phi \oplus \Phi_1 \vdash \Phi_2$ wf, then $\Phi_1 \perp \Phi_2$ and $\Phi \vdash \Phi_1 \oplus \Phi_2$ wf. (Proof uses merge on exps/specs/deps-wf.)
(2) If $\emptyset\vdash \Xi_1$ wf and $\Xi_1.\Phi \vdash \Xi_2$ wf and $\Xi_1 \perp \Xi_2$, then $\emptyset\vdash \Xi_1 \oplus \Xi_2$ wf. (Proof uses cut on physical contexts and merge on logical contexts.)

Lemma 8 (Thinning preserves well-formedness)**.** If $\emptyset\vdash \Xi$ wf and $\vdash \Xi \xrightarrow{t} \Xi'$ then $\emptyset\vdash \Xi'$ wf and $\mathsf{depends}_{\Xi.\Phi}(N) \subseteq N$, where $N = \mathsf{dom}(\Xi'.\Phi)$.

Lemma 9 (Well-formed contexts contain all modules' dependencies)**.** If $\emptyset\vdash \Phi$ wf then $\mathsf{depends}_\Phi(\mathsf{dom}(\Phi)) \subseteq \mathsf{dom}(\Phi)$.

Lemma 10 (Well-formed contexts contain all modules' dependencies)**.** If $\emptyset\vdash \Phi'$ wf and $\nu{:}\tau^- \in \Phi'$, then $\Phi \vdash \tau$ wf. (Proof straightforward since this judgment checks the same properties as the context judgments.)

Lemma 11 (Path manipulation preserves well-formedness)**.**
- If $\mathsf{cons}(p;\Sigma)$ is defined and $\Phi \vdash \Sigma$ wf, then $\Phi \vdash \mathsf{cons}(p;\Sigma)$ wf.
- If $\mathsf{rename}(r;\Xi)$ is defined and $\Phi \vdash \Xi$ wf, then $\Phi \vdash \mathsf{rename}(r;\Xi)$ wf.

10 Invariants of auxiliary EL module machinery

10.1 Import resolution

Lemma 12 (Soundness of module entity environment construction (EL))**.** If $\emptyset\Vdash (\tilde{\Phi};\tilde{\mathcal{L}})$ wf and $\tilde{\mathcal{L}};\nu_0 \Vdash_{\mathrm{c}} \mathit{impdecls}; \mathit{defs} \rightsquigarrow \mathit{eenv}$ and $\mathit{aenv}^+ = (\tilde{\Phi};\nu_0;\mathit{defs})$ [and $\mathsf{nooverlap}(\mathit{defs})$], then $\mathit{aenv}^+ \Vdash_{\mathrm{c}} \mathit{eenv}$ wf and $\mathsf{haslocaleenv}(\mathit{eenv};\nu_0;\mathit{defs})$.

Lemma 13 (Soundness of signature entity environment construction (EL))**.** If $\emptyset \Vdash (\tilde{\Phi}; \tilde{\mathcal{L}})$ wf and $\tilde{\mathcal{L}}; \tilde{\tau} \Vdash_{\mathsf{c}} \mathit{impdecls}; \mathit{decls} \rightsquigarrow \mathit{eenv}$ and $\mathit{aenv}^- = (\tilde{\Phi}; \tilde{\tau}; \mathit{decls})$ and [and $\mathsf{nooverlap}(\mathit{defs})$], then $\mathit{aenv}^- \Vdash_{\mathsf{c}} \mathit{eenv}$ wf and $\mathsf{haslocaleenv}(\mathit{eenv}; \tilde{\tau}; \mathit{decls})$.

Lemma 14 (Soundness of import declaration resolution (EL))**.** If $\emptyset \Vdash (\tilde{\Phi}; \tilde{\mathcal{L}})$ wf and $\tilde{\mathcal{L}} \Vdash_{\mathsf{c}} \mathit{impdecl} \rightsquigarrow \mathit{eenv}$ and $\mathit{aenv} = (\tilde{\Phi}; \nu_0; \mathit{ds})$ [and $\mathsf{nooverlap}(\mathit{aenv})$], then $\mathit{aenv} \Vdash_{\mathsf{c}} \mathit{eenv}$ wf.

Lemma 15 (Soundness of imported module specification resolution (EL))**.** If $\emptyset \Vdash (\tilde{\Phi}; \tilde{\mathcal{L}})$ wf and $\tilde{\mathcal{L}}; p \Vdash_{\mathsf{c}} \mathit{impspec} \rightsquigarrow \overline{\mathit{espc}}$ and $\mathit{aenv} = (\tilde{\Phi}; \nu_0; \mathit{ds})$ [and $\mathsf{nooverlap}(\mathit{aenv})$], then $\mathit{aenv} \Vdash_{\mathsf{c}} \overline{\mathit{espc}}$ wf.

Lemma 16 (Soundness of imported module entity resolution (EL))**.** If $\emptyset \Vdash (\tilde{\Phi}; \tilde{\mathcal{L}})$ wf and $\tilde{\mathcal{L}}; p \Vdash_{\mathsf{c}} \mathit{import} \rightsquigarrow \mathit{espc}$ and $\mathit{aenv} = (\tilde{\Phi}; \nu_0; \mathit{ds})$, then $\mathit{aenv} \Vdash_{\mathsf{c}} \mathit{espc}$ wf.

10.1.1 Technical lemmas needed

Property 17 (Local entity environment is always well-formed)**.**

- $(\tilde{\Phi}; \nu_0; \mathit{defs}) \Vdash_{\mathsf{c}} \mathsf{mklocaleenv}(\nu_0; \mathit{defs})$ wf
- $(\tilde{\Phi}; \tilde{\tau}; \mathit{decls}) \Vdash_{\mathsf{c}} \mathsf{mklocaleenv}(\tilde{\tau}; \mathit{decls})$ wf

Lemma 18 (Qualification preserves entity env well-formedness)**.** If $\mathit{aenv} \Vdash_{\mathsf{c}} \mathit{eenv}$ wf then $\mathit{aenv} \Vdash_{\mathsf{c}} \mathsf{qualify}(\mathit{eenv}; m)$ wf.

Lemma 19 (Merging well-formed entity environments produces well-formed entity environments)**.**

- If $\mathit{aenv} \Vdash_{\mathsf{c}} \mathit{espc}_1$ wf and $\mathit{aenv} \Vdash_{\mathsf{c}} \mathit{espc}_2$ wf [and $\mathsf{nooverlap}(\mathit{defs})$] and $\mathit{espc}_1 \perp \mathit{espc}_2$, then $\mathit{aenv} \Vdash_{\mathsf{c}} \mathit{espc}_1 \oplus \mathit{espc}_2$ wf.
- If $\mathit{aenv} \Vdash_{\mathsf{c}} \overline{\mathit{espc}_1}$ wf and $\mathit{aenv} \Vdash_{\mathsf{c}} \overline{\mathit{espc}_2}$ wf [and $\mathsf{nooverlap}(\mathit{defs})$] and $\overline{\mathit{espc}_1} \perp \overline{\mathit{espc}_2}$, then $\mathit{aenv} \Vdash_{\mathsf{c}} \overline{\mathit{espc}_1} \oplus \overline{\mathit{espc}_2}$ wf.
- If $\mathit{aenv} \Vdash_{\mathsf{c}} \mathit{eenv}_1$ wf and $\mathit{aenv} \Vdash_{\mathsf{c}} \mathit{eenv}_2$ wf [and $\mathsf{nooverlap}(\mathit{defs})$] and $\mathit{eenv}_1 \perp \mathit{eenv}_2$, then $\mathit{aenv} \Vdash_{\mathsf{c}} \mathit{eenv}_1 \oplus \mathit{eenv}_2$ wf.

Lemma 20.

- If $\mathit{aenv} \Vdash_{\mathsf{c}} \mathit{espc}$ wf then $\mathit{aenv} \Vdash_{\mathsf{c}} \mathsf{mkeenv}(\mathit{espc})$ wf.
- If $\mathit{aenv} \Vdash_{\mathsf{c}} \overline{\mathit{espc}}$ wf then $\mathit{aenv} \Vdash_{\mathsf{c}} \mathsf{mkeenv}(\overline{\mathit{espc}})$ wf.

Property 21 (Domains of certain kinds of entity environments)**.**

- $\forall \mathit{eref} \in \mathsf{dom}(\mathsf{qualify}(m; \mathit{eenv})) : \mathit{eref} = m.\chi$ for some χ.
- $\forall \mathit{eref} \in \mathsf{dom}(\mathsf{mkeenv}(\mathit{espcs})) : \mathit{eref} = \chi$ for some χ.
- $\forall \mathit{eref} \in \mathsf{dom}(\mathsf{mklocaleenv}(\mathit{espcs};)) : \mathit{eref} = \chi$ or $\mathsf{Local}.\chi$ for some χ.
- $\forall \mathit{eref} \in \mathsf{dom}(\mathsf{mklocaleenv}(\nu; \mathit{defs})) : \mathit{eref} = \chi$ or $\mathsf{Local}.\chi$ for some χ.
- $\forall \mathit{eref} \in \mathsf{dom}(\mathsf{mklocaleenv}(\tilde{\tau}; \mathit{decls})) : \mathit{eref} = \chi$ or $\mathsf{Local}.\chi$ for some χ.

Property 22 (Distributivity of substitution over entity environment merging)**.** This is a corollary to Property 65.

(1) If $\mathit{ee\check{n}v}'_1 = \mathsf{apply}(\theta; \mathit{ee\check{n}v}_1)$ is defined and if $\mathit{ee\check{n}v}'_2 = \mathsf{apply}(\theta; \mathit{ee\check{n}v}_2)$ is defined and $\mathit{ee\check{n}v}_1 \perp \mathit{ee\check{n}v}_2$ and $\mathit{ee\check{n}v}'_1 \perp \mathit{ee\check{n}v}'_2$, then $\mathsf{apply}(\theta; \mathit{ee\check{n}v}_1 \oplus \mathit{ee\check{n}v}_2)$ is defined and equals $\mathit{ee\check{n}v}'_1 \oplus \mathit{ee\check{n}v}'_2$.

(2) If $\forall i \in [1..n] : \mathsf{apply}(\theta; \mathit{ee\check{n}v}_i)$ defined and $\forall i, j \in [1..n] : \mathit{ee\check{n}v}_i \perp \mathit{ee\check{n}v}_j \wedge \mathsf{apply}(\theta; \mathit{ee\check{n}v}_i) \perp \mathsf{apply}(\theta; \mathit{ee\check{n}v}_j)$ defined, then $\mathsf{apply}(\theta; \bigoplus_{i \in [1..n]} \mathit{ee\check{n}v}_i)$ is defined and equals $\bigoplus_{i \in [1..n]} \mathsf{apply}(\theta; \mathit{ee\check{n}v}_i)$.

10.2 Definition and declaration checking

Axiom 23 (Soundness of module definition checking in IL)**.** If $\mathit{fenv} \vdash^{\text{IL}}_{\text{c}} \overline{\check{\mathit{defs}}} : \overline{\check{\mathit{dspc}}}$ and $\emptyset \vdash \mathit{fenv}$ wf and $\mathit{fenv}; f_0; \overline{\check{\mathit{defs}}} \vdash^{\text{IL}}_{\text{c}} \check{\mathit{eenv}}$ wf and $\mathit{ftyp} = \langle\!| \overline{\check{\mathit{dspc}}} ; \overline{\check{\mathit{espc}}} ; \overline{f'} |\!\rangle$ and $\mathit{fenv} \perp f{:}\mathit{ftyp}^+$, then $\mathit{fenv} \oplus f{:}\mathit{ftyp}^+ \vdash \mathit{ftyp}^+$ specs-wf.

Corollary 24 (Soundness of module definition checking in image of translation)**.** If $\Phi^\star \vdash^{\text{IL}}_{\text{c}} [: \mathsf{refs}^\star_{\nu_0}(\mathit{eenv})][\nu_0^\star]\mathsf{refs}^\star_{\nu_0}(\mathit{defs})\overline{\mathit{dspc}^\star}$ and $\emptyset \vdash \Phi$ wf and $\mathsf{shape}(\Phi); \nu_0; \mathit{defs} \Vdash_{\text{c}} \mathit{eenv}$ wf and $\tau = \langle\!| \overline{\mathit{dspc}} ; \overline{\mathit{espc}} ; \overline{\nu'} |\!\rangle$ and $\Phi \perp \nu_0{:}\tau^+$, then $\Phi \oplus \nu_0{:}\tau^+ \vdash \tau^+$ specs-wf. The proof of this corollary was sketched on paper (around 11 Jan; later 26 Feb) and requires a few lemmas that were also proved/sketched on paper.

Axiom 25 (Regularity of definition checking in IL)**.** If $\mathit{fenv}; \check{\mathit{eenv}}; f \vdash^{\text{IL}}_{\text{c}} \overline{\check{\mathit{def}}} : \overline{\check{\mathit{dspc}}}$ then $|\overline{\check{\mathit{def}}}| = |\overline{\check{\mathit{dspc}}}|$ and $\overline{\check{\mathit{def}}} \sqsubseteq \overline{\check{\mathit{dspc}}}$ and $\mathsf{nooverlap}(\overline{\check{\mathit{def}}})$ and $\{\overline{\mathsf{provs}(\check{\mathit{dspc}})}\} \subseteq \mathsf{dom}(\mathit{fenv}) \cup \{f_0\}$.

Corollary 26 (Regularity of definition checking in image of translation)**.** If $\Phi^\star; \mathsf{refs}^\star_{\nu_0}(\mathit{eenv}); \nu_0^\star \vdash^{\text{IL}}_{\text{c}} \overline{\mathsf{refs}^\star_{\nu_0}(\mathit{def})} : \overline{\check{\mathit{dspc}}}$ then $|\overline{\mathsf{refs}^\star_{\nu_0}(\mathit{def})}| = |\overline{\check{\mathit{dspc}}}|$ and $\exists \overline{\mathit{dspc}}$ s.t. $\overline{\mathit{dspc}^\star} = \overline{\check{\mathit{dspc}}}$ and $\overline{\mathit{def}} \sqsubseteq \overline{\mathit{dspc}}$ and $\mathsf{nooverlap}(\overline{\mathit{def}})$ and $\{\overline{\mathsf{provs}(\mathit{dspc})}\} \subseteq \mathsf{dom}(\Phi) \cup \{\nu_0\}$.

Axiom 27 (Soundness of declaration checking in EL)**.** If $\Phi; \mathit{eenv}; \tilde{\sigma} \vdash_{\text{c}} \mathit{decls} \rightsquigarrow \overline{\mathit{dspc}}$ and $\overline{\mathit{dspc}} \sqsubseteq \overline{\mathit{espc}}$ and $\Phi_\sigma = \mathsf{mksigenv}(\overline{\mathit{dspc}}; \overline{\mathit{espc}})$ and $\emptyset \vdash \Phi$ wf and $\Phi \perp \Phi_\sigma$, then $\Phi \oplus \mathsf{shallow}(\Phi_\sigma) \vdash \Phi_\sigma$ specs-wf.

Axiom 28 (Regularity of declaration checking in EL)**.** If $\Phi; \mathit{eenv}; \tilde{\tau} \vdash_{\text{c}} \overline{\mathit{decl}} \rightsquigarrow \overline{\mathit{dspc}}$ then $|\overline{\mathit{decl}}| = |\overline{\mathit{dspc}}|$ and $\overline{\mathit{decl}} \sqsubseteq \overline{\mathit{dspc}}$ and $\mathsf{nooverlap}(\overline{\mathit{decl}})$ and $\{\overline{\mathsf{provs}(\mathit{dspc})}\} \subseteq \mathsf{dom}(\Phi) \cup \mathsf{provs}(\tilde{\tau})$.

10.3 Export resolution

Lemma 29 (Soundness of module export resolution (EL))**.** If $\nu_0; \mathit{eenv} \Vdash_{\text{c}} \mathit{expdecl} \rightsquigarrow \overline{\mathit{espc}}$ and $\emptyset \vdash \Phi$ wf and $\mathit{aenv}^+ = (\mathsf{shape}(\Phi); \nu_0; \overline{\mathit{def}})$ and $\mathit{aenv}^+ \Vdash_{\text{c}} \mathit{eenv}$ wf [and $\mathsf{nooverlap}(\overline{\mathit{def}})$] and $\overline{\mathit{def}} \sqsubseteq \mathit{dspc}$ and $\tau = \langle\!| \overline{\mathit{dspc}} ; \overline{\mathit{espc}} ; \overline{\nu'} |\!\rangle$ and $\Phi \perp \nu_0{:}\tau^+$, then $\Phi \oplus \nu_0{:}\tau^+ \vdash \tau$ exports-wf.

Lemma 30 (Soundness of declaration exports in EL)**.** If $\tilde{\tau} \Vdash_{\text{c}} \mathit{decls} \rightsquigarrow \overline{\mathit{espc}}$ and $\mathsf{nooverlap}(\mathit{decls})$ and $\overline{\mathit{dspc} \sqsubseteq \mathit{espc}}$ and $\emptyset \vdash \Phi$ wf and $\Phi_\sigma = \mathsf{mksigenv}(\overline{\mathit{dspc}}; \overline{\mathit{espc}})$ and $\Phi \perp \Phi_\sigma$, then $\Phi \oplus \Phi_\sigma \vdash \Phi_\sigma$ exports-wf.

Lemma 31 (Regularity of declaration exports in EL)**.** If $\tilde{\tau} \Vdash_{\text{c}} \overline{\mathit{decl}} \rightsquigarrow \overline{\mathit{espc}}$ then $|\overline{\mathit{decl}}| = |\overline{\mathit{espc}}|$ and $\overline{\mathit{decl} \sqsubseteq \mathit{espc}}$.

Lemma 32 (Kinding is liftable)**.** If $\Phi^\star; \mathit{kenv} \vdash^{\text{IL}}_{\text{c}} \mathit{typ}^\star :: \mathit{knd}$ then $\Phi; \mathit{kenv} \vdash_{\text{c}} \mathit{typ} :: \mathit{knd}$.

10.3.1 Technical lemmas needed

Lemma 33 (Soundness of single export resolution (EL))**.** If $\nu_0; \mathit{eenv} \Vdash_{\text{c}} \mathit{export} \rightsquigarrow \overline{\mathit{espc}}$ and $\emptyset \vdash \Phi$ wf and $\mathit{aenv}^+ = (\mathsf{shape}(\Phi); \nu_0; \overline{\mathit{def}})$ and $\mathit{aenv}^+ \Vdash_{\text{c}} \mathit{eenv}$ wf and $\mathsf{consistent}(\mathit{aenv}^+)$ and $\overline{\mathit{def}} \sqsubseteq \mathit{dspc}$ and $\tau = \langle\!| \mathit{dspc} ; \overline{\mathit{espc}} ; \overline{\nu'} |\!\rangle$ and $\Phi \perp \nu_0{:}\tau^+$, then $\Phi \oplus \nu_0{:}\tau^- \vdash \tau$ exports-wf. *(Note the polarity of τ in the context of the conclusion.)*

Lemma 34. If $\emptyset \vdash \Phi$ wf and $aenv = (\mathsf{shape}(\Phi); \nu_0; \overline{def})$ and $aenv \Vdash_{\mathrm{c}} espc$ wf and consistent$(aenv)$ and ident$(espc) = \nu_0$ and $espc \in \overline{espc}$ and $\overline{def} \sqsubseteq \overline{dspc}$ and $\tau = \langle\!| \overline{dspc} \mathbin{;} \overline{espc} \mathbin{;} \overline{\nu'} |\!\rangle$ and $\Phi \perp \nu_0{:}\tau^+$, then $\Phi \oplus \nu_0{:}\tau^+ \vdash espc$ wf. (Used in proof of export soundness for the local exports case. This is not worth making a lemma! Just inline it.)

Lemma 35 (Consistent augmented environments come from mergeable modules (EL))**.** If $\overline{def \sqsubseteq dspc}$ and $\overline{dspc} \in \tau$ and $\Phi \perp \nu_0{:}\tau^+$ [and nooverlap$(\overline{def})$, then consistent$(aenv)$, where $aenv = (\Phi; \nu_0; defs)$. (Proof very straightforward; uses Lemma 58.)

Lemma 36 (Export well-formedness is invariant to polarity in the context)**.** If $\overline{\nu{:}\tau^m} \vdash \Phi'$ exports-wf then, for any $\overline{m'}$ s.t. $|\overline{m'}| = |\overline{m}|$, $\overline{\nu{:}\tau^{m'}} \vdash \Phi'$ exports-wf.

Property 37 (Matching in augmented environments succeeds or it doesn't)**.**

- Either $\exists espc_0 : espc_0 =$ locmatch$(aenv; espc)$ or nolocmatch$(aenv; espc)$, but not both.
- Either $\exists espc_1 : espc_1 =$ ctxmatch$(aenv; espc)$ or noctxmatch$(aenv; espc)$, but not both.

Property 38 (Matching in augmented environments is unique)**.**

- If $espc_0 =$ locmatch$(aenv; espc)$ then, $\forall espc_0' :\ espc_0' =$ locmatch$(aenv; espc)$, $espc_0' = espc_0$.
- If $espc_1 =$ ctxmatch$(aenv; espc)$ then, $\forall espc_1' :\ espc_1' =$ ctxmatch$(aenv; espc)$, $espc_1' = espc_1$.

Property 39 (Matching in augmented environments is preserved under mergeability)**.**

- If $espc_0 =$ locmatch$(aenv; espc)$ and $espc' \perp espc$ then, $espc_0 =$ locmatch$(aenv; espc')$.
- If $espc_1 =$ ctxmatch$(aenv; espc)$ and $espc' \perp espc$ then, $espc_1 =$ ctxmatch$(aenv; espc')$.
- If nolocmatch$(aenv; espc)$ and $espc' \perp espc$ then, nolocmatch$(aenv; espc')$.
- If noctxmatch$(aenv; espc)$ and $espc' \perp espc$ then, noctxmatch$(aenv; espc')$.

Lemma 40 (In consistent definition environments, a context match implies a better local match)**.** If $\emptyset \Vdash \tilde{\Phi}$ wf and $aenv^+ = (\tilde{\Phi}; \nu_0; defs)$ and consistent$(aenv^+)$ and ident$(espc) = \nu_0$ and $espc_1 =$ ctxmatch$(aenv^+; espc)$, then $\exists espc_0 :\ espc_0 =$ locmatch$(aenv^+; espc)$ and $espc_0 \leq espc_1$.

Corollary 41. If $\emptyset \Vdash \tilde{\Phi}$ wf and $aenv^+ = (\tilde{\Phi}; \nu_0; defs)$ and consistent$(aenv^+)$ and $espc_0 =$ locmatch$(aenv^+; espc)$ and $espc_1 =$ ctxmatch$(aenv^+; espc)$, then $espc_0 \leq espc_1$.

Corollary 42. If $\emptyset \Vdash \tilde{\Phi}$ wf and $aenv^+ = (\tilde{\Phi}; \nu_0; defs)$ and consistent$(aenv^+)$ and nolocmatch$(aenv^+; espc)$ and $espc_1 =$ ctxmatch$(aenv^+; espc)$, then ident$(espc) \neq \nu_0$.

Definition 43 (Consistent augmented local environments in EL)**.** If ν_0 exists within $\tilde{\Phi}$ then for each specification therein, there is an implementing definition among $defs$.

$$\mathsf{consistent}((\tilde{\Phi}; \nu_0; defs)) \quad \stackrel{\text{def}}{\Leftrightarrow} \quad \begin{cases} \mathsf{nooverlap}(defs), \\ \forall \tilde{dspc} \in \tilde{\Phi}(\nu_0) :\ \exists def \in defs, dspc' : \\ \quad def \sqsubseteq dspc' \ \wedge\ \mathsf{shape}(dspc') \leq \tilde{dspc} \end{cases}$$

10.4 Dependency invariants

Corollary 44 (Module context strengthening for definition checking in image of translation)**.** If $\Phi; \nu_0; \mathit{eenv} \vdash^{\text{IL}}_{\text{c}} \mathit{defs} : \overline{\mathit{dspc}}$ and $\emptyset \vdash \Phi\ \mathsf{wf}$ and $N \supseteq \left(\bigcup_{\mathit{espc} \in \mathit{eenv}} \mathsf{depends}_\Phi(\mathsf{ident}(\mathit{espc})) \cup \{\mathsf{ident}(\mathit{esp}$ then $(\Phi|_N); \nu_0; \mathit{eenv} \vdash^{\text{IL}}_{\text{c}} \mathit{defs} : \overline{\mathit{dspc}}$.

Axiom 45 (Module context strengthening for declaration checking in EL)**.** If $\Phi; \tilde{\tau}; \mathit{eenv} \vdash^{\text{IL}}_{\text{c}} \mathit{decls} : \overline{\mathit{dspc}}$ and $\emptyset \vdash \Phi\ \mathsf{wf}$ and $N \supseteq \left(\bigcup_{\mathit{espc} \in \mathit{eenv}} \mathsf{depends}_\Phi(\mathsf{ident}(\mathit{espc})) \cup \{\mathsf{ident}(\mathit{espc})\}\right)$, then $(\Phi|_N); \tilde{\tau}; \mathit{eenv} \vdash^{\text{IL}}_{\text{c}} \mathit{decls} : \overline{\mathit{dspc}}$.

Lemma 46 (Locally available entities come from dependencies of imports)**.**

- If $\emptyset \vdash (\Phi; \mathcal{L})\ \mathsf{wf}$ and $\mathsf{shape}(\mathcal{L}); \nu_0 \Vdash_{\text{c}} \mathit{impdecls}; \mathit{defs} \rightsquigarrow \mathit{eenv}$, then $\forall \mathit{espc} \in \mathit{eenv} : \mathsf{ident}(\mathit{espc}) \in \{\nu_0\} \cup \left(\bigcup_{\nu' \in \overline{\nu'}} \mathsf{depends}_\Phi(\nu') \cup \{\nu'\}\right)$.
- If $\emptyset \vdash (\Phi; \mathcal{L})\ \mathsf{wf}$ and $\mathsf{shape}(\mathcal{L}) \Vdash_{\text{c}} \mathit{impdecl} \rightsquigarrow \mathit{eenv}$, then $\forall \mathit{espc} \in \mathit{eenv} : \mathsf{ident}(\mathit{espc}) \in \mathsf{depends}_\Phi(\nu') \cup \{\nu'\}$.
- If $\emptyset \vdash (\Phi; \mathcal{L})\ \mathsf{wf}$ and $\mathsf{shape}(\mathcal{L}); p \Vdash_{\text{c}} \mathit{impspec} \rightsquigarrow \overline{\mathit{espc}}$, then $\forall \mathit{espc} \in \overline{\mathit{espc}} : \mathsf{ident}(\mathit{espc}) \in \mathsf{depends}_\Phi(\mathsf{ident}(\mathcal{L}(p))) \cup \{\mathsf{ident}(\mathcal{L}(p))\}$.
- If $\emptyset \vdash (\Phi; \mathcal{L})\ \mathsf{wf}$ and $\mathsf{shape}(\mathcal{L}); p \Vdash_{\text{c}} \mathit{import} \rightsquigarrow \mathit{espc}$, then $\mathsf{ident}(\mathit{espc}) \in \mathsf{depends}_\Phi(\mathsf{ident}(\mathcal{L}(p))) \cup \{\mathsf{ident}(\mathcal{L}(p))\}$.

Lemma 47 (Well-typed modules preserve dependency invariants)**.** If $(\Phi; \mathcal{L}); \nu_0 \vdash_{\text{c}} M : \tau$ and $\emptyset \vdash (\Phi; \mathcal{L})\ \mathsf{wf}$ and $\Phi \perp \nu_0{:}\tau^+$, then $\Phi \oplus \nu_0{:}\tau^+ \vdash \nu_0{:}\tau^+\ \mathsf{deps\text{-}wf}$.

Property 48. If $\nu \in \mathsf{depends}_\Phi(\nu_0)$ then $\mathsf{depends}_\Phi(\nu) \subseteq \mathsf{depends}_\Phi(\nu_0)$. (Proof relies on graph interpretation.)

Property 49 (Antitonicity of dependencies in the context)**.**

- If $\Phi \perp \Phi_{\mathsf{W}}$ and $\nu \in \mathsf{dom}(\Phi)$, then $\mathsf{depends}_\Phi(\nu) \subseteq \mathsf{depends}_{\Phi \oplus \Phi_{\mathsf{W}}}(\nu)$.
- If $\Phi \perp \Phi_{\mathsf{W}}$ and $\nu \in \mathsf{dom}(\Phi)$, then $\mathsf{depends}_{\Phi;N}(\nu) \subseteq \mathsf{depends}_{\Phi \oplus \Phi_{\mathsf{W}};N}(\nu)$.

Property 50 (Preservation of dependencies under substitution)**.** If $\nu \in \mathsf{depends}_\Phi(\nu_0)$ and $\Phi_\theta = \mathsf{apply}(\theta; \Phi)$ is defined, then $\theta\nu \in \mathsf{depends}_{\Phi_\theta}(\theta\nu_0)$. (Proof relies on correspondence of $\mathsf{depends}_\Phi(-)$ to Φ-graph reachability; θ as a graph homomorphism; and graph homomorphism's preservation of connectedness.)

Lemma 51. If $\forall \nu \in \mathsf{dom}(\Phi|_N) : \mathsf{depends}_\Phi(\nu) \subseteq \mathsf{dom}(\Phi|_N)$, then $\forall \nu \in \mathsf{dom}(\Phi|_N) : \mathsf{depends}_\Phi(\nu) = \mathsf{depends}_{\Phi|_N}(\nu)$.

Property 52 (A module's dependencies contain the provenances of its types)**.** If $\Phi(\nu) = \tau$ then $\mathsf{provs}(\tau) \subseteq \mathsf{depends}_\Phi(\nu)$.

11 Algebraic properties of classifiers

11.1 Package signatures

Property 53. Package signatures (Ξ) form an idempotent partial commutative monoid.

- *Idempotence*: $\Xi \perp \Xi$ and $\Xi \oplus \Xi = \Xi$.
- *Commutativity*: If $\Xi_1 \perp \Xi_2$, then $\Xi_2 \perp \Xi_1$ and $\Xi_1 \oplus \Xi_2 = \Xi_2 \oplus \Xi_1$.
- *Associativity*: If $\Xi_1 \perp \Xi_2$ and $\Xi_1 \oplus \Xi_2 \perp \Xi_3$, then $\Xi_2 \perp \Xi_3$ and $\Xi_1 \perp \Xi_2 \oplus \Xi_3$ and $(\Xi_1 \oplus \Xi_2) \oplus \Xi_3 = \Xi_1 \oplus (\Xi_2 \oplus \Xi_3)$.
- *Identity*: $(\emptyset; \emptyset) \perp \Xi$ and $(\emptyset; \emptyset) \oplus \Xi = \Xi$.

11.2 Logical module contexts

Property 54. Logical module contexts ($\mathcal{L}$) form an idempotent partial commutative monoid with identity element ($\emptyset$).

11.3 Physical module contexts

Property 55. Physical module contexts (Φ) form an idempotent partial commutative monoid with identity element ($\emptyset$).

Property 56 (Substitution distributes over context merging)**.**

(1) If $\Phi'_1 = \mathsf{apply}(\theta; \Phi_1)$ is defined and if $\Phi'_2 = \mathsf{apply}(\theta; \Phi_2)$ is defined and $\Phi_1 \perp \Phi_2$ and $\Phi'_1 \perp \Phi'_2$, then $\mathsf{apply}(\theta; \Phi_1 \oplus \Phi_2)$ is defined and equals $\Phi'_1 \oplus \Phi'_2$.

(2) If $\forall i \in [1..n] : \mathsf{apply}(\theta; \Phi_i)$ defined and $\forall i, j \in [1..n] : \Phi_i \perp \Phi_j \wedge \mathsf{apply}(\theta; \Phi_i) \perp \mathsf{apply}(\theta; \Phi_j)$, then $\mathsf{apply}(\theta; \bigoplus_{i \in [1..n]} \Phi_i)$ is defined and equals $\bigoplus_{i \in [1..n]} \mathsf{apply}(\theta; \Phi_i)$.

11.4 Polarized module types

Property 57. Polarized module types (τ^m) form an idempotent partial commutative monoid with identity element $\langle\!|\ \emptyset\ ;\ \emptyset\ ;\ \emptyset\ |\!\rangle^-$.

Lemma 58.

- If $\tau^m = \tau_1^{m_1} \oplus \tau_2^{m_2}$ then $m = m_1 \oplus m_2$ and $\tau = \tau_1 \oplus \tau_2$.
- If $\tau_1^{m_1} \perp \tau_2^{m_2}$ and $m_1 = +$ then $\tau_1^{m_1} \oplus \tau_2^{m_2} = \tau_1^{m_1}$.

11.5 Module types

Property 59. Module types (τ) form an idempotent partial commutative monoid with identity element $\langle\!|\ \emptyset\ ;\ \emptyset\ ;\ \emptyset\ |\!\rangle$.

Property 60. If $\tau = \tau_1 \oplus \tau_2$ then $\mathsf{provs}(\tau) = \mathsf{provs}(\tau_1) \cup \mathsf{provs}(\tau_2)$.

11.6 Entity specifications

Property 61. Sets of entity specifications ($dspcs$) form an idempotent partial commutative monoid with identity element ($\emptyset$).

Property 62. Merging of two entity specifications ($dspc_1 \oplus dspc_2$) is idempotent, associative, and commutative.

11.7 Export specifications

Property 63. Merging of two export specifications ($espc_1 \oplus espc_2$) is idempotent, associative, and commutative.

Property 64. Sets of export specifications ($espcs$) form an idempotent partial commutative monoid with identity element ($\emptyset$).

Property 65 (Substitution distributes over $espc$-set merging)**.**

(1) If $espcs'_1 = \mathsf{apply}(\theta; espcs_1)$ is defined and if $espcs'_2 = \mathsf{apply}(\theta; espcs_2)$ is defined and $espcs_1 \perp espcs_2$ and $espcs'_1 \perp espcs'_2$, then $\mathsf{apply}(\theta; espcs_1 \oplus espcs_2)$ is defined and equals $espcs'_1 \oplus espcs'_2$. (Proof uses "proper" definitions of merge and substitution, as Georg showed on the board.)

(2) If $\forall i \in [1..n] : \mathsf{apply}(\theta; espcs_i)$ defined and $\forall i, j \in [1..n] : espcs_i \perp espcs_j \wedge \mathsf{apply}(\theta; espcs_i) \perp \mathsf{apply}(\theta; espcs_j)$, then $\mathsf{apply}(\theta; \bigoplus_{i\in[1..n]} espcs_i)$ is defined and equals $\bigoplus_{i\in[1..n]} \mathsf{apply}(\theta; espcs_i)$. (Proof uses previous part, Property 72.)

Property 66 (Substitution on non-overlapping *espc*-sets is always defined)**.** If $\mathsf{nooverlap}(espcs)$ then $\mathsf{apply}(\theta; espcs)$ is defined. We often write $\theta espcs$ instead of $\mathsf{apply}(\theta; espcs)$ for non-overlapping $espcs$.

Property 67 (Membership in a merging of *espc*-sets entails similarity to some *espc*)**.**

(1) If $espc \in espcs_1 \oplus espcs_2$ then $\exists i \in \{1, 2\}, espc_i \in espcs_i : espc \perp espc_i$.
(2) If $espc \in \bigoplus_{i\in[1..n]} espcs_i$ then $\exists i \in [1..n], espc_i \in espcs_i : espc \perp espc_i$.

Property 68 (Merge produces a greatest lower bound on *espc*)**.**

(1) If $espc \leq espc_1$ and $espc \leq espc_2$ then $espc \leq espc_1 \oplus espc_2$.
(2) If $\forall i \in [1..n] : espc \leq espc_i$ then $espc \leq \bigoplus_{i\in[1..n]} espc_i$.

11.8 Extra properties of semantic objects

Lemma 69 (Name disjointedness of two merged sets of uniquely-named entities)**.**

- If $\overline{dspc_1} \perp \overline{dspc_2}$ and $\mathsf{nooverlap}(\overline{dspc_1})$ and $\mathsf{nooverlap}(\overline{dspc_2})$, then $\mathsf{nooverlap}(\overline{dspc_1} \oplus \overline{dspc_2}) \Leftrightarrow \forall \chi_1{:}dspc_1 \in \overline{dspc_1}, \chi_2{:}dspc_2 \in \overline{dspc_2}$ s.t. $\chi_1 \neq \chi_2 : \mathsf{allnames}(dspc_1) \# \mathsf{allnames}(dspc_2)$.
- Likewise but with *espc* everywhere instead of *dspc*.

Lemma 70 (Name disjointness of three merged sets of uniquely-named entities)**.**

- If $\mathsf{nooverlap}(\overline{dspc_1} \oplus \overline{dspc_2})$ and $\mathsf{nooverlap}(\overline{dspc_2} \oplus \overline{dspc_3})$ and $\mathsf{nooverlap}(\overline{dspc_2} \oplus \overline{dspc_3})$ and $\forall i \in \{1, 2, 3\} : \mathsf{nooverlap}(\overline{dspc_i})$, then $\mathsf{nooverlap}(\overline{dspc_1} \oplus (\overline{dspc_2} \oplus \overline{dspc_3}))$.
- Likewise but with *espc* everywhere instead of *dspc*.

Property 71 (Partiality of multiple merges)**.**

- If $dspc_1 \perp dspc_2$ and $dspc_2 \perp dspc_3$ and $dspc_1 \perp dspc_3$, then $dspc_1 \perp (dspc_2 \oplus dspc_3)$.
- If $espc_1 \perp espc_2$ and $espc_2 \perp espc_3$ and $espc_1 \perp espc_3$, then $espc_1 \perp (espc_2 \oplus espc_3)$.
- If $dspcs_1 \perp dspcs_2$ and $dspcs_2 \perp dspcs_3$ and $dspcs_1 \perp dspcs_3$, then $dspcs_1 \perp (dspcs_2 \oplus dspcs_3)$.
- If $espcs_1 \perp espcs_2$ and $espcs_2 \perp espcs_3$ and $espcs_1 \perp espcs_3$, then $espcs_1 \perp (espcs_2 \oplus espcs_3)$.
- If $\tau_1 \perp \tau_2$ and $\tau_2 \perp \tau_3$ and $\tau_1 \perp \tau_3$, then $\tau_1 \perp (\tau_2 \oplus \tau_3)$. (Proof also uses Lemma 70.)
- If $\tau_1^{m_1} \perp \tau_2^{m_2}$ and $\tau_2^{m_2} \perp \tau_3^{m_3}$ and $\tau_1^{m_1} \perp \tau_3^{m_3}$, then $\tau_1^{m_1} \perp (\tau_2^{m_2} \oplus \tau_3^{m_3})$.
- If $\Phi_1 \perp \Phi_2$ and $\Phi_2 \perp \Phi_3$ and $\Phi_1 \perp \Phi_3$, then $\Phi_1 \perp (\Phi_2 \oplus \Phi_3)$.

Property 72 (Partiality of multiple merges)**.** If $\bigoplus_{i\in[1..n]} A_i$ defined and $\forall i \in [1..n] : A_i \perp A'$, then $\bigoplus_{i\in[1..n]} A_i \perp A'$, where A ranges over all syntactic categories in previous statement.

Property 73 (Preservation of entity membership by substitution)**.**

- If $\mathsf{apply}(\theta;\Phi)$ and $dspc \in \Phi(\nu)$, then $\exists dspc' \in \mathsf{apply}(\theta;\Phi)(\theta\nu) : dspc' \leq \theta dspc$. Likewise for $espc$.
- If $\mathsf{apply}(\theta; espcs)$ and $espc \in espcs$, then $\exists espc' \in \mathsf{apply}(\theta; espcs) : espc' \leq \theta espc$.

Property 74 (Structure of sums of environments and types)**.**

- If $\bigoplus_{i\in[1..n]} \Phi_i$ is fully defined and $\nu{:}\tau^m \in (\bigoplus_{i\in[1..n]} \Phi_i)$ and $I = \{i \in [1..n] \mid \nu \in \mathsf{dom}(\Phi_i)\}$ and $\{\tau_i^{m_i}\}_{i\in I} = \{(\tau')^{m'} \mid \nu{:}(\tau')^{m'} \in \Phi_i\}_{i\in I}$, then $\tau^m = \bigoplus_{i\in I} (\tau_i^{m_i})$.
- If $\bigoplus_{i\in[1..n]} \tau_i$ is fully defined and $\chi{:}dspc \in (\bigoplus_{i\in[1..n]} \tau_i)$ and $I = \{i \in [1..n] \mid \exists dspc' : \chi{:}dspc' \in \tau_i\}$ and $\{dspc_i\}_{i\in I} = \{dspc' \mid \chi{:}dspc' \in \tau_i\}_{i\in I}$, then $dspc = \bigoplus_{i\in I} dspc_i$.
- Likewise but with $espc$ everywhere instead of $dspc$.

Property 75.

- If $espc_1 \perp espc_2$ then $\mathsf{mkphnms}(espc_1 \oplus espc_2) = \mathsf{mkphnms}(espc_1) \cup \mathsf{mkphnms}(espc_2)$.
- If $\overline{espc_1} \perp \overline{espc_2}$ then $\mathsf{mkphnms}(\overline{espc_1} \oplus \overline{espc_2}) = \mathsf{mkphnms}(\overline{espc_1}) \cup \mathsf{mkphnms}(\overline{espc_2})$.

Lemma 76. If $\tilde{dspc} \sqsubseteq espc$ and $\mathsf{shape}(dspc) \leq \tilde{dspc}$ then $\exists espc' : \ dspc \sqsubseteq espc' \leq espc$.

Lemma 77.

- If $dspc_1 \in \Phi_1(\nu)$ and $\Phi_1 \perp \Phi_2$ then $\exists dspc_2 : dspc_1 \oplus dspc_2 \in (\Phi_1 \oplus \Phi_2)(\nu)$.
- Likewise for $espc$.

Lemma 78. If $dspc_1 \sqsubseteq espc_2$ and $dspc_1 \perp dspc_2$ then $\exists espc_2 : dspc_2 \sqsubseteq espc_2$.

Property 79 (Monotonicity of spec matching)**.**

(1) If $dspc_1 \sqsubseteq espc_1$ and $dspc_2 \sqsubseteq espc_2$ and $dspc_1 \perp dspc_2$, then $espc_1 \perp espc_2$ and $dspc_1 \oplus dspc_2 \sqsubseteq espc_1 \oplus espc_2$.

(2) If $dspc = \bigoplus_{i\in I} dspc_i$ and $espc = \bigoplus_{i\in I} espc_i$ and $\forall i \in I : \ dspc_i \sqsubseteq espc_i$, then $dspc \sqsubseteq espc$.

Property 80 (Monotonicity of export specs)**.**

(1) If $espc_1 \leq espc_1'$ and $espc_2 \leq espc_2'$ and $espc_1 \perp espc_2$, then $espc_1' \perp espc_2'$ and $espc_1 \oplus espc_2 \leq espc_1' \oplus espc_2'$.

(2) If $espc = \bigoplus_{i\in I} espc_i$ and $espc' = \bigoplus_{i\in I} espc_i'$ and $\forall i \in I : \ espc_i \leq espc_i'$, then $espc \leq espc'$.

Lemma 81. If $\Phi \perp \nu{:}\tau^+$ then $\Phi \perp \nu{:}\tau^m$ and $(\Phi \oplus \nu{:}\tau^m)(\nu) = \tau$.

Property 82 (Substitution on singleton maps indexed by identities)**.**

- If $\Phi = \nu{:}\tau^m$ then $\mathsf{apply}(\theta;\Phi)$ is defined and equals $(\theta\nu){:}(\theta\tau)^m$.
- If $espcs = (espc)$ then $\mathsf{apply}(\theta; espcs)$ is defined and equals $(\theta espc)$.

(All proofs straightforward by the definition of substitution in terms of merging (over a singleton index set).)

Part III
Internal language metatheory

12 Judgmental properties of IL terms

12.1 Strengthening

Lemma 83 (Preservation of well-formedness under context strengthening)**.** Suppose $\forall f \in \mathsf{dom}(\mathit{fenv}) \cap F : \mathsf{depends}_{\mathit{fenv}}(f) \subseteq \mathsf{dom}(\mathit{fenv}) \cap F$.

(1) If $\mathit{fenv} \vdash^{\text{IL}}_{\text{c}} \mathit{hsmod} : \mathit{ftyp}$ and $\mathit{fenv}(f_0) = \mathit{ftyp}$ and $f_0 \in F$, then $\mathit{fenv}|_F \vdash^{\text{IL}}_{\text{c}} \mathit{hsmod} : \mathit{ftyp}$. (Proof uses Corollary 85 to show F is suitable for Axiom 96; defn of $\mathsf{depends}_{\mathit{fenv}}(f_0)$ to show F suitably large for imports and Lemma 97 for strengthening of imports.)

(2) If $\mathit{fenv}; f_0 \vdash \mathit{tfexp}$ and $\mathit{fenv}(f_0) = \mathsf{typ}(\mathit{tfexp})$ and $f_0 \in F$, then $\mathit{fenv}|_F; f_0 \vdash \mathit{tfexp}$. (Proof uses Property 52 and τ part of Lemma 6.)

(3) If $\emptyset \vdash \mathit{fenv}\ \mathsf{wf}$ then $\emptyset \vdash \mathit{fenv}|_F\ \mathsf{wf}$. (This is no different from the analogous statement on the EL, Lemma 6.)

(4) If $\emptyset \vdash \mathit{dexp}$ and $\mathit{fenv} = \mathsf{mkfenv}(\mathit{dexp})$, then $\emptyset \vdash \mathit{dexp}|_F$. (Proof uses filtering on *fenv*-wf and previous part for *fenv* wf; Property 109 and *tfexp* part for *tfexp* wf.)

Lemma 84 (Locally available entities come from dependencies of imports (IL))**.** Directly corresponds to the analogous lemma in the EL, Lemma 46. Suppose $\emptyset \vdash \mathit{fenv}\ \mathsf{wf}$

(1) If $\mathit{fenv}; f \vdash^{\text{IL}}_{\text{c}} \check{\mathit{import}} \rightsquigarrow \check{\mathit{espc}}$, then $\mathsf{ident}(\check{\mathit{espc}}) \in \mathsf{depends}_{\mathit{fenv}}(f) \cup \{f\}$. (Proof uses $\mathit{fenv} \vdash \mathit{fenv}\ \mathsf{deps\text{-}wf}$ and $\mathsf{ident}(\check{\mathit{espc}}) \in \mathsf{provs}(\mathit{fenv}(f))$.)

(2) If $\mathit{fenv}; f \vdash^{\text{IL}}_{\text{c}} \check{\mathit{impspec}} \rightsquigarrow \check{\mathit{espcs}}$, then $\forall \check{\mathit{espc}} \in \check{\mathit{espcs}} : \mathsf{ident}(\check{\mathit{espc}}) \in \mathsf{depends}_{\mathit{fenv}}(f) \cup \{f\}$. (Proof uses Property 67 and previous part.)

(3) If $\mathit{fenv} \vdash^{\text{IL}}_{\text{c}} \check{\mathit{impdecl}} \rightsquigarrow \check{\mathit{eenv}}$, then $\forall \check{\mathit{espc}} \in \check{\mathit{espcs}} : \mathsf{ident}(\check{\mathit{espc}}) \in \mathsf{depends}_{\mathit{fenv}}(\mathsf{imp}(\check{\mathit{impdecl}})) \cup \{\mathsf{imp}(\check{\mathit{impdecl}})\}$. (Proof straightforward by previous part and definition of $\mathsf{mkeenv}(-)$.)

(4) If $\mathit{fenv}; f_0 \vdash^{\text{IL}}_{\text{c}} \check{\mathit{impdecls}}; \check{\mathit{defs}} \rightsquigarrow \check{\mathit{eenv}}$, then $\forall \check{\mathit{espc}} \in \check{\mathit{eenv}} : \mathsf{ident}(\mathit{espc}) \in \{f_0\} \cup \left(\bigcup_{f' \in \mathsf{imps}(\check{\mathit{impdecls}})} \mathsf{depends}_{\mathit{fenv}}(f') \cup \{f'\}\right)$. (Proof uses Property 102 and previous part and definition of $\mathsf{mklocaleenv}(f_0; -)$.)

Corollary 85 (Dependencies of this mod contains those of locally available entities)**.** If $\emptyset \vdash \mathit{fenv}\ \mathsf{wf}$ and $\mathit{fenv}; f_0 \vdash^{\text{IL}}_{\text{c}} \check{\mathit{impdecls}}; \check{\mathit{defs}} \rightsquigarrow \check{\mathit{eenv}}$ and $\mathsf{imps}(\check{\mathit{impdecls}}) = \mathsf{imps}(\mathit{fenv}(f_0))$, then $\forall \check{\mathit{espc}} \in \check{\mathit{eenv}} : \mathsf{depends}_{\mathit{fenv}}(\mathsf{ident}(\check{\mathit{espc}})) \cup \{\mathsf{ident}(\check{\mathit{espc}})\} \subseteq \mathsf{depends}_{\mathit{fenv}}(f_0) \cup \{f_0\}$. (Proof splits into cases whether $\mathsf{ident}(\mathit{espc}) = f_0$, using Lemma 84 and Property 48 in the case that it's not f_0.)

12.2 Substitutability

Lemma 86 (Invariance under substitution of IL terms)**.** Suppose $\emptyset \vdash \mathit{fenv}\ \mathsf{wf}$ and $\mathit{fenv}_\theta = \mathsf{apply}(\theta; \mathit{fenv})$ is defined.

(1) If $\mathit{fenv} \vdash^{\text{IL}}_{\text{c}} \mathit{hsmod} : \mathit{ftyp}$ and $\mathit{fenv}_\theta \perp (\theta f_0){:}\theta\mathit{ftyp}^+$ and $\mathsf{validsubst}(\theta; \mathit{hsmod})$, where $f_0 = \mathsf{name}(\mathit{hsmod})$, then $\mathit{fenv}_\theta \vdash^{\text{IL}}_{\text{c}} \theta\mathit{hsmod} : \theta\mathit{ftyp}$. (Proof uses Lemma 87 and Lemma 99 for import resolution; Lemma 104 for export

resolution; Axiom 98 and tricky stuff on whiteboard about valid subst for definition checking.)

(2) If $fenv; f_0 \vdash tfexp$ and $fenv_\theta \perp (\theta f_0){:}(\theta ftyp)^m$ and $\mathsf{validsubst}(\theta; tfexp)$, where $ftyp = \mathsf{typ}(tfexp)$ and $m = \mathsf{pol}(tfexp)$, then $fenv_\theta; \theta f_0 \vdash \theta tfexp$. (Proof uses above part and invariance for τ well-formedness Lemma 5.)

(3) If $\mathsf{apply}(\theta; dexp)$ is defined and $fenv_\theta \perp \mathsf{apply}(\theta; dexp)$ and $fenv \vdash dexp$ and $\mathsf{validsubst}(\theta; dexp)$, then $fenv_\theta \vdash \mathsf{apply}(\theta; dexp)$. (Proof uses Lemma 112 and invariance on *fenv*-wf and Lemma 112 for wf-ness of the *dexp*'s file env; Property 108 and Lemma 88 and invariance on *tfexp* and merging on *tfexp* for typing of files.)

Lemma 87 (Consistent augmented environments come from mergeable modules (IL))**.** If $\overline{\check{def} \sqsubseteq \check{dspc}}$ and $\overline{\check{dspc}} \in ftyp$ and $fenv \perp f_0{:}ftyp^+$ [and $\mathsf{nooverlap}(\overline{\check{def}})$, then $\mathsf{consistent}(\check{aenv})$, where $\check{aenv} = (fenv; f_0; \check{defs})$. (Proof very straightforward; uses Lemma 58.)

Lemma 88. If $\mathsf{apply}(\theta; fenv_1)$ is defined and $\mathsf{apply}(\theta; fenv_2)$ is defined and $\mathsf{apply}(\theta; fenv_1) \perp \mathsf{apply}(\theta; fenv_2)$, then $\mathsf{apply}(\theta; fenv_1 \oplus fenv_2)$ is defined and equals $\mathsf{apply}(\theta; fenv_1) \oplus \mathsf{apply}(\theta; fenv_2)$ and $\forall f{:}ftyp^m \in fenv_2 : \mathsf{apply}(\theta; fenv_1) \perp (\theta f){:}(\theta ftyp)^m$. (First part is just Property 56; second part uses the fact that substitution on contexts is just the merge of all substituted singletons, and that merge is commutative and associative.)

Definition 89 (Consistent augmented local environment in IL)**.** Analogous to EL definition.

$$\mathsf{consistent}((fenv; f_0; \check{defs})) \stackrel{\text{def}}{\Leftrightarrow} \begin{cases} \mathsf{nooverlap}(\check{defs}), \\ \forall \check{dspc} \in fenv(f_0) : \exists \check{def} \in \check{defs}, \check{dspc}' : \\ \quad \check{def} \sqsubseteq \check{dspc}' \ \wedge \ \check{dspc}' \leq \check{dspc} \end{cases}$$

Definition 90 (Valid substitutions in IL)**.** Analogous to EL definition. Make sure that the substitution entirely avoids the logical import aliases in all files.

$$\mathsf{validsubst}(\theta; dexp) \stackrel{\text{def}}{\Leftrightarrow} \begin{cases} \mathsf{apply}(\theta; dexp) \text{ defined} \\ \forall (f \mapsto tfexp) \in dexp : \ \mathsf{validsubst}(\theta; tfexp) \end{cases}$$

$$\mathsf{validsubst}(\theta; - : ftyp) \stackrel{\text{def}}{\Leftrightarrow} \text{always}$$

$$\mathsf{validsubst}(\theta; hsmod : ftyp) \stackrel{\text{def}}{\Leftrightarrow} \begin{cases} \forall f \in \mathsf{aliases}(hsmod) : \ \theta f = f \\ \forall f \in hsmod : \ \theta f \neq f \ \Rightarrow \ \theta f \notin \mathsf{aliases}(hsmod) \end{cases}$$

12.3 Weakening

Lemma 91 (Weakening on IL term judgments)**.** Suppose $fenv \perp fenv_{\mathsf{W}}$.

(1) If $fenv \vdash^{\text{IL}}_{\text{c}} hsmod : ftyp$ then $fenv \oplus fenv_{\mathsf{W}} \vdash^{\text{IL}}_{\text{c}} hsmod : ftyp$. (Proof uses Axiom 94 and Lemma 95.)

(2) If $fenv; f_0 \vdash tfexp$ then $fenv \oplus fenv_{\mathsf{W}}; f_0 \vdash tfexp$. (Proof uses weakening on *hsmod* and on *ftyp*/*tau* wf.)

(3) If $fenv \vdash dexp$ then $fenv \oplus fenv_{\mathsf{W}} \vdash dexp$.

12.4 Merging

Lemma 92 (Merging on IL term judgments)**.** Suppose $fenv_1 \perp fenv_2$.

(1) If $fenv_1; f_0 \vdash tfexp_1$ and $fenv_1; f_0 \vdash tfexp_1$ and $tfexp_1 \perp tfexp_2$, then $fenv_1 \oplus fenv_2; f_0 \vdash tfexp_1 \oplus tfexp_2$. (Proof uses weakening on *tfexp* and merging on $ftyp/\tau$-wf.)

(2) Merging on *dexp* is not necessary.

12.5 Misc

Lemma 93 (Well-formedness of a directory of signatures)**.** If $fenv \vdash fenv'$ wf and $fenv' \preceq -$ and $fenv' = \mathsf{mkfenv}(dexp)$, then $fenv \vdash dexp$. (Proof straightforward application of Lemma 10.)

13 Judgmental properties of auxiliary IL module machinery

13.1 Weakening

Axiom 94 (Weakening on core definition checking)**.** If $fenv; f_0; ee\check{n}v \vdash^{\text{IL}}_{\text{c}} d\check{e}fs : ds\check{p}cs$ and $fenv \perp fenv_{\mathsf{W}}$, then $fenv \oplus fenv_{\mathsf{W}}; f_0; ee\check{n}v \vdash^{\text{IL}}_{\text{c}} d\check{e}fs : ds\check{p}cs$.

Lemma 95 (Weakening of import judgments)**.** Suppose $fenv \perp fenv_{\mathsf{W}}$.

(1) If $fenv; f \vdash^{\text{IL}}_{\text{c}} im\check{p}ort \rightsquigarrow es\check{p}c$ then $fenv \oplus fenv_{\mathsf{W}}; f \vdash^{\text{IL}}_{\text{c}} im\check{p}ort \rightsquigarrow es\check{p}c$. (Proof uses Lemma 77.)

(2) If $fenv; f \vdash^{\text{IL}}_{\text{c}} im\check{p}spec \rightsquigarrow es\check{p}cs$ then $fenv \oplus fenv_{\mathsf{W}}; f \vdash^{\text{IL}}_{\text{c}} im\check{p}spec \rightsquigarrow es\check{p}cs$.

(3) If $fenv \vdash^{\text{IL}}_{\text{c}} im\check{p}decl \rightsquigarrow ee\check{n}v$ then $fenv \oplus fenv_{\mathsf{W}} \vdash^{\text{IL}}_{\text{c}} im\check{p}decl \rightsquigarrow ee\check{n}v$.

(4) If $fenv; f_0 \vdash^{\text{IL}}_{\text{c}} im\check{p}decls; d\check{e}fs \rightsquigarrow ee\check{n}v$ then $fenv \oplus fenv_{\mathsf{W}}; f_0 \vdash^{\text{IL}}_{\text{c}} im\check{p}decls; d\check{e}fs \rightsquigarrow ee\check{n}v$.

13.2 Strengthening

Axiom 96 (Module context strengthening for definition checking in IL)**.** If $fenv; f_0; ee\check{n}v \vdash^{\text{IL}}_{\text{c}} d\check{e}fs : \overline{ds\check{p}c}$ and $\emptyset \vdash fenv$ wf and $F \supseteq \left(\bigcup_{es\check{p}c \in ee\check{n}v} \mathsf{depends}_{fenv}(\mathsf{ident}(es\check{p}c)) \cup \{\mathsf{ident}(es\check{p}c)\}\right)$, then $(fenv|_F); f_0; ee\check{n}v \vdash^{\text{IL}}_{\text{c}} d\check{e}fs : \overline{ds\check{p}c}$.

Lemma 97 (Strengthening of import judgments)**.**

(1) If $f \in F$ and $fenv; f \vdash^{\text{IL}}_{\text{c}} im\check{p}ort \rightsquigarrow es\check{p}c$, then $fenv|_F \vdash^{\text{IL}}_{\text{c}} im\check{p}ort \rightsquigarrow es\check{p}c$.

(2) If $f \in F$ and $fenv; f \vdash^{\text{IL}}_{\text{c}} im\check{p}spec \rightsquigarrow es\check{p}cs$, then $fenv|_F; f \vdash^{\text{IL}}_{\text{c}} im\check{p}spec \rightsquigarrow es\check{p}cs$.

(3) If $\mathsf{imp}(im\check{p}decl) \in F$ and $fenv \vdash^{\text{IL}}_{\text{c}} im\check{p}decl \rightsquigarrow ee\check{n}v$, then $fenv|_F \vdash^{\text{IL}}_{\text{c}} im\check{p}decl \rightsquigarrow ee\check{n}v$.

(4) If $\mathsf{imps}(im\check{p}decls) \subseteq F$ and $fenv; f_0 \vdash^{\text{IL}}_{\text{c}} im\check{p}decls; d\check{e}fs \rightsquigarrow ee\check{n}v$, then $fenv|_F; f_0 \vdash^{\text{IL}}_{\text{c}} im\check{p}decls; d\check{e}fs \rightsquigarrow ee\check{n}v$. (All proofs very straightforward.)

13.3 Substitutability

Axiom 98 (Invariance under substitution of core definition checking)**.** If $\emptyset \vdash$ *fenv* wf and $\mathsf{apply}(\theta; fenv)$ is defined and $fenv; f_0; ee\check{n}v \vdash^{\text{IL}}_{\text{c}} d\check{e}fs : d\check{s}pc$ and $\mathsf{validsubst}(\theta; ee\check{n}v)$, then $\mathsf{apply}(\theta; fenv); \theta f_0; \mathsf{apply}(\theta; ee\check{n}v) \vdash^{\text{IL}}_{\text{c}} \theta d\check{e}fs : \theta d\check{s}pc$.

Lemma 99 (Invariance under substitution of IL import resolution)**.** Suppose $\emptyset \vdash$ *fenv* wf and $fenv_\theta = \mathsf{apply}(\theta; fenv)$ is defined.

(1) If $fenv; f \vdash^{\text{IL}}_{\text{c}} im\check{p}ort \rightsquigarrow e\check{s}pc$, then $fenv_\theta; \theta f \vdash^{\text{IL}}_{\text{c}} \theta im\check{p}ort \rightsquigarrow \theta e\check{s}pc$.

(2) If $fenv; f \vdash^{\text{IL}}_{\text{c}} imp\check{s}pec \rightsquigarrow e\check{s}pcs$, then $\mathsf{apply}(\theta; e\check{s}pcs)$ defined and $fenv_\theta; \theta f \vdash^{\text{IL}}_{\text{c}} \theta imp\check{s}pec \rightsquigarrow \mathsf{apply}(\theta; e\check{s}pcs)$. (Proof uses Lemma 100 and Property 66 for definedness.)

(3) If $fenv \vdash^{\text{IL}}_{\text{c}} im\check{p}decl \rightsquigarrow ee\check{n}v$, then $\mathsf{apply}(\theta; ee\check{n}v)$ defined and $fenv_\theta \vdash^{\text{IL}}_{\text{c}} \theta im\check{p}decl \rightsquigarrow \mathsf{apply}(\theta; ee\check{n}v)$. (Proof uses Property 66 and Property 105 and Property 22 for definedness/distributivity of substituted, merged envs.)

(4) If $fenv; f_0 \vdash^{\text{IL}}_{\text{c}} im\check{p}decls; d\check{e}fs \rightsquigarrow ee\check{n}v$ and $\mathsf{consistent}(\theta fenv; \theta f_0; \theta d\check{e}fs)$, then $\mathsf{apply}(\theta; ee\check{n}v)$ defined and $fenv_\theta; \theta f_0 \vdash^{\text{IL}}_{\text{c}} \theta im\check{p}decls; \theta d\check{e}fs \rightsquigarrow \mathsf{apply}(\theta; ee\check{n}v)$. (Proof uses Lemma 100 and Lemma 103(1) and Property 65 for definedness/distributivity of substituted, merged import envs; Property 105 for definedness/distributivity of substituted local env; Lemma 101 and Lemma 103(2) and Property 65 for definedness/distributivity of substituted, merged import and local env.)

Lemma 100 (Imports produce non-overlapping *espcs* that match the context)**.** Suppose $\emptyset \vdash$ *fenv* wf and $ae\check{n}v = (fenv; f_0; d\check{e}fs)$.

(1) If $fenv; f \vdash^{\text{IL}}_{\text{c}} im\check{p}ort \rightsquigarrow e\check{s}pc$, then $\mathsf{ctxmatch}(ae\check{n}v; e\check{s}pc)$ and $\exists e\check{s}pc' \in fenv(f) : e\check{s}pc' \leq e\check{s}pc$.

(2) If $fenv; f \vdash^{\text{IL}}_{\text{c}} imp\check{s}pec \rightsquigarrow e\check{s}pcs$, then $\mathsf{nooverlap}(e\check{s}pcs)$ and $\forall espc \in e\check{s}pcs : \mathsf{ctxmatch}(aenv; espc)$.

(3) If $fenv \vdash^{\text{IL}}_{\text{c}} im\check{p}decl \rightsquigarrow ee\check{n}v$, then $\mathsf{nooverlap}(\mathsf{locals}(ee\check{n}v))$ and $\forall espc \in ee\check{n}v : \mathsf{ctxmatch}(aenv; espc)$ and $\forall e\check{r}ef \in \mathsf{dom}(ee\check{n}v) : e\check{r}ef = \chi$ or $\mathsf{alias}(im\check{p}decl).\chi$. (Proof uses Property 21 for last result.)

Lemma 101 (Entities in local entity environment produce local matches)**.** $\forall e\check{s}pc \in \mathsf{mklocaleenv}(f_0; d\check{e}fs), \mathsf{locmatch}((fenv; f_0; d\check{e}fs); e\check{s}pc)$.

Property 102 (Membership of an *espc* in a merging of entity envs)**.** Straightforward corollary of Property 67, by definition of $\oplus$ on entity envs.

(1) If $e\check{s}pc \in ee\check{n}v_1 \oplus ee\check{n}v_2$ then $\exists i \in \{1, 2\}, e\check{s}pc_i \in ee\check{n}v_i : e\check{s}pc \perp e\check{s}pc_i$.

(2) If $e\check{s}pc \in \bigoplus_{i \in [1..n]} ee\check{n}v_i$ then $\exists i \in [1..n], e\check{s}pc_i \in ee\check{n}v_i : e\check{s}pc \perp e\check{s}pc_i$.

Corollary 103 (Definedness of substitution on merges of entity envs)**.** Suppose $\emptyset \vdash$ *fenv* wf and $\mathsf{apply}(\theta; fenv)$ defined and $aenv = (fenv; f_0; defs)$.

(1) If $\forall espc_1 \in ee\check{n}v_1 : \mathsf{ctxmatch}(aenv; espc_1)$ and $\forall espc_2 \in ee\check{n}v_2 : \mathsf{ctxmatch}(aenv; espc_2)$, then $\mathsf{apply}(\theta; ee\check{n}v_1)$ defined and $\mathsf{apply}(\theta; ee\check{n}v_2)$ defined and $\mathsf{apply}(\theta; ee\check{n}v_1) \perp \mathsf{apply}(\theta; ee\check{n}v_2)$.

(2) If $\mathsf{consistent}(\mathsf{apply}(\theta; aenv))$ and $\forall espc_1 \in ee\check{n}v_1 : \mathsf{ctxmatch}(aenv; espc_1)$ and $\forall espc_2 \in ee\check{n}v_2 : \mathsf{locmatch}(aenv; espc_2)$, then $\mathsf{apply}(\theta; ee\check{n}v_1)$ defined and $\mathsf{apply}(\theta; ee\check{n}v_2)$ defined and $\mathsf{apply}(\theta; ee\check{n}v_1) \perp \mathsf{apply}(\theta; ee\check{n}v_2)$.

Lemma 104 (Invariance under substitution of IL export resolution)**.** Suppose $\mathsf{validsubst}(\theta; ee\check{n}v)$.

(1) If $f_0; ee\check{n}v \vdash^{\text{IL}}_{\text{c}} ex\check{p}ort \rightsquigarrow es\check{p}c$, then $\theta f_0; \mathsf{apply}(\theta; ee\check{n}v) \vdash^{\text{IL}}_{\text{c}} \theta ex\check{p}ort \rightsquigarrow \theta es\check{p}c$. (Proof uses Property 73.)

(2) If $f_0; ee\check{n}v \vdash^{\text{IL}}_{\text{c}} ex\check{p}decl \rightsquigarrow es\check{p}cs$, then $\mathsf{apply}(\theta; es\check{p}cs)$ defined and $\theta f_0; \mathsf{apply}(\theta; ee\check{n}v) \vdash^{\text{IL}}_{\text{c}} \theta ex\check{p}decl \rightsquigarrow \mathsf{apply}(\theta; es\check{p}cs)$. (Proof uses Property 66 for definedness of specs; Property 65 for definedness/distributivity of substituted specs.)

Property 105 (Distributivity of substitution over entity environment operations)**.**

- If $\mathsf{apply}(\theta; es\check{p}cs)$ is defined then $\mathsf{apply}(\theta; \mathsf{mkeenv}(es\check{p}cs))$ is defined and equals $\mathsf{mkeenv}(\mathsf{apply}(\theta; es\check{p}cs))$.
- If $\mathsf{apply}(\theta; es\check{p}cs)$ is defined then $\mathsf{apply}(\theta; \mathsf{mklocaleenv}(f_0; d\check{e}fs))$ is defined and equals $\mathsf{mklocaleenv}(\theta f_0; \theta d\check{e}fs)$.
- If $\mathsf{apply}(\theta; ee\check{n}v)$ is defined then $\mathsf{apply}(\theta; \mathsf{qualify}(f; ee\check{n}v))$ is defined and equals $\mathsf{qualify}(\theta f; \mathsf{apply}(\theta; ee\check{n}v))$.

Lemma 106 (Definedness of substitution on merges of contextual and local *espc*-sets)**.** Suppose $\emptyset \vdash fenv\ \mathsf{wf}$ and $\mathsf{apply}(\theta; fenv)$ defined and $aenv = (fenv; f_0; defs)$.

(1) If $\forall espc_1 \in espcs_1 : \mathsf{ctxmatch}(aenv; espc_1)$ and $\forall espc_2 \in espcs_2 : \mathsf{ctxmatch}(aenv; espc_2)$, then $\mathsf{apply}(\theta; espcs_1)$ defined and $\mathsf{apply}(\theta; espcs_2)$ defined and $\mathsf{apply}(\theta; espcs_1) \perp \mathsf{apply}(\theta; espcs_2)$.

(2) If $\mathsf{consistent}(\mathsf{apply}(\theta; aenv))$ and $\forall espc_1 \in espcs_1 : \mathsf{ctxmatch}(aenv; espc_1)$ and $\forall espc_2 \in espcs_2 : \mathsf{locmatch}(aenv; espc_2)$, then $\mathsf{apply}(\theta; espcs_1)$ defined and $\mathsf{apply}(\theta; espcs_2)$ defined and $\mathsf{apply}(\theta; espcs_1) \perp \mathsf{apply}(\theta; espcs_2)$.

Definition 107 (Valid substitutions for an entity environment)**.** The substitution preserves lookup in the entity environment.

$$\mathsf{validsubst}(\theta; ee\check{n}v) \quad \stackrel{\text{def}}{\Leftrightarrow} \quad \begin{cases} \mathsf{apply}(\theta; ee\check{n}v) \text{ defined} \\ \forall er\check{e}f, ph\check{n}m \text{ s.t. } ee\check{n}v(er\check{e}f) = ph\check{n}m : \\ \quad \mathsf{apply}(\theta; ee\check{n}v)(\theta er\check{e}f) = \theta ph\check{n}m \end{cases}$$

14 Algebraic properties of IL terms

Property 108 (Structure of sums of directory expressions)**.**

- If $\bigoplus_{i\in[1..n]} dexp_i$ is fully defined and $(f \mapsto tfexp) \in (\bigoplus_{i\in[1..n]} dexp_i)$ and $I = \{i \in [1..n] \mid f \in \mathsf{dom}(dexp_i)\}$ and $\{tfexp_i\}_{i\in I} = \{tfexp' \mid f \mapsto tfexp' \in \Phi_i\}_{i\in I}$, then $tfexp = \bigoplus_{i\in I} tfexp_i$.

Property 109 (Projections from directory environments)**.** If $fenv = \mathsf{mkfenv}(dexp)$ and $dexp(f) = tfexp$, then $fenv(f) = \mathsf{typ}(tfexp)$.

Property 110 (Typed file expressions are mergeable when their polarized types are mergeable)**.**

(1) $tfexp_1 \perp tfexp_2$ if and only if $\mathsf{typ}(tfexp_1)^{\mathsf{pol}(tfexp_1)} \perp \mathsf{typ}(tfexp_2)^{\mathsf{pol}(tfexp_2)}$.

(2) $\bigoplus_{i\in[1..n]} tfexp_i$ is defined if and only if $\bigoplus_{i\in[1..n]} \mathsf{typ}(tfexp_i)^{\mathsf{pol}(tfexp_i)}$ is defined.

(Proofs straightforward from definition of *tfexp* merging.)

Property 111 (Merging of directories commutes with file environment creation)**.**

(1) $dexp_1 \perp dexp_2$ if and only if $\mathsf{mkfenv}(dexp_1) \perp \mathsf{mkfenv}(dexp_2)$, and if so, then $\mathsf{mkfenv}(dexp_1) \oplus \mathsf{mkfenv}(dexp_2) = \mathsf{mkfenv}(dexp_1 \oplus dexp_2)$.

(2) $\bigoplus_{i\in[1..n]} dexp_i$ is defined if and only if $\bigoplus_{i\in[1..n]} \mathsf{mkfenv}(dexp_i)$ is defined, and if so, then $(\bigoplus_{i\in[1..n]} \mathsf{mkfenv}(dexp_i)) = \mathsf{mkfenv}(\bigoplus_{i\in[1..n]} dexp_i)$.

(Proofs straightforward from Property 110.)

Property 112 (Substitution on *dexp* commutes with $\mathsf{mkfenv}(-)$)**.** $\mathsf{apply}(\theta; \mathsf{mkfenv}(dexp))$ is defined if and only if $\mathsf{apply}(\theta; dexp)$ is defined, and if so, then $\mathsf{apply}(\theta; \mathsf{mkfenv}(dexp)) = \mathsf{mkfenv}(\mathsf{apply}(\theta; dexp))$. (Proof straightforward from Property 111.)

Property 113 (Distributivity of merging and substitution over *tfexp* attributes)**.**

- If $tfexp_1 \perp tfexp_2$ then $\mathsf{typ}(tfexp_1 \oplus tfexp_2) = \mathsf{typ}(tfexp_1) \oplus \mathsf{typ}(tfexp_2)$.
- If $tfexp_1 \perp tfexp_2$ then $\mathsf{pol}(tfexp_1 \oplus tfexp_2) = \mathsf{pol}(tfexp_1) \oplus \mathsf{pol}(tfexp_2)$.
- Likewise for n-ary merges over a non-empty index set I.
- $\mathsf{typ}(\theta tfexp) = \theta\mathsf{typ}(tfexp)$

Property 114 (Substitution on singleton directory expressions)**.** If $dexp = \{f \mapsto tfexp\}$ then $\mathsf{apply}(\theta; dexp)$ is defined and equals $\{(\theta f) \mapsto (\theta tfexp)\}$. (Proofs straightforward by the definition of substitution in terms of merging (over a singleton index set).)

Part IV
Elaboration metatheory

15 Invariants needed for elaboration soundness

15.1 Import resolution

Lemma 115 (Elaboration of imports produces translations of original entity envs)**.** Suppose $\emptyset \vdash (\Phi; \mathcal{L})\ \mathsf{wf}$.

(1) If $\mathsf{shape}(\mathcal{L}); p \Vdash_{\mathrm{c}} import \rightsquigarrow espc$ then $\Phi^\star; \mathcal{L}^\star(p) \vdash^{\mathrm{IL}}_{\mathrm{c}} \mathsf{mkentimp}(espc) \rightsquigarrow espc^\star$.

(2) If $\mathsf{shape}(\mathcal{L}); p \Vdash_{\mathrm{c}} impspec \rightsquigarrow espcs$ then $\Phi^\star; \mathcal{L}^\star(p) \vdash^{\mathrm{IL}}_{\mathrm{c}} \mathsf{mkimpspec}(\mathcal{L}; p; impspec) \rightsquigarrow espcs^\star$.

(3) If $\mathsf{shape}(\mathcal{L}) \Vdash_{\mathrm{c}} impdecl \rightsquigarrow eenv$ then $eenv^\star$ defined and $\Phi^\star \vdash^{\mathrm{IL}}_{\mathrm{c}} \mathsf{mkimpdecl}(\mathcal{L}; impdecl) \rightsquigarrow eenv^\star$. (Proof uses Property 116.)

(4) If $\mathsf{shape}(\mathcal{L}); \nu_0 \Vdash_{\mathrm{c}} impdecl_1, \ldots, impdecl_n; defs \rightsquigarrow eenv$ then $\Phi^\star; \nu_0^\star \vdash^{\mathrm{IL}}_{\mathrm{c}} \mathsf{mkimpdecls}(\mathcal{L}; impdecls); \mathsf{refs}^\star_{\nu_0}(defs) \rightsquigarrow \mathsf{refs}^\star_{\nu_0}(eenv)$. (Proof uses previous part and Property 116.)

Property 116 (Distributivity of translation over entity env operations)**.**

- $\mathsf{mkeenv}(espcs)^\star$ is defined and equals $\mathsf{mkeenv}(espcs^\star)$.
- If $\mathsf{qualify}(p; eenv)$ is defined, then $eenv^\star$ is defined, and $\mathsf{qualify}(p; eenv)^\star$ is defined and equals $\mathsf{qualify}(p; eenv^\star)$.

- $\mathsf{refs}^\star_{\nu_0}(\mathsf{mklocaleenv}(\nu_0; \mathit{defs})) = \mathsf{mklocaleenv}(\nu_0^\star; \mathsf{refs}^\star_{\nu_0}(\mathit{defs}))$.
- $\mathsf{locals}(\mathit{eenv})^\star = \mathsf{locals}(\mathit{eenv}^\star)$.
- If $\mathit{eenv}_1 \perp \mathit{eenv}_2$, then $\mathsf{refs}^\star_{\nu_0}(\mathit{eenv}_1) \oplus \mathsf{refs}^\star_{\nu_0}(\mathit{eenv}_2)$ is defined and equals $\mathsf{refs}^\star_{\nu_0}(\mathit{eenv}_1 \oplus \mathit{eenv}_2)$.
- If $\forall i, j \in [1..n] : \mathit{eenv}_i \perp \mathit{eenv}_j$, then $\bigoplus_{i\in[1..n]} \mathsf{refs}^\star_{\nu_0}(\mathit{eenv}_i)$ is defined and equals $\mathsf{refs}^\star_{\nu_0}(\bigoplus_{i\in[1..n]} \mathit{eenv}_i)$.

15.2 Core definition checking

Lemma 117 (Translated substitution is always valid on elaborations)**.** If $\mathit{hsmod} = \mathsf{mkmod}(\mathcal{L}; \nu_0; \mathit{eenv}; \mathit{impdecls}; \mathit{defs}; \mathit{espcs})$ is defined, then $\mathsf{validsubst}(\theta^\star; \mathit{hsmod})$. ($\theta^\star$ maps file names in the image of $(-)^\star$ to other file names in the image of $(-)^\star$. In *hsmod* each alias is in *Paths*, rather than the image of $(-)^\star$, so $\theta^\star$ is the identity on these and never maps anything into them.)

15.3 Export resolution

Lemma 118 (Elaboration of exports produces translations of exports)**.**

(1) If $\mathit{eenv}(\mathit{eref}) = \mathit{phnm} : \mathit{espc}'$ and $\mathit{espc}' \leq \mathit{espc} \sqsubseteq \mathit{phnm}$, then $\nu_0^\star; \mathsf{refs}^\star_{\nu_0}(\mathit{eenv}) \vdash^{\mathrm{LL}}_{\mathrm{c}} \mathsf{mkexp}(\mathit{eenv}; \mathit{espc}) \rightsquigarrow \mathit{espc}^\star$. (Proof uses Property 121 to show that the translated export reference is well-formed.)

(2) If $\nu_0; \mathit{eenv} \Vdash_{\mathrm{c}} \mathit{expdecl} \rightsquigarrow \mathit{espcs}$ then $\nu_0^\star; \mathsf{refs}^\star_{\nu_0}(\mathit{eenv}) \vdash^{\mathrm{LL}}_{\mathrm{c}} \mathsf{mkexpdecl}(\mathit{espcs}; \mathit{eenv}) \rightsquigarrow \mathit{espcs}^\star$. (Proof uses Lemma 120 to prove existence of a syntactic reference that identifies each export (for $\mathsf{mkexpdecl}(-;-)$); and previous part.)

Lemma 119 (Identifiability of local entities in EL)**.** If $\mathsf{Local}.\chi \in \mathsf{dom}(\mathit{eenv})$ and $\mathsf{haslocaleenv}(\mathit{eenv}; \nu_0; \mathit{defs})$ then $\mathit{eenv}(\mathsf{Local}.\chi) = [\nu_0]\chi$. (Proof from definition of $\mathsf{mklocaleenv}(\nu_0; \mathit{defs})$.)

Lemma 120 (Identifiability of exported entities)**.**

(1) If $\tilde{\Phi}; \nu_0; \mathit{defs}; \mathit{eenv} \Vdash_{\mathrm{c}} \mathit{export} \rightsquigarrow \mathit{espcs}$ then $\forall \mathit{espc} \in \mathit{espcs} : \exists \mathit{eref} \in \mathsf{dom}(\mathit{eenv}), \mathit{espc}' \in \mathit{eenv} : (\mathit{eenv}(\mathit{eref}) = p : \mathit{espc}') \wedge \mathit{espc}' \leq \mathit{espc} \sqsubseteq p$. (Proof clear in all cases but ExpModAll; in that case, follows by defn of $\mathsf{filterespcs}(-;-)$ and $\mathsf{filterespc}(-;-)$.)

(2) If $\nu_0; \mathit{eenv} \Vdash_{\mathrm{c}} \mathit{expdecl} \rightsquigarrow \mathit{espcs}$ then $\forall \mathit{espc} \in \mathit{espcs} : \exists \mathit{eref} \in \mathsf{dom}(\mathit{eenv}), \mathit{espc}' \in \mathit{eenv} : (\mathit{eenv}(\mathit{eref}) = p : \mathit{espc}') \wedge \mathit{espc}' \leq \mathit{espc} \sqsubseteq p$. (Proof uses Lemma 119 in ExpLocal case; in ExpList, need to show that all individual lookup results combine to a single one; use previous part and Property 67 to get some initial *eref* to use; then Property 122 and Property 68 to combine those results and relate them to the initial *eref*.)

Property 121 (Preservation of entity environment lookup by translation)**.**

(1) If $\mathit{eenv}(\mathit{eref}) = \mathit{phnm}$ then $\mathsf{refs}^\star_{\nu_0}(\mathit{eenv})(\mathsf{refs}^\star_{\nu_0}(\mathit{eref})) = p^\star$. (Proof uses injectivity of $\mathsf{refs}^\star_{\nu_0}(-)$ (Property 123).)

(2) If $\mathit{eenv}(\mathit{eref}) = \mathit{phnm} : \mathit{espc}$ then $\mathsf{refs}^\star_{\nu_0}(\mathit{eenv})(\mathsf{refs}^\star_{\nu_0}(\mathit{eref})) = p^\star : \mathit{espc}^\star$. (Proof uses Property 127 and Property 116.)

Property 122 (Uniqueness of entities looked up in environment)**.** If $\mathit{eenv}(\mathit{eref}) = p : \mathit{espc}$ and $\mathit{eenv}(\mathit{eref}') = p : \mathit{espc}'$ then $\mathit{espc} = \mathit{espc}'$. (Proof by uniqueness in an *espc*-set, $\mathsf{locals}(\mathit{eenv})$.)

16 Algebraic properties of translation

Property 123 (Injectivity of entity translation)**.**

- If $\mathsf{refs}^\star_{\nu_0}(eref_1) = \mathsf{refs}^\star_{\nu_0}(eref_2)$ then $eref_1 = eref_2$.
- If $\mathsf{refs}^\star_{\nu_0}(eenv_1) = \mathsf{refs}^\star_{\nu_0}(eenv_2)$ then $eenv_1 = eenv_2$.

Property 124 (Translation distributes over (total) substitutions)**.** $(\theta\nu)^\star = \theta^\star\nu^\star$. Likewise for *typ*, *dspc*, *dspcs*, *phnm*, *espc*, *espcs*, τ.

Conjecture 125 (Physical context well-formed implies translation well-formed)**.** If $\Phi_1 \vdash \Phi_2\ \mathsf{wf}$ then $\Phi_1^\star \vdash \Phi_2^\star\ \mathsf{wf}$. (Obvious from structure of judgments and objects.)

Property 126 (Translation distributes over merging)**.**

- If $\Phi_1 \oplus \Phi_2$ defined, then $(\Phi_1 \oplus \Phi_2)^\star = \Phi_1^\star \oplus \Phi_2^\star$, which is itself defined.
- If $\tau_1^{m_1} \oplus \tau_2^{m_2}$ defined, then $(\tau_1^{m_1} \oplus \tau_2^{m_2})^\star = ({\tau_1}^\star)^{m_1} \oplus ({\tau_2}^\star)^{m_2}$, which is itself defined.
- If $\tau_1 \oplus \tau_2$ defined, then $(\tau_1 \oplus \tau_2)^\star = \tau_1^\star \oplus \tau_2^\star$, which is itself defined.
- If $dspcs_1 \oplus dspcs_2$ defined, then $(dspcs_1 \oplus dspcs_2)^\star = dspcs_1^\star \oplus dspcs_2^\star$, which is itself defined.
- If $espcs_1 \oplus espcs_2$ defined, then $(espcs_1 \oplus espcs_2)^\star = espcs_1^\star \oplus espcs_2^\star$, which is itself defined.
- If $dspc_1 \oplus dspc_2$ defined, then $(dspc_1 \oplus dspc_2)^\star = dspc_1^\star \oplus dspc_2^\star$, which is itself defined.
- If $espc_1 \oplus espc_2$ defined, then $(espc_1 \oplus espc_2)^\star = espc_1^\star \oplus espc_2^\star$, which is itself defined.
- Likewise for all n-ary merges over a non-empty index set I.

Property 127 (Distributivity of translation over semantic object operations)**.**

- $(\nu_1{:}{\tau_1}^{m_1}, \ldots, \nu_n{:}{\tau_n}^{m_n})^\star = (\nu_1^\star{:}{\tau_1^\star}^{m_1}, \ldots, \nu_n^\star{:}{\tau_n^\star}^{m_n})$.
- If $\tau = \Phi(\nu)$ then $\tau^\star = \Phi^\star(\nu^\star)$.
- If $espc \in \Phi(\nu)$ then $espc^\star \in \Phi^\star(\nu^\star)$.
- If $dspc \in \Phi(\nu)$ then $dspc^\star \in \Phi^\star(\nu^\star)$.
- $\mathsf{mkphnms}(espc)^\star = \mathsf{mkphnms}(espc^\star)$.
- $\mathsf{mkphnms}(espcs)^\star = \mathsf{mkphnms}(espcs^\star)$.

Lemma 128 (Kind extraction ignores translation)**.** $\mathsf{mkknd}(dspc^\star)$ is defined if and only if $\mathsf{mkknd}(dspc)$ is defined, and if so, $\mathsf{mkknd}(dspc^\star) = \mathsf{mkknd}(dspc)$.

Property 129 (Substitution commutes with translation on contexts)**.** If $\mathsf{apply}(\theta; \Phi)$ is defined then $\mathsf{apply}(\theta^\star; \Phi^\star)$ is defined and equals $\mathsf{apply}(\theta; \Phi)^\star$. (Proof relies on injectivity of $(-)^\star$ and Property 126.)

17 Properties of the soundness relation

Lemma 130 (Definition of the members of substitutions of corresponding contexts)**.** Suppose $\Phi = (\nu_i{:}{\tau_i}^{m_i} \mid i \in [1..n])$ and $dexp = (f_i \mapsto tfexp_i \mid i \in [1..n])$ and $\forall i \in [1..n] : {\nu_i}^\star = f_i$ and $\mathsf{apply}(\theta; \Phi)$ and $\mathsf{apply}(\theta; dexp)$. Then

(1) $\mathsf{dom}(\mathsf{apply}(\theta; \Phi))^\star = \mathsf{dom}(\mathsf{apply}(\theta^\star; dexp))$, and

(2) $\forall \nu{:}\tau^m \in \mathsf{apply}(\theta;\Phi)$:

(a) $I = \{i \in [1..n] \mid \nu = \theta\nu_i\}$ and $\tau^m = \bigoplus_{i\in I}(\theta\tau_i)^{m_i}$, and

(b) $(f \mapsto \mathit{tfexp}) \in \mathsf{apply}(\theta^\star; \mathit{dexp})$, where $f = \nu^\star$ and $\mathit{tfexp} = \bigoplus_{i\in I} \theta^\star \mathit{tfexp}_i$.

Proof of (1) uses Property 67 and injectivity of $(-)^\star$ and Property 124; proof of (2) follows from definitions of substitution and the same properties.)

Lemma 131 (Related packages form a bijection)**.** If $\Phi_{\mathsf{pkg}} \vdash \Phi \sim \mathit{dexp}$ then

- $\Phi = (\nu_i{:}{\tau_i}^{m_i} \mid i \in [1..n])$,
- $\mathit{dexp} = \{f_i \mapsto \mathit{tfexp}_i \mid i \in [1..n]\}$, and
- $\forall i \in [1..n] :\ \nu_i^\star = f_i \ \wedge\ \tau_i^\star = \mathsf{typ}(\mathit{tfexp}_i) \ \wedge\ m_i = \mathsf{pol}(\mathit{tfexp}_i)$.

(Clear by definition of soundness relation since $(-)^\star$ is injective and the two mappings have the same size.)

Lemma 132 (A negative EL context is related to its IL stub file translation)**.** If $\Phi_{\mathsf{ctx}} \vdash \Phi\ \mathsf{wf}$ and $\Phi \preceq -$, then $\Phi_{\mathsf{ctx}} \vdash \Phi \sim \mathsf{mkstubs}(\Phi)$. (Proof uses Lemma 125 and Lemma 93.)

Lemma 133 (EL contexts translate to the environment of related terms)**.** If $\Phi_{\mathsf{pkg}} \vdash \Phi \sim \mathit{dexp}$ then $\mathsf{mkfenv}(\mathit{dexp}) = \Phi^\star$. (Proof straightforward by Lemma 131 and Property 127.)

17.1 Strengthening

Lemma 134 (Relatedness of packages is preserved under filtering)**.** If $\emptyset\vdash \Phi \sim \mathit{dexp}$ and $\mathsf{depends}_\Phi(N) \subseteq N$, then $\emptyset\vdash (\Phi|_N) \sim (\mathit{dexp}|_{N^\star})$. (Proof uses Property 135; Lemma 6 and Lemma 83 for well-formedness of filtered context and directory.)deps-stof

Property 135 (Dependencies are preserved by translation)**.** If $\nu \in \mathsf{dom}(\Phi)$ then $\mathsf{depends}_\Phi(\nu)^\star = \mathsf{depends}_{\Phi^*}(\nu^\star)$.

17.2 Substitutability

Lemma 136 (Relatedness of packages is preserved under substitution)**.** If $\emptyset\vdash \Phi \sim \mathit{dexp}$ and $\mathsf{apply}(\theta;\Phi)$ defined, then $\mathsf{apply}(\theta^\star;\mathit{dexp})$ is defined and $\emptyset\vdash \mathsf{apply}(\theta;\Phi) \sim \mathsf{apply}(\theta;\mathit{dexp})$. (Proof uses Lemma 133 and Lemma 112 for definability of substitution on *dexp*; Lemma 137 and Lemma 86 for well-typedness of $\theta^\star\mathit{dexp}$; Lemma 5 for well-formedness of $\theta\Phi$; Lemma 130 for gathering substituted pieces; and Lemma 138 and Lemma 139 for preserving the relations.)

Lemma 137 (Translated substitution is always valid on related *dexp*s)**.** If $\Phi_{\mathsf{pkg}} \vdash \Phi \sim \mathit{dexp}$ then $\mathsf{validsubst}(\theta^\star;\mathit{dexp})$. (Every $\mathit{hsmod} \in \mathit{dexp}$ is the result of elaboration, by definition of the soundness relation.)

Lemma 138 (Relatedness of single module types is preserved under substitution)**.** If $\vdash \nu{:}\tau^m \sim f \mapsto \mathit{tfexp}$ then $\vdash (\theta\nu){:}(\theta\tau)^m \sim (\theta^\star f) \mapsto (\theta^\star \mathit{tfexp})$. (Proof straightforward by Property 113.)

17.3 Merging

Lemma 139 (Relatedness of single module types is preserved under merging)**.**

(1) If $\vdash \nu{:}{\tau_1}^{m_1} \sim f \mapsto \mathit{tfexp}_1$ and $\vdash \nu{:}{\tau_2}^{m_2} \sim f \mapsto \mathit{tfexp}_2$ and $\tau_1^{m_1} \perp \tau_2^{m_2}$ and $\mathit{tfexp}_1 \perp \mathit{tfexp}_2$, then $\vdash \nu{:}(\tau_1 \oplus \tau_2)^{m_1 \oplus m_2} \sim f \mapsto (\mathit{tfexp}_1 \oplus \mathit{tfexp}_2)$. (Proof straightforward by Property 113.)
(2) Likewise for n-ary merges over a non-empty index set I.