Modularly Composing Typed Language Fragments (Supplemental Material)

Abstract

This document provides the full technical development described in the paper "Modularly Composing Typed Language Fragments" submitted to PLDI 2015.

Contents

List of Figures

1 Internal Language

1.1 Semantics

We assume an internal language where the statics are specified in the standard way by judgements for type formation $\Delta \vdash \tau$, typing context formation $\Delta \vdash \Gamma$ and type assignment $\Delta\ \Gamma \vdash \iota : \tau^{+}$. The internal dynamics are specified as a structural operational semantics with a stepping judgement $\iota \mapsto \iota^{+}$ and a value judgement $\iota\ \mathtt{val}$. The multi-step judgement $\iota \mapsto^{*} \iota^{+}$ is the reflexive, transitive closure of the stepping judgement and the evaluation judgement $\iota \Downarrow \iota'$ is defined iff $\iota \mapsto^{*} \iota'$ and $\iota'\ \mathtt{val}$. Both the static and dynamic semantics of the IL can be found in any standard textbook covering typed lambda calculi (we directly follow [1]), so we assume familiarity and give the lemmas in this section without proof.

Our particular choice in the paper is $\mathcal{L}\{\rightharpoonup \forall\, \mu\, 1 \times +\}$, the syntax for which is shown in Figure 1, as representative of any intermediate language for a typed functional language. In fact, our intention is not to prescribe a particular choice of IL, so we will here only review the key metatheoretic properties that the IL must possess. Each choice of IL is technically a distinct dialect of $@\lambda$, but for the broad class of ILs that enjoy these properties, the metatheory in the remainder of the supplement should follow without trouble.

Assumption 1 (Internal Type Assignment)**.** *If* $\Delta \vdash \Gamma$ *and* $\Delta\ \Gamma \vdash \iota : \tau$ *then* $\Delta \vdash \tau$*.*

Internal type safety follows the standard methodology: preservation and progress lemmas.

Assumption 2 (Internal Progress)**.** *If* $\emptyset\ \emptyset \vdash \iota : \tau$ *then either* $\iota\ \mathtt{val}$ *or* $\iota \mapsto \iota'$*.*

Assumption 3 (Internal Preservation)**.** *If* $\emptyset\ \emptyset \vdash \iota : \tau$ *and* $\iota \mapsto \iota'$ *then* $\emptyset\ \emptyset \vdash \iota' : \tau$*.*

We assume that typing and type formation contexts, Δ and Γ, are identified up to exchange and contraction and the internal semantics obeys weakening. We assume that the names of variables and type variables are unimportant, so that α-equivalent terms, types and contexts are identified implicitly throughout this work.

1.2 Explicit Substitutions

We also assume substitution is defined in the usual way. We define explicit n-ary substitutions for type variables, δ, and term variables, γ, also in the standard way, writing $[\delta]\Gamma$, $[\delta]\tau$ and $[\delta]\iota$ for application of type substitutions and $[\gamma]\iota$ for application of term substitutions. We need the definitions of substitution validity shown in Figures 7 and 8 and assume the following lemmas.

Assumption 4 (Internal Term Substitutions)**.** *If* $\Delta \vdash \Gamma$ *and* $\Delta \vdash \Gamma'$ *and* $\Delta\ \Gamma \vdash \gamma : \Gamma'$ *and* $\Delta\ \Gamma\Gamma' \vdash \iota : \tau$ *then* $\Delta\ \Gamma \vdash [\gamma]\iota : \tau$*.*

Assumption 5 (Internal Type Substitution on Types)**.** *If* $\Delta \vdash \delta : \Delta'$ *and* $\Delta\Delta' \vdash \tau$ *then* $\Delta \vdash [\delta]\tau$*.*

Assumption 6 (Internal Type Substitutions on Typing Contexts)**.** *If* $\Delta \vdash \delta : \Delta'$ *and* $\Delta\Delta' \vdash \Gamma$ *then* $\Delta \vdash [\delta]\Gamma$*.*

Assumption 7 (Internal Type Substitutions on Terms)**.** *If* $\Delta \vdash \delta : \Delta'$ *and* $\Delta\Delta' \vdash \tau$ *and* $\Delta\Delta' \vdash \Gamma$ *and* $\Delta\Delta'\ \Gamma \vdash \iota : \tau$ *then* $\Delta\ [\delta]\Gamma \vdash [\delta]\iota : [\delta]\tau$*.*

2 Tycon Contexts

Tycon contexts, Φ, are semantically defined as ordered mappings from tycon names $\textsc{tc}$ to tycon definitions. As with all ordered mappings, we implicitly assume that all names are unique and that certain operations, e.g. dom(Φ), which extracts a set of tycon names, and $\mathsf{tycon}\ \textsc{tc}\ \{\theta\} \sim \psi \in \Phi$, which checks whether a tycon definition is in Φ, and $\Phi\Phi'$, which concatenates two tycon contexts with disjoint domains, are available without defining them explicitly.

2.1 Validity

The tycon context validity judgement $\vdash \Phi$ is specified in Figure 10. The kinding and kind formation rules will be given in the next section.

Lemma 1 (Tycon Context Validity)**.** *If (a)* $\vdash \Phi$ *and (b)* $\mathsf{tycon}\ \textsc{tc}\ \{\mathsf{trans} = \sigma_{schema}\ \mathsf{in}\ \omega\} \sim \mathsf{tcsig}[\kappa_{tyidx}]\ \{\chi\} \in \Phi$ *then there exists a prefix* Φ' *of* Φ *such that (i)* $\vdash \Phi'$ *and (ii)* $\vdash \Phi', \mathsf{tycon}\ \textsc{tc}\ \{\mathsf{trans} = \sigma_{schema}\ \mathsf{in}\ \omega\} \sim \mathsf{tcsig}[\kappa_{tyidx}]\ \{\chi\}$ *and (iii)* $\vdash \kappa_{tyidx}\ \mathsf{eq}$ *and (iv)* $\emptyset\ \emptyset \vdash^0_{\Phi'} \sigma_{schema} :: \kappa_{tyidx} \to \mathsf{ITy}$ *and (v)* $\vdash_{\Phi', \mathsf{tycon}\ \textsc{tc}\ \{\mathsf{trans}=\sigma_{schema}\ \mathsf{in}\ \omega\} \sim \mathsf{tcsig}[\kappa_{tyidx}]\ \{\chi\}} \omega \sim \mathsf{tcsig}[\kappa_{tyidx}]\ \{\chi\}$.

Proof. Rule induction on (a) and (b).

Case (tcc-emp). Does not apply by (b).

Case (tcc-ext). We have two cases by (b):

Case $\Phi = \Phi', \mathsf{tycon}\ \textsc{tc}\ \{\mathsf{trans} = \sigma_{\text{schema}}\ \mathsf{in}\ \omega\} \sim \mathsf{tcsig}[\kappa_{\text{tyidx}}]\ \{\chi\}$. In this case, we pick Φ' and have our conclusions:

(i) $\vdash \Phi'$ (premise 1 of (tcc-ext))

(ii) $\vdash \Phi', \mathsf{tycon}\ \textsc{tc}\ \{\mathsf{trans} = \sigma_{\text{schema}}\ \mathsf{in}\ \omega\} \sim \mathsf{tcsig}[\kappa_{\text{tyidx}}]\ \{\chi\}$ (by assumption (a))

(iii) $\vdash \kappa_{\text{tyidx}}\ \mathsf{eq}$ (premise 2 of (tcc-ext))

(iv) $\emptyset\ \emptyset \vdash^0_{\Phi'} \sigma_{\text{schema}} :: \kappa_{\text{tyidx}} \to \mathsf{ITy}$ (premise 3 of (tcc-ext))

(v) $\vdash_{\Phi', \mathsf{tycon}\ \textsc{tc}\ \{\mathsf{trans}=\sigma_{\text{schema}}\ \mathsf{in}\ \omega\} \sim \mathsf{tcsig}[\kappa_{\text{tyidx}}]\ \{\chi\}} \omega \sim \mathsf{tcsig}[\kappa_{\text{tyidx}}]\ \{\chi\}$ (premise 4 of (tcc-ext))

Case $\Phi = \Phi', \mathsf{tycon}\ \textsc{tc}'\ \{\theta'\} \sim \psi'$ for $\textsc{tc} \neq \textsc{tc}'$ and (1) $\mathsf{tycon}\ \textsc{tc}\ \{\mathsf{trans} = \sigma_{\text{schema}}\ \mathsf{in}\ \omega\} \sim \mathsf{tcsig}[\kappa_{\text{tyidx}}]\ \{\chi\} \in \Phi'$. We have the following:

(2) $\vdash \Phi'$ (premise 1 of (tcc-ext))

Apply IH to (2) and (1). There exists a prefix Φ'' of Φ' such that

(3) $\vdash \Phi''$

(4) $\vdash \Phi'', \mathsf{tycon}\ \textsc{tc}\ \{\mathsf{trans} = \sigma_{\text{schema}}\ \mathsf{in}\ \omega\} \sim \mathsf{tcsig}[\kappa_{\text{tyidx}}]\ \{\chi\}$

(5) $\vdash \kappa_{\text{tyidx}}\ \mathsf{eq}$

(6) $\emptyset\ \emptyset \vdash^0_{\Phi''} \sigma_{\text{schema}} :: \kappa_{\text{tyidx}} \to \mathsf{ITy}$

(7) $\vdash_{\Phi'',\mathsf{tycon}\ \text{TC}\ \{\mathsf{trans}=\sigma_{\text{schema}}\ \mathsf{in}\ \omega\}\sim\mathsf{tcsig}[\kappa_{\text{tyidx}}]\,\{\chi\}}\ \omega \sim \mathsf{tcsig}[\kappa_{\text{tyidx}}]\,\{\chi\}$

By transitivity of prefixing, we have that Φ'' is a prefix of Φ. Picking Φ'', conclusions (i-v) are (3-7), respectively.

□

In the final premise of (tcc-ext), the opcon structure is checked against the tycon's signature by the rules in Figure 11. Opcon structures and opcon signatures are finite maps, so we implicitly assume that all operator names are unique and standard operations are defined as above.

Lemma 2 (Intro Opcon Existence and Validity)**.** *If* $\vdash_\Phi \omega \sim \mathsf{tcsig}[\kappa_{tyidx}]\,\{\chi\}$ *then (i)* $\mathsf{intro}[\kappa_{tmidx}] \in \chi$ *and (ii)* $\emptyset \vdash \kappa_{tmidx}$ *and (iii)* $\mathsf{ana\ intro} = \sigma_{def} \in \omega$ *and (iv)* $\emptyset\ \emptyset \vdash^0_\Phi \sigma_{def} :: \kappa_{tyidx} \to \kappa_{tmidx} \to \mathsf{List}[\mathsf{Arg}] \to \mathsf{ITm}$.

Proof. Rule induction on assumption.

Case (ocstruct-intro). We have that $\omega = \mathsf{ana\ intro} = \sigma_{\text{def}}$ and $\chi = \mathsf{intro}[\kappa_{\text{tmidx}}]$.

(i) $\mathsf{intro}[\kappa_{\text{tmidx}}] \in \mathsf{intro}[\kappa_{\text{tmidx}}]$ (by definition)

(ii) $\emptyset \vdash \kappa_{\text{tmidx}}$ (premise 1)

(iii) $\mathsf{ana\ intro} = \sigma_{\text{def}} \in \mathsf{ana\ intro} = \sigma_{\text{def}}$ (by definition)

(iv) $\emptyset\ \emptyset \vdash^0_\Phi \sigma_{\text{def}} :: \kappa_{\text{tyidx}} \to \kappa_{\text{tmidx}} \to \mathsf{List}[\mathsf{Arg}] \to \mathsf{ITm}$ (premise 2)

Case (ocstruct-targ). We have that $\omega = \omega'; \mathsf{syn}\ \mathbf{op} = \sigma_{\text{def}}$ and $\chi = \chi'; \mathbf{op}[\kappa'_{\text{tmidx}}]$.

(1) $\vdash_\Phi \omega' \sim \mathsf{tcsig}[\kappa_{\text{tyidx}}]\,\{\chi\}$ (premise 1)

Apply IH to (1):

(2) $\mathsf{intro}[\kappa_{\text{tmidx}}] \in \chi'$

(3) $\emptyset \vdash \kappa_{\text{tmidx}}$

(4) $\mathsf{ana\ intro} = \sigma_{\text{def}} \in \omega'$

(5) $\emptyset\ \emptyset \vdash^0_\Phi \sigma_{\text{def}} :: \kappa_{\text{tyidx}} \to \kappa_{\text{tmidx}} \to \mathsf{List}[\mathsf{Arg}] \to \mathsf{ITm}$

We can now conclude:

(i) $\mathsf{intro}[\kappa_{\text{tmidx}}] \in \chi$ by extension of finite mappings on (2)

(ii) $\emptyset \vdash \kappa_{\text{tmidx}}$ by (3)

(iii) $\mathsf{ana\ intro} = \sigma_{\text{def}} \in \omega$ by extension of finite mappings on (4)

(iv) $\emptyset\ \emptyset \vdash^0_\Phi \sigma_{\text{def}} :: \kappa_{\text{tyidx}} \to \kappa_{\text{tmidx}} \to \mathsf{List}[\mathsf{Arg}] \to \mathsf{ITm}$ by (5).

□

Lemma 3 (Targeted Opcon Validity)**.** *If (a)* $\vdash_\Phi \omega \sim \mathsf{tcsig}[\kappa_{tyidx}]\ \{\chi\}$ *and (b)* $\mathsf{syn}\ \boldsymbol{op} = \sigma_{def} \in \omega$ *then (i)* $\boldsymbol{op}[\kappa_{tmidx}] \in \chi$ *and (ii)* $\emptyset \vdash \kappa_{tmidx}$ *and (iii)* $\emptyset\ \emptyset \vdash^0_\Phi \sigma_{def} :: \kappa_{tyidx} \rightarrow \kappa_{tmidx} \rightarrow \mathsf{List}[\mathsf{Arg}] \rightarrow (\mathsf{Ty} \times \mathsf{ITm})$.

Proof. Rule induction on assumption (a).

Case (ocstruct-intro). Does not apply by (b).

Case (ocstruct-targ). We have two cases by (b).

Case $\omega = \omega'; \mathsf{syn}\ \textbf{op} = \sigma_{\text{def}}$ and $\chi = \chi', \textbf{op}[\kappa_{\text{tmidx}}]$. Conclusion (i) follows by definition and conclusions (ii) and (iii) are premises 2 and 3 of (ocstruct-targ).

Case $\omega = \omega'; \mathsf{syn}\ \textbf{op}' = \sigma'_{\text{def}}$ such that **op** $\neq$ **op'** and (1) $\mathsf{syn}\ \textbf{op} = \sigma_{\text{def}} \in \omega'$. In this case, we have

(2) $\vdash_\Phi \omega' \sim \mathsf{tcsig}[\kappa_{\text{tyidx}}]\ \{\chi'\}$ (premise 1 of (ocstruct-targ))

Applying the IH to (2) and (1), we have:

(3) $\textbf{op}[\kappa_{\text{tmidx}}] \in \chi'$

(4) $\emptyset \vdash \kappa_{\text{tmidx}}$

(5) $\emptyset\ \emptyset \vdash^0_\Phi \sigma_{\text{def}} :: \kappa_{\text{tyidx}} \rightarrow \kappa_{\text{tmidx}} \rightarrow \mathsf{List}[\mathsf{Arg}] \rightarrow (\mathsf{Ty} \times \mathsf{ITm})$

We thus have conclusion (i) by extension of finite mappings on (3), and conclusions (ii) and (iii) are (4) and (5).

□

2.2 Signature Validity

For the purposes of the SL, only the type signatures are relevant.[1] It is thus useful to define judgements for checking tycon signature validity separately. The judgements $\vdash \psi$ for tycon signatures, $\vdash \chi$ for opcon signatures, and $\vdash \Phi$ `sigsok` for all signatures in a tycon context are in Figure 12. Kind and equality kind formation are covered in the next section.

Lemma 4 (Signatures Define Equality Type Index Kinds)**.** *If* $\vdash \mathsf{tcsig}[\kappa_{tyidx}]\ \{\chi\}$ *then* $\vdash \kappa_{tyidx}\ \mathsf{eq}$.

Proof. By rule induction on the assumption. The conclusion is the first premise. □

Lemma 5 (Valid Opcon Structures Define Valid Opcon Signatures)**.** *If* $\vdash_\Phi \omega \sim \mathsf{tcsig}[\kappa_{tyidx}]\ \{\chi\}$ *then* $\vdash \chi$.

[1] In fact, only the type index kind is relevant, though in future work the remainder will likely be and it is cleaner to define complete signature checking.

Proof. By rule induction on the assumption.

Case (ocstruct-intro). We have that $\chi = \mathsf{intro}[\kappa_{\text{tmidx}}]$ and (1) $\emptyset \vdash \kappa_{\text{tyidx}}$. Apply (ocsig-intro-ok) to (1).

Case (ocstruct-targ-ok). We have that $\chi = \chi', \mathbf{op}[\kappa_{\text{tmidx}}]$ and (1) $\vdash_\Phi \omega \sim \mathsf{tcsig}[\kappa_{\text{tmidx}}]\,\{\chi'\}$ and (2) $\emptyset \vdash \kappa_{\text{tmidx}}$. Applying the IH to (1), we have (3) $\vdash \chi'$. Apply (ocsig-targ-ok) to (3) and (2). □

Lemma 6 (Well-Defined Contexts Have Well-Formed Signatures)**.** *If* $\vdash \Phi$ *then* $\vdash \Phi$ $\texttt{sigsok}$.

Proof. By rule induction on the assumption.

Case (tcc-emp). Apply (tcc-sigsok-emp).

Case (tcc-ext). We have that $\Phi = \Phi', \mathsf{tycon}\ \textsc{tc}\ \{\theta\} \sim \mathsf{tcsig}[\kappa_{\text{tyidx}}]\,\{\chi\}$ and (1) $\vdash \Phi'$ and (2) $\vdash \kappa_{\text{tyidx}}\ \mathsf{eq}$ and (3) $\vdash_\Phi \omega \sim \mathsf{tcsig}[\kappa_{\text{tyidx}}]\,\{\chi\}$.

By the IH on (1), we have (4) $\vdash \Phi'$ $\texttt{sigsok}$.

By Lemma 5 on (3) we have (5) $\vdash \chi$.

By (tcsig-ok) on (2) and (5) we have (6) $\vdash \mathsf{tcsig}[\kappa_{\text{tyidx}}]\,\{\chi\}$.

By (tcc-sigsok-ext) on (4) and (6) we have our conclusion. □

Lemma 7 (Tycons Have Valid Signatures)**.** *If (a)* $\mathsf{tycon}\ \textsc{tc}\ \{\theta\} \sim \psi \in \Phi$ *and (b)* $\vdash \Phi$ $\texttt{sigsok}$ *then* $\vdash \psi$.

Proof. By rule induction on (b) and (a).

Case (tcc-sigsok-emp). Does not apply by (a).

Case (tcc-sigsok-ext). We have two possible cases by (a):

> **Case** $\Phi = \Phi', \mathsf{tycon}\ \textsc{tc}\ \{\theta\} \sim \psi$. Then the conclusion is premise 2 of (b).
>
> **Case** $\Phi = \Phi', \mathsf{tycon}\ \textsc{tc}'\ \{\theta'\} \sim \psi'$ for $\textsc{tc} \neq \textsc{tc}'$ and (1) $\mathsf{tycon}\ \textsc{tc}\ \{\theta\} \sim \psi \in \Phi'$. We have (2) $\vdash \Phi'$ $\texttt{sigsok}$ by premise 1 of (b). Apply the IH to (1) and (2).

□

Lemma 8 (Tycon Context Signature Validity Respects Extension)**.** *If (a)* $\vdash \Phi$ $\texttt{sigsok}$ *and (b)* $\vdash \Phi'$ $\texttt{sigsok}$ *and (c)* $\mathit{dom}(\Phi) \cap \mathit{dom}(\Phi') = \emptyset$ *then* $\vdash \Phi\Phi'$ $\texttt{sigsok}$.

Proof. Rule induction on (b).

Case (tcc-sigsok-emp). Here, $\Phi' = \emptyset$ so $\Phi\Phi' = \Phi$ and the conclusion follows by (a).

Case (tcc-sigsok-ext). Here, $\Phi' = \Phi'', \mathsf{tycon}\ \textsc{tc}\ \{\mathsf{trans} = \sigma_{\text{schema}}\ \mathsf{in}\ \omega\} \sim \mathsf{tcsig}[\kappa_{\text{tyidx}}]\,\{\chi\}$ and (1) $\vdash \Phi''$ $\texttt{sigsok}$ and (2) $\vdash \psi$.

$\Phi\Phi''$ is well-formed because prefixing respects disjointness and (c).

Applying the IH on (a) and (1), we have that (3) $\vdash \Phi\Phi''$ $\texttt{sigsok}$.

By (tcc-sigsok-ext) on (3) and (2) we have our conclusion. □

3 Static Language

3.1 Syntax

In the paper, we described the static language (SL) as consisting of a total functional core plus three additional kinds. That core is derived directly from [1]. Since the form of the judgements are not technically identical, we will re-state the rules below, but we will only sketch the corresponding cases in the proofs for concision (the core ignores the tycon context, argument bound n, etc.).

The only change from the paper here is that we define the SL as having a built in list kind, rather than the more general mechanism for inductive kinds hinted at in the paper (lists are the only inductive kind we strictly need). The details of inductive kinds are in [1], but again would distract from our main point and aren't needed to prove our theorems. Another approach is to simply allow recursive kinds, which would make it possible to define static terms that diverge. This does not break our fundamental guarantees, only decidability of typechecking (which we do not go into in detail here, as it is a more practical concern and requires non-trivial proofs of weak normalization that would also distract from our main point).

As with the IL, each choice of SL forms a dialect of @λ. We anticipate that the SL (essentially, a domain-specific language for implementing typed translation semantics), and the IL would need to evolve much more slowly than the EL (e.g. the GHC compiler's IL has remained stable for many years), so dialect formation should be needed far more rarely

3.2 Kinds

3.2.1 Kind Formation

The kind formation rules are straightforward, shown in Figure 13. Kind formation contexts $\boldsymbol{\Delta}$ are defined as finite mappings and thus are identified up to contraction and exchange, and we can assume the domains are disjoint. We assume substitution is defined in the usual way, so that standard substitution lemmas hold.

Assumption 8 (Kind Variable Substitution - Kinds)**.** *If $\boldsymbol{\Delta}, \boldsymbol{\alpha} \vdash \kappa'$ and $\boldsymbol{\Delta} \vdash \kappa$ then $\boldsymbol{\Delta} \vdash [\kappa/\boldsymbol{\alpha}]\kappa'$.*

Lemma 9 (Weakening of Kind Formation)**.** *If $\boldsymbol{\Delta} \vdash \kappa$ then $\boldsymbol{\Delta}\boldsymbol{\Delta}' \vdash \kappa$.*

Proof. Rule induction on assumption.

Cases (kf-unit), (kf-ty), (kf-ity), (kf-itm). These rules are defined for all $\boldsymbol{\Delta}$, so we re-apply the same rule.

Cases (kf-arrow), (kf-prod), (kf-sum). Apply the IH to premise 1 to get (1) $\boldsymbol{\Delta}\boldsymbol{\Delta}' \vdash \kappa$. Apply the IH to premise 2 to get (2) $\boldsymbol{\Delta}\boldsymbol{\Delta}' \vdash \kappa_2$. Re-apply the same rule to (1) and (2).

Case (kf-list). Here, $\kappa = \mathsf{List}[\kappa']$. Apply the IH to premise 1 to get (1) $\boldsymbol{\Delta}\boldsymbol{\Delta}' \vdash \kappa'$. Apply (kf-list) to (1).

Case (kf-alpha). By extension of finite mappings, if $\boldsymbol{\alpha} \in \boldsymbol{\Delta}$ then (1) $\boldsymbol{\alpha} \in \boldsymbol{\Delta}\boldsymbol{\Delta}'$. Apply (kf-alpha) to (1).

Case (kf-forall). Here, $\kappa = \forall(\boldsymbol{\alpha}.\kappa')$. Apply the IH to premise 1 to get (1) $(\boldsymbol{\Delta}, \boldsymbol{\alpha})\boldsymbol{\Delta}' \vdash \kappa'$. By exchange, $(\boldsymbol{\Delta}, \boldsymbol{\alpha})\boldsymbol{\Delta}' = \boldsymbol{\Delta}\boldsymbol{\Delta}', \boldsymbol{\alpha}$ and thus we can apply (kf-forall) to (1).

□

3.2.2 Equality Kinds

Equality kinds are included to avoid complicating type equality (and guarantee *argument independence* of values, see below). Arrow kinds, polymorphic kinds and term quotation kinds (because they are argument-dependent, see below) are not equality kinds (though an approach for quantifying over only equality kinds could be introduced to allow polymorphic equality kinds, as Standard ML provides [2]). The rules are given in Figure 14.

Lemma 10 (Equality Kind Well-Formedness)**.** *If* $\vdash \kappa\ \texttt{eq}$ *then* $\emptyset \vdash \kappa$.

Proof. Rule induction on the assumption.

Case (keq-list). We have $\kappa = \mathsf{List}[\kappa']$ and (1) $\vdash \kappa'\ \texttt{eq}$. Apply IH to (1) to get that (2) $\emptyset \vdash \kappa'$. Apply (kf-list) to (2).

Case (keq-unit). We have $\kappa = 1$. Apply (kf-unit).

Case (keq-prod). We have $\kappa = \kappa_1 \times \kappa_2$ and (1) $\vdash \kappa_1\ \texttt{eq}$ and $\vdash \kappa_2\ \texttt{eq}$. Apply IH to (1) to get (3) $\emptyset \vdash \kappa_1$. Apply IH to (2) to get (4) $\emptyset \vdash \kappa_2$. Apply (kf-prod) to (3) and (4).

Case (keq-sum). We have $\kappa = \kappa_1 + \kappa_2$ and (1) $\vdash \kappa_1\ \texttt{eq}$ and $\vdash \kappa_2\ \texttt{eq}$. Apply IH to (1) to get (3) $\emptyset \vdash \kappa_1$. Apply IH to (2) to get (4) $\emptyset \vdash \kappa_2$. Apply (kf-sum) to (3) and (4).

Case (keq-ty). We have $\kappa = \mathsf{Ty}$. Apply (kf-ty).

Case (keq-ity). We have $\kappa = \mathsf{ITy}$. Apply (kf-ity). □

3.2.3 Kinding Contexts

Kinding contexts map static variables, written in bold, to kinds. The kinding context formation judgement checks these kinds against a kind formation context. Because both are finite mappings, we implicitly identify them up to exchange and contraction and the domain can be assumed disjoint implicitly. Like the IL, static terms and kinds are implicitly identified up to α-renaming of kind and static term variables, and we assume substitution is defined in the usual way.

Lemma 11 (Kinding Context Lookup)**.** *If (a)* $\mathbf{\Delta} \vdash \mathbf{\Gamma}$ *and (b)* $\boldsymbol{x} :: \kappa \in \mathbf{\Gamma}$ *then* $\mathbf{\Delta} \vdash \kappa$.

Proof. Rule induction on (a).

Case (kctx-emp). Does not apply by (b).

Case (kctx-ext). We have that $\mathbf{\Gamma} = \mathbf{\Gamma}', \boldsymbol{x}' :: \kappa'$ and (1) $\mathbf{\Delta} \vdash \mathbf{\Gamma}'$ and (2) $\mathbf{\Delta} \vdash \kappa$. There are two cases by (b):

Case $\boldsymbol{x} = \boldsymbol{x}'$ and $\kappa = \kappa'$. We have our conclusion by (2).

Case $\boldsymbol{x} \neq \boldsymbol{x}'$ and (3) $\boldsymbol{x} :: \kappa \in \mathbf{\Gamma}'$. Apply the IH to (1) and (3).

□

Lemma 12 (Weakening of Kinding Context Formation)**.** *If* $\mathbf{\Delta} \vdash \mathbf{\Gamma}$ *then* $\mathbf{\Delta}\mathbf{\Delta}' \vdash \mathbf{\Gamma}$.

Proof. Rule induction on assumption

Case (kctx-emp). Apply (kctx-emp).

Case (kctx-ext). We have $\mathbf{\Gamma} = \mathbf{\Gamma}', \boldsymbol{x} :: \kappa$. Applying the IH to premise 1, we have (1) $\mathbf{\Delta\Delta}' \vdash \mathbf{\Gamma}'$. By Lemma 9 on premise 2, we have (2) $\mathbf{\Delta\Delta}' \vdash \kappa$. Apply (kctx-ext) to (1) and (2). □

3.3 Kinding

The kinding rules are given in Figures 16-22. Figure 16 describes kinding for the core and the rules are directly based on those in [1], only tacking Φ and n onto each rule, but never inspecting either one. For cases of our proofs where an existing proof script would be satisfactory by simply inserting these additional subscripts onto the kinding judgement, we will simply note this. Figure 17 describes kinding for types. Figures 18 and 19 describes kinding for quoted translational internal types. Figures 20 and 21 describe kinding for quoted translational internal terms. Only the rules for forms specific to quoted terms in Figures 18 and 20 are interesting; the rules in Figures 19 and 21 simply recurse generically over all sub-terms of shared form with ι or τ. We will give only representative example cases for these shared forms in our proofs, rather than covering each form separately. Figure 22 describes kinding for the SL-EL interface.

Lemma 13 (Kind Assignment)**.** *If (a) $\mathbf{\Delta} \vdash \mathbf{\Gamma}$ and (b) $\mathbf{\Delta}\ \mathbf{\Gamma} \vdash^n_\Phi \sigma :: \kappa$ then $\mathbf{\Delta} \vdash \kappa$.*

Proof. By rule induction on (b). The cases for the rules in Figure 16 follow the usual script and are omitted.

Cases (k-ty-parr) and (k-ty-ext) and (k-ty-other). In each case, $\kappa = \mathsf{Ty}$. Apply (kf-ty).

Cases (k-tycase-parr) and (k-tycase-ext). In each case, $\sigma = \mathsf{tycase}[c](\sigma'; \boldsymbol{x}.\sigma_1; \sigma_2)$ and (1) $\mathbf{\Delta}\ \mathbf{\Gamma} \vdash^n_\Phi \sigma_2 :: \kappa$. Apply IH to (a) and (1).

Cases [all rules in Figures 18 and 19]. In each case, $\kappa = \mathsf{ITy}$. Apply (kf-ity).

Cases [all rules in Figure 20 and 21]. In each case, $\kappa = \mathsf{ITm}$. Apply (kf-itm).

Case (k-ana). We have $\kappa = \mathsf{ITm}$. Apply (kf-itm).

Case (k-syn). We have $\kappa = \mathsf{Ty} \times \mathsf{ITm}$. Apply (kf-prod). Apply (kf-ty). Apply (kf-itm). □

Lemma 14 (Weakening of Argument Bounds)**.** *If (a) $\mathbf{\Delta}\ \mathbf{\Gamma} \vdash^{n'}_\Phi \sigma :: \kappa$ and (b) $n' < n$ then $\mathbf{\Delta}\ \mathbf{\Gamma} \vdash^n_\Phi \sigma :: \kappa$.*

Proof. Rule induction on (a).

All cases except the following proceed generically by applying the IH to all kinding premises then re-applying the same rule, because they are defined for all n and all premises are checked under the same n.

Case (k-qitm-anatrans). In this case, $\sigma = \rhd(\mathsf{anatrans}[n''](\sigma'))$ and (1) $n'' < n'$ and (2) $\mathbf{\Delta}\ \mathbf{\Gamma} \vdash^{n'}_\Phi \sigma' :: \mathsf{Ty}$. By transitivity on (1) and (b), we have (3) $n'' < n$. By the IH on (2) and (b), we have (4) $\mathbf{\Delta}\ \mathbf{\Gamma} \vdash^{\sigma'}_n \mathsf{Ty} ::$. Apply (k-qitm-anatrans) to (3) and (4).

Case (k-qitm-syntrans). In this case, $\sigma = \rhd(\mathsf{syntrans}[n''])$ and (1) $n'' < n'$. By transitivity on (1) and (b), we have (2) $n'' < n$. Apply (k-qitm-syntrans) to (2).

Case (k-ana). In this case, $\sigma = \mathsf{ana}[n''](\sigma')$ and (1) $n'' < n'$ and (2) $\mathbf{\Delta}\ \mathbf{\Gamma} \vdash^{n'}_\Phi \sigma' :: \mathsf{Ty}$. By transitivity on (1) and (b), we have (3) $n'' < n$. By the IH on (2) and (b), we have (4) $\mathbf{\Delta}\ \mathbf{\Gamma} \vdash^{\sigma'}_n \mathsf{Ty} ::$. Apply (k-ana) to (3) and (4).

Case (k-syn). In this case, $\sigma = \mathsf{syn}[n'']$ and (1) $n'' < n'$. By transitivity on (1) and (b), we have (2) $n'' < n$. Apply (k-syn) to (2). □

3.4 Static Dynamics

Figures 23 to 30 give the dynamics of the SL as a structural operational semantics. Although the number of rules are large, the majority of them are uninteresting. Figure 23 simply gives the standard dynamics of the core language (i.e. these rules come from [1], with an $\mathcal{A}$ tacked on to each judgement but never inspected). We assume standard substitution lemmas:

Assumption 9 (Static Type Variable Substitution)**.** *If (a)* $\boldsymbol{\Delta}, \boldsymbol{\alpha}\ \boldsymbol{\Gamma} \vdash^n_\Phi \sigma :: \kappa$ *and (b)* $\boldsymbol{\Delta} \vdash \kappa'$ *then* $\boldsymbol{\Delta}\ [\kappa'/\boldsymbol{\alpha}]\boldsymbol{\Gamma} \vdash^n_\Phi [\kappa'/\boldsymbol{\alpha}]\sigma :: [\kappa'/\boldsymbol{\alpha}]\kappa$*.*

Assumption 10 (Static Term Variable Substitution)**.** *If (a)* $\boldsymbol{\Delta}\ \boldsymbol{\Gamma}, \boldsymbol{x} :: \kappa' \vdash^n_\Phi \sigma :: \kappa$ *and (b)* $\boldsymbol{\Delta}\ \boldsymbol{\Gamma} \vdash^n_\Phi \sigma' :: \kappa'$ *then* $\boldsymbol{\Delta}\ \boldsymbol{\Gamma} \vdash^n_\Phi [\sigma'/\boldsymbol{x}]\sigma :: \kappa$*.*

Figure 24 gives the dynamics of types and type case analysis.

Definition 1 (Types)**.** *We write* $\sigma\ \mathtt{type}_\Phi$ *iff* $\emptyset\ \emptyset \vdash^n_\Phi \sigma :: \mathsf{Ty}$ *and* $\sigma\ \mathtt{val}_{\overline{e};\Upsilon;\Phi}$*.*

Figures 26, 28 and 29 simply recurse through forms shared between the internal language and the translational internal language, to get to one of the forms that are unique to the translational internal language, defined in Figures 25 and 27. The following lemmas hold:

Lemma 15 (Stepping Quoted Translational Internal Types)**.** *If* $\blacktriangleright(\hat{\tau}) \mapsto_{\mathcal{A}} \sigma'$ *then* $\sigma' = \blacktriangleright(\hat{\tau}')$*.*

Proof. By rule induction on the assumption. All rules that apply are given in Figure 26, and in all cases the right hand side of the stepping judgement is of the form $\blacktriangleright(\hat{\tau}')$. □

Lemma 16 (Stepping Quoted Translational Internal Terms)**.** *If* $\rhd(\hat{\iota}) \mapsto_{\mathcal{A}} \sigma'$ *then* $\sigma' = \rhd(\hat{\iota}')$*.*

Proof. By rule induction on the assumption. All rules that apply are given in Figure 28 and 29, and in all cases the right hand side of the stepping judgement is of the form $\rhd(\hat{\iota}')$. □

Figure 30 gives the dynamics of the SL-EL interface.

3.4.1 Canonical Forms

Lemma 17 (Static Canonical Forms)**.** *Letting* $\mathcal{A} = \overline{e}; \Upsilon; \Phi$*, if (a)* $\emptyset\ \emptyset \vdash^n_\Phi \sigma :: \kappa$ *and (b)* $\sigma\ \mathtt{val}_{\mathcal{A}}$ *then:*

1. *If* $\kappa = \kappa_1 \to \kappa_2$*, then (i)* $\sigma = \lambda\boldsymbol{x}{::}\kappa_1.\sigma'$ *and (ii)* $\emptyset\ \emptyset, \boldsymbol{x} :: \kappa_1 \vdash^n_\Phi \sigma' :: \kappa_2$*.*

2. *If* $\kappa = \forall(\boldsymbol{\alpha}.\kappa)$*, then (i)* $\sigma = \Lambda(\boldsymbol{\alpha}.\sigma')$ *and (ii)* $\emptyset, \boldsymbol{\alpha}\ \emptyset \vdash^n_\Phi \sigma' :: \kappa$*.*

3. *If* $\kappa = \mathsf{List}[\kappa]$*, then either (i)* $\sigma = \mathsf{nil}[\kappa]$ *or (ii) (I)* $\sigma = \mathsf{cons}(\sigma_{hd}; \sigma_{tl})$ *and (II)* $\emptyset\ \emptyset \vdash^n_\Phi \sigma_{hd} :: \kappa$ *and (III)* $\sigma_{hd}\ \mathtt{val}_{\mathcal{A}}$ *and (IV)* $\emptyset\ \emptyset \vdash^n_\Phi \sigma_{tl} :: \mathsf{List}[\kappa]$ *and (V)* $\sigma_{tl}\ \mathtt{val}_{\mathcal{A}}$*.*

4. *If* $\kappa = 1$*, then* $\sigma = ()$*.*

5. *If* $\kappa = \kappa_1 \times \kappa_2$ *then (i)* $\sigma = (\sigma_1, \sigma_2)$ *and (ii)* $\emptyset\;\emptyset \vdash^n_\Phi \sigma_1 :: \kappa_1$ *and (iii)* $\sigma_1\ \mathtt{val}_\mathcal{A}$ *and (iv)* $\emptyset\;\emptyset \vdash^n_\Phi \sigma_2 :: \kappa_2$ *and (v)* $\sigma_2\ \mathtt{val}_\mathcal{A}$*.*

6. *If* $\kappa = \kappa_1 + \kappa_2$ *then either (I) (i)* $\sigma = \mathsf{inl}[\kappa](\sigma')$ *and (ii)* $\emptyset\;\emptyset \vdash^n_\Phi \sigma' :: \kappa_1$ *and (iii)* $\sigma'\ \mathtt{val}_\mathcal{A}$*, or (II) (i)* $\sigma = \mathsf{inr}[\kappa](\sigma')$ *and (ii)* $\emptyset\;\emptyset \vdash^n_\Phi \sigma' :: \kappa_2$ *and (iii)* $\sigma'\ \mathtt{val}_\mathcal{A}$*.*

7. *If* $\kappa = \mathsf{Ty}$ *then either*

   (I) $\sigma = \rightharpoonup\langle(\sigma_1, \sigma_2)\rangle$ *and (i)* $\sigma_1\ \mathtt{type}_\Phi$ *and (ii)* $\sigma_2\ \mathtt{type}_\Phi$ *and (iii)* $\emptyset\;\emptyset \vdash^n_\Phi (\sigma_1, \sigma_2) :: \mathsf{Ty} \times \mathsf{Ty}$ *and (iv)* $(\sigma_1, \sigma_2)\ \mathtt{val}_\mathcal{A}$*; or*

   (II) $\sigma = \mathtt{TC}\langle\sigma_{tyidx}\rangle$ *and (i)* $\mathsf{tycon}\ \mathtt{TC}\ \{\omega\} \sim \mathsf{tcsig}[\kappa_{tyidx}]\ \{\chi\} \in \Phi$ *and (ii)* $\emptyset\;\emptyset \vdash^0_\Phi \sigma :: \kappa_{tyidx}$ *and (iii)* $\sigma_{tyidx}\ \mathtt{val}_\mathcal{A}$*; or*

   (III) $\sigma = \mathsf{other}[m;\kappa]\langle(\sigma_{data}, \blacktriangleright(\hat{\tau}))\rangle$ *and (i)* $\emptyset \vdash \kappa$ *and (ii)* $\emptyset\;\emptyset \vdash^n_\Phi \sigma_{data} :: \kappa$ *and (iii)* $\sigma_{data}\ \mathtt{val}_\mathcal{A}$ *and (iv)* $\emptyset\;\emptyset \vdash^n_\Phi \blacktriangleright(\hat{\tau}) :: \mathsf{ITy}$ *and (v)* $\blacktriangleright(\hat{\tau})\ \mathtt{val}_\mathcal{A}$*.*

8. *If* $\kappa = \mathsf{ITy}$ *then* $\sigma = \blacktriangleright(\hat{\tau})$ *and either*

   (I) $\hat{\tau} = \mathsf{trans}(\sigma')$ *and* $\sigma'\ \mathtt{type}_\Phi$*; or*

   (II) The outer form of $\hat{\tau}$ *is shared with* τ *and if* $\hat{\tau}'$ *is a sub-term of* $\hat{\tau}$ *then* $\emptyset\;\emptyset \vdash^0_\Phi \blacktriangleright(\hat{\tau}') :: \mathsf{ITy}$ *and* $\blacktriangleright(\hat{\tau}')\ \mathtt{val}$*.*

9. *If* $\kappa = \mathsf{ITm}$ *then* $\sigma = \rhd(\hat{\iota})$ *and either*

   (I) $\hat{\iota} = \mathsf{anatrans}[n'](\sigma')$ *and (i)* $n' < n$ *and* $\mathcal{A} = \overline{e}; \Upsilon; \Phi'$ *and (ii)* $|\overline{e}| = n''$ *and (iii)* $n' < n''$ *and (iv)* $\sigma'\ \mathtt{type}_\Phi$*; or*

   (II) $\hat{\iota} = \mathsf{syntrans}[n]$ *and (i)* $n' < n$ *and* $\mathcal{A} = \overline{e}; \Upsilon; \Phi'$ *and (ii)* $|\overline{e}| = n''$ *and (iii)* $n' < n''$*.*

   (III) The outer form of $\hat{\iota}$ *is shared with* ι *and if* $\hat{\iota}'$ *is a sub-term of* $\hat{\iota}$ *then* $\emptyset\;\emptyset \vdash^n_\Phi \rhd(\hat{\iota}') :: \mathsf{ITm}$ *and* $\rhd(\hat{\iota}')\ \mathtt{val}_\mathcal{A}$ *and if* $\hat{\tau}$ *is a sub-term of* $\hat{\iota}$ *then* $\emptyset\;\emptyset \vdash^0_\Phi \blacktriangleright(\hat{\tau}) :: \mathsf{ITy}$ *and* $\blacktriangleright(\hat{\tau})\ \mathtt{val}$*.*

Proof. The proofs for parts 1-6 follow the usual script and are omitted [1].

7. Rule induction on (a) and (b). There is one form for which a kinding rule where κ can be Ty and a value rule are both defined, $\sigma = c\langle\sigma_{\text{tyidx}}\rangle$. The only value rule that applies is (n-ty-val), so (1) $\sigma_{\text{tyidx}}\ \mathtt{val}_\mathcal{A}$. Case analysis on c:

   Case $c = \rightharpoonup$. The only kinding rule that applies is (k-ty-parr), so (2) $\emptyset\;\emptyset \vdash^n_\Phi \sigma_{\text{tyidx}} :: \mathsf{Ty} \times \mathsf{Ty}$.

   By the IH on (2) and (1), we thus have that $\sigma_{\text{tyidx}} = (\sigma_1, \sigma_2)$ where (3) $\emptyset\;\emptyset \vdash^n_\Phi \sigma_1 :: \mathsf{Ty}$ and (4) $\sigma_1\ \mathtt{val}_\mathcal{A}$ and (5) $\emptyset\;\emptyset \vdash^n_\Phi \sigma_2 :: \mathsf{Ty}$ and (6) $\sigma_2\ \mathtt{val}_\mathcal{A}$.

   By the definition of types applied to (3) and (4), we have (7) $\sigma_1\ \mathtt{type}_\Phi$.

   By the definition of types applied to (5) and (6), we have (8) $\sigma_2\ \mathtt{type}_\Phi$.

   By (k-prod) on (3) and (5), we have (9) $\emptyset\;\emptyset \vdash^n_\Phi (\sigma_1, \sigma_2) :: \mathsf{Ty} \times \mathsf{Ty}$.

By (n-pair-val) on (4) and (6), we have (10) $(\sigma_1, \sigma_2)\ \mathtt{val}_{\mathcal{A}}$.

Thus, we can complete our proof by (I) with (7), (8), (9) and (10).

Case $c = \texttt{TC}$. The only kinding rule that applies is (k-ty-ext), so (2) $\mathsf{tycon}\ \texttt{TC}\ \{\omega\} \sim \mathsf{tcsig}[\kappa_{\text{tyidx}}]\ \{\chi\} \in \Phi$ and (3) $\emptyset\ \emptyset \vdash^0_\Phi \sigma :: \kappa_{\text{tyidx}}$. Thus, we can complete our proof by (II) with (2), (3) and (1).

Case $c = \mathsf{other}[m; \kappa]$. The only kinding rule that applies is (k-ty-other), so (2) $\emptyset \vdash \kappa$ and (3) $\emptyset\ \emptyset \vdash^n_\Phi \sigma_{\text{tyidx}} :: \kappa \times \mathsf{ITy}$. By the IH on (3), (b), (c) and (1), we thus have that $\sigma_{\text{tyidx}} = (\sigma_{\text{data}}, \sigma_{\text{trans}})$ where (4) $\emptyset\ \emptyset \vdash^n_\Phi \sigma_{\text{data}} :: \kappa$ and (5) $\sigma_{\text{data}}\ \mathtt{val}_{\mathcal{A}}$ and (6) $\emptyset\ \emptyset \vdash^n_\Phi \sigma_{\text{trans}} :: \mathsf{ITy}$ and (7) $\sigma_{\text{trans}}\ \mathtt{val}_{\mathcal{A}}$.

By the IH on (6) and (7), we have that $\sigma_{\text{trans}} = \blacktriangleright(\hat{\tau})$.

Thus, we can complete our proof by (III) with (2), (4), (5), (6) and (7).

8. Rule induction on (a) and (b). All forms that can be given kind ITy and define a value rule are of the form $\blacktriangleright(\hat{\tau})$. Case analysis on the form of $\hat{\tau}$:

   Case $\hat{\tau} = \blacktriangleleft(\sigma)$. There is no value rule for this form.

   Case $\hat{\tau} = \mathsf{trans}(\sigma')$. The only kinding rule for this form is (k-qity-trans), so (1) $\emptyset\ \emptyset \vdash^n_\Phi \sigma' :: \mathsf{Ty}$.

   The only value rule for this form is (n-q-tytrans-val), so (2) $\sigma'\ \mathtt{val}_{\mathcal{A}}$.

   By the definition of types on 1 and 2, we have (3) $\sigma'\ \mathtt{type}_\Phi$. We can thus complete our proof by (I) on (3).

   Case [$\hat{\tau}$ has form shared with τ]. All kinding rules for these forms are given in Figure 19. For each form, there is one rule. That rule has the property that (1) if $\hat{\tau}'$ is a direct sub-term of $\hat{\tau}$, then there is a premise of the form $\emptyset\ \emptyset \vdash^n_\Phi \blacktriangleright(\hat{\tau}') :: \mathsf{ITy}$.

   All rules in the static dynamics for these forms are given in Figure 26. For each form, there is one value rule. This rule has the property that (2) if $\hat{\tau}'$ is a sub-term of $\hat{\tau}$, the rule has a premise of the form (2) $\blacktriangleright(\hat{\tau}')\ \mathtt{val}_{\mathcal{A}}$.

   We can thus complete our proof by (II) on (1) and (2) in each case.

9. Rule induction on (a) and (b). All forms that can be given kind ITm and define a value rule are of the form $\triangleright(\hat{\iota})$. Case analysis on the form of $\hat{\iota}$:

   Case $\hat{\iota} = \triangleleft(\sigma')$. There is no value rule for this form.

   Case $\hat{\iota} = \mathsf{anatrans}[n'](\sigma')$. The only kinding rule for this form is (k-qitm-anatrans), so (1) $n' < n$ and (2) $\emptyset\ \emptyset \vdash^n_\Phi \sigma' :: \mathsf{Ty}$.

   The only value rule for this form is (n-q-anatrans-val), so $\mathcal{A} = \overline{e}; \Upsilon; \Phi'$ and (3) $\sigma'\ \mathtt{val}_{\mathcal{A}}$ and (4) $|\overline{e}| = n''$ and (5) $n' < n''$.

   By the definition of types on (2) and (3), we have (6) $\sigma'\ \mathtt{type}_\Phi$.

   We can thus complete our proof by (I) on (1), (4), (5) and (6).

Case $\hat{\iota} = \mathsf{syntrans}[n']$. The only kinding rule for this form is (k-qitm-syntrans) so (1) $n' < n$.

The only value rule for this form is (n-q-syntrans-val) so $\mathcal{A} = \overline{e}; \Upsilon; \Phi'$ and (2) $|\overline{e}| = n''$ and (3) $n' < n''$.

We can thus complete our proof by (II) on (1), (2) and (3).

Case [$\hat{\iota}$ has form shared with ι]. All kinding rules for these forms are given in Figure 21. For each form, there is one rule. That rule has the property that (1) if $\hat{\iota}'$ is a direct sub-term of $\hat{\iota}$, then there is a premise of the form $\emptyset\ \emptyset \vdash^n_\Phi \rhd(\hat{\iota}') :: \mathsf{ITm}$, and (2) if $\hat{\tau}'$ is a direct sub-term of $\hat{\iota}$, then there is a premise of the form $\emptyset\ \emptyset \vdash^n_\Phi \blacktriangleright(\hat{\tau}') :: \mathsf{ITy}$.

All rules in the static dynamics for these forms are given in Figure 28 and 29. For each form, there is one value rule. This rule has the property that (3) if $\hat{\iota}'$ is a sub-term of $\hat{\iota}$, then there is a premise of the form $\rhd(\hat{\iota}')\ \mathtt{val}_{\mathcal{A}}$, and (4) if $\hat{\tau}'$ is a sub-term of $\hat{\iota}$, then there is a premise of the form $\blacktriangleright(\hat{\tau}')\ \mathtt{val}_{\mathcal{A}}$.

We can thus complete our proof by (III) on (1), (2), (3) and (4) in each case.

□

As suggested by Lemma 17, only static values of kind ITm are argument-dependent (i.e. are indexed by a natural number). It is helpful to have a lemma that says that values of other kinds, in particular equality kinds, are argument independent.

Lemma 18 (Values of Equality Kind are Argument Independent)**.** *If (a)* $\emptyset\ \emptyset \vdash^n_\Phi \sigma :: \kappa$ *and (b)* $\sigma\ \mathtt{val}_{\mathcal{A}}$ *and (c)* $\vdash \kappa\ \mathtt{eq}$ *then (i)* $\emptyset\ \emptyset \vdash^0_\Phi \sigma :: \kappa$ *and (ii)* $\sigma\ \mathtt{val}_{\mathcal{A}'}$.

Proof. By rule induction on (c).

Case (keq-list). $\kappa = \mathsf{List}[\kappa']$ and $\vdash \kappa'\ \mathtt{eq}$. Rule induction on (a) and (b). There are only twos forms that can have kind $\mathsf{List}[\kappa']$ and define value rules, $\sigma = \mathsf{nil}[\kappa']$ and $\sigma = \mathsf{cons}(\sigma_1; \sigma_2)$. The only kinding rules that apply are (k-nil) and (k-cons), respectively. The only value rules that apply are (n-nil-val) and (n-cons-val), respectively. All of these rules are defined for all n and all $\mathcal{A}$, and only kind check against equality kinds κ' and $\mathsf{List}[\kappa']$ in their premises, so we can simply re-apply them inductively to conclude (i) and (ii) in each case.

Case (keq-unit). $\kappa = 1$. Rule induction on (a) and (b). There is only one form that can have kind 1 and defines a value rule, $\sigma = ()$. The only kinding rule that applies is (k-unit) and the only value rule that applies is (n-triv-val). Both are defined for all n and all $\mathcal{A}$, and have no premises, so we can simply re-apply them to conclude (i) and (ii).

Case (keq-prod). $\kappa = \kappa_1 \times \kappa_2$ and $\vdash \kappa_1\ \mathtt{eq}$ and $\vdash \kappa_2\ \mathtt{eq}$. Rule induction on (a) and (b). There is only one form that can have kind $\kappa_1 \times \kappa_2$ and defines a value rule, $\sigma = (\sigma_1, \sigma_2)$. The only kinding rule that applies is (k-pair). The only value rule that applies is (n-pair-val). Both are defined for all n and all $\mathcal{A}$, and only kind check against equality kinds κ_1 or κ_2 in the premises, so we can simply re-apply them inductively to conclude (i) and (ii).

Case (keq-sum). $\kappa = \kappa_1 + \kappa_2$. Rule induction on (a) and (b). There are two forms that can have kind $\kappa_1 + \kappa_2$ and define value rules, $\sigma = \mathsf{inl}[\kappa](\sigma_1)$ and $\sigma = \mathsf{inr}[\kappa](\sigma_2)$. The only kinding rules that apply are (k-inl) and (k-inr), respectively. The only value rules that apply are (n-inl-val) and (n-inr-val), respectively. All of these rules are defined for all n and all $\mathcal{A}$, and only kind check against equality kinds κ_1 or κ_2 in the premises, so we can simply re-apply them inductively to conclude (i) and (ii) in each case.

Case (keq-ty). $\kappa = \mathsf{Ty}$. Rule induction on (a) and (b). There is one form that can have kind Ty and defines value rules, $\sigma = c\langle\sigma_{\text{tyidx}}\rangle$. The only value rule that applies is (n-ty-val), so we have (1) $\sigma_{\text{tyidx}}\ \mathtt{val}_{\mathcal{A}}$. Three kinding rules might apply:

> **Case** (k-ty-parr). In this case, $c =\rightharpoonup$ and (2) $\emptyset\ \emptyset \vdash^n_\Phi \sigma_{\text{tyidx}} :: \mathsf{Ty} \times \mathsf{Ty}$. By (keq-prod) and (keq-ty) and (keq-ty), we have that (3) $\vdash \mathsf{Ty} \times \mathsf{Ty}\ \mathtt{eq}$. Applying the IH to (2), (1) and (3), we have that (4) $\emptyset\ \emptyset \vdash^0_\Phi \sigma_{\text{tyidx}} :: \mathsf{Ty} \times \mathsf{Ty}$ and (6) $\sigma_{\text{tyidx}}\ \mathtt{val}_{\mathcal{A}'}$. By (k-ty-parr) on (5), we have (i). By (n-ty-val) on (6), we have (ii).
>
> **Case** (k-ty-ext). In this case, $c = \textsc{tc}$ and (2) $\mathsf{tycon}\ \textsc{tc}\ \{\theta\} \sim \mathsf{tcsig}[\kappa_{\text{tyidx}}]\ \{\chi\} \in \Phi$ and (3) $\emptyset\ \emptyset \vdash^n_\Phi \sigma_{\text{tyidx}} :: \kappa_{\text{tyidx}}$. By Lemma 7 on (2), we have that (4) $\vdash \mathsf{tcsig}[\kappa_{\text{tyidx}}]\ \{\chi\}$. By Lemma 4 on (4), we have that (5) $\vdash \kappa_{\text{tyidx}}\ \mathtt{eq}$. By the IH on (3), (1) and (5) we have (6) $\emptyset\ \emptyset \vdash^0_\Phi \sigma_{\text{tyidx}} :: \kappa_{\text{tyidx}}$ and (7) $\sigma_{\text{tyidx}}\ \mathtt{val}_{\mathcal{A}'}$. Applying (k-ty-ext) to (2) and (6), we have (i). Applying (n-ty-val) to (1), we have (ii).
>
> **Case** (k-ty-other). In this case, $c = \mathsf{other}[m;\kappa']$ and (2) $\vdash \kappa'\ \mathtt{eq}$ and (3) $\emptyset\ \emptyset \vdash^n_\Phi \sigma_{\text{tyidx}} :: \kappa' \times \mathsf{ITy}$. By (keq-ity), we have (4) $\vdash \mathsf{ITy}\ \mathtt{eq}$. By (keq-prod) on (2) and (4), we have (5) $\vdash \kappa' \times \mathsf{ITy}\ \mathtt{eq}$. By the IH on (3), (1) and (5), we have (6) $\emptyset\ \emptyset \vdash^0_{\sigma_{\text{tyidx}}} \kappa' \times \mathsf{ITy} ::$ and (7) $\sigma_{\text{tyidx}}\ \mathtt{val}_{\mathcal{A}'}$. By (k-ty-other) on (2) and (6), we have (i). By (n-ty-val) on (7), we have (ii).

Case (keq-ity). $\kappa = \mathsf{ITy}$. Rule induction on (a) and (b). The only forms that can have kind ITy and define value rules are of the form $\blacktriangleright(\hat{\tau})$. The kinding rules for these are defined in Figures 18 and 19.

> **Case** [k-qity-trans]. In this case, $\hat{\tau} = \mathsf{trans}(\sigma')$ and (1) $\emptyset\ \emptyset \vdash^n_\Phi \sigma' :: \mathsf{Ty}$. The only value rule that applies is (n-q-tytrans-val), so (2) $\sigma'\ \mathtt{val}_{\mathcal{A}}$. Note that (3) $\vdash \mathsf{Ty}\ \mathtt{eq}$ by (keq-ty). By the IH on (1), (2) and (3), we have (4) $\emptyset\ \emptyset \vdash^0_\Phi \sigma' :: \mathsf{Ty}$ and (5) $\sigma'\ \mathtt{val}_{\mathcal{A}'}$. By (k-qity-trans) on (4), we have (i). By (n-q-tytrans-val) on (5), we have (ii).
>
> **Case** [all rules in Figure 19]. Observe that all rules where $\hat{\tau}$ is of shared form are defined for all n, and that for all sub-terms $\hat{\tau}'$ of $\hat{\tau}$, there is a premise of the form $\emptyset\ \emptyset \vdash^n_\Phi \blacktriangleright(\hat{\tau}') :: \mathsf{ITy}$, and there are no other premises.
>
> The value rules for these are defined in Figures 26. Observe that all rules are defined for all $\mathcal{A}$ and for all sub-terms $\hat{\tau}'$ of $\hat{\tau}$, there is a premise of the form $\blacktriangleright(\hat{\tau}')\ \mathtt{val}_{\mathcal{A}}$, and there are no other premises.
>
> Thus, in every case, we can apply the IH to each sub-term of $\hat{\tau}$, then reapply the same rules to derive (i) and (ii), respectively.

□

Lemma 19 (Types are Argument Independent). *$\sigma\ \mathtt{type}_\Phi$ iff (a) $\emptyset\ \emptyset \vdash^0_\Phi \sigma :: \mathsf{Ty}$ and (b) $\sigma\ \mathtt{val}_{\cdot;\emptyset;\Phi}$.*

Proof. In the forward direction, by the definition of $\sigma~\mathtt{type}_\Phi$ we have (1) $\emptyset~\emptyset \vdash^n_\Phi \sigma :: \mathsf{Ty}$ and (2) $\sigma~\mathtt{val}_{\overline{e};\Upsilon;\Phi}$. By (keq-ty), we have (3) $\vdash \mathsf{Ty}~\mathtt{eq}$. By Lemma 18 on 1, 2 and 3, we have (a) and (b).

In the backwards direction, apply the definition of $\sigma~\mathtt{type}_\Phi$ to (a) and (b). □

3.5 Kind Safety

Kind safety ensures that normalization of well-kinded static terms cannot go wrong. We can take a standard progress and preservation based approach.

Theorem 1 (Static Progress)**.** *Letting $\mathcal{A} = \overline{e}; \Upsilon; \Phi$, if (a) $\emptyset~\emptyset \vdash^n_\Phi \sigma :: \kappa$ and (b) $|\overline{e}| = n$ then (I) $\sigma \mapsto_{\mathcal{A}} \sigma'$ or (II) $\sigma~\mathtt{val}_{\mathcal{A}}$ or (III) $\sigma~\mathtt{err}_{\mathcal{A}}$.*

Proof. Rule induction on (a).

Cases [All rules in Figure 16]. The proof of progress for core language constructs follows the standard script [1] and is omitted.

Case (k-ty-parr). In this case, we have $\sigma = \rightharpoonup\langle\sigma_{\text{tyidx}}\rangle$ and $\kappa = \mathsf{Ty}$ and (1) $\emptyset~\emptyset \vdash^n_\Phi \sigma_{\text{tyidx}} :: \mathsf{Ty} \times \mathsf{Ty}$. Applying the IH to (1) and (b), we have three cases:

> **Case** (2) $\sigma_{\text{tyidx}} \mapsto_{\mathcal{A}} \sigma'_{\text{tyidx}}$. Applying (n-ty-step) to (2), we have (3) $\rightharpoonup\langle\sigma_{\text{tyidx}}\rangle \mapsto_{\mathcal{A}} \rightharpoonup\langle\sigma'_{\text{tyidx}}\rangle$. We can conclude by (I) on (3).
>
> **Case** (2) $\sigma_{\text{tyidx}}~\mathtt{val}_{\mathcal{A}}$. Applying (n-ty-val) to (2), we have (3) $\rightharpoonup\langle\sigma_{\text{tyidx}}\rangle~\mathtt{val}_{\mathcal{A}}$. We can conclude by (II) on (3).
>
> **Case** (2) $\sigma_{\text{tyidx}}~\mathtt{err}_{\mathcal{A}}$. Applying (n-ty-err-prop) to (2), we have (3) $\rightharpoonup\langle\sigma_{\text{tyidx}}\rangle~\mathtt{err}_{\mathcal{A}}$. We can conclude by (III) on (3).

Case (k-ty-ext). In this case, we have $\sigma = \textsc{tc}\langle\sigma_{\text{tyidx}}\rangle$ and $\kappa = \mathsf{Ty}$ and (1) $\emptyset~\emptyset \vdash^n_\Phi \sigma_{\text{tyidx}} :: \kappa_{\text{tyidx}}$. Applying the IH to (1) and (b), we have three cases:

> **Case** (2) $\sigma_{\text{tyidx}} \mapsto_{\mathcal{A}} \sigma'_{\text{tyidx}}$. Applying (n-ty-step) to (2), we have (3) $\textsc{tc}\langle\sigma_{\text{tyidx}}\rangle \mapsto_{\mathcal{A}} \rightharpoonup\langle\sigma'_{\text{tyidx}}\rangle$. We can conclude by (I) on (3).
>
> **Case** (2) $\sigma_{\text{tyidx}}~\mathtt{val}_{\mathcal{A}}$. Applying (n-ty-val) to (2), we have (3) $\textsc{tc}\langle\sigma_{\text{tyidx}}\rangle~\mathtt{val}_{\mathcal{A}}$. We can conclude by (II) on (3).
>
> **Case** (2) $\sigma_{\text{tyidx}}~\mathtt{err}_{\mathcal{A}}$. Applying (n-ty-err-prop) to (2), we have (3) $\textsc{tc}\langle\sigma_{\text{tyidx}}\rangle~\mathtt{err}_{\mathcal{A}}$. We can conclude by (III) on (3).

Case (k-ty-other). In this case, we have $\sigma = \mathsf{other}[m;\kappa']\langle\sigma_{\text{tyidx}}\rangle$ and $\kappa = \mathsf{Ty}$ and (1) $\emptyset~\emptyset \vdash^n_\Phi \sigma_{\text{tyidx}} :: \kappa' \times \mathsf{ITy}$. Applying the IH to (1) and (b), we have three cases:

> **Case** (2) $\sigma_{\text{tyidx}} \mapsto_{\mathcal{A}} \sigma'_{\text{tyidx}}$. Applying (n-ty-step) to (2), we have (3) $\textsc{tc}\langle\sigma_{\text{tyidx}}\rangle \mapsto_{\mathcal{A}} \mathsf{other}[m;\kappa']\langle\sigma'_{\text{tyidx}}\rangle$. We can conclude by (I) on (3).
>
> **Case** (2) $\sigma_{\text{tyidx}}~\mathtt{val}_{\mathcal{A}}$. Applying (n-ty-val) to (2), we have (3) $\mathsf{other}[m;\kappa']\langle\sigma_{\text{tyidx}}\rangle~\mathtt{val}_{\mathcal{A}}$. We can conclude by (II) on (3).
>
> **Case** (2) $\sigma_{\text{tyidx}}~\mathtt{err}_{\mathcal{A}}$. Applying (n-ty-err-prop) to (2), we have (3) $\mathsf{other}[m;\kappa']\langle\sigma_{\text{tyidx}}\rangle~\mathtt{err}_{\mathcal{A}}$. We can conclude by (III) on (3).

Case (k-tycase-parr). In this case, we have $\sigma = \mathsf{tycase}[\rightharpoonup](\sigma'; \boldsymbol{x}.\sigma_1; \sigma_2)$ and (1) $\emptyset\ \emptyset \vdash^n_\Phi \sigma' :: \mathsf{Ty}$. Applying the IH to (1) and (b), we have three cases:

Case (2) $\sigma' \mapsto_{\mathcal{A}} \sigma''$. Applying (n-tycase-step) to (2), we have (3)

$$\mathsf{tycase}[\rightharpoonup](\sigma'; \boldsymbol{x}.\sigma_1; \sigma_2) \mapsto_{\mathcal{A}} \mathsf{tycase}[\rightharpoonup](\sigma''; \boldsymbol{x}.\sigma_1; \sigma_2)$$

We can conclude by (I) on (3).

Case (2) $\sigma'\ \texttt{val}_{\mathcal{A}}$. By Lemma 17.7 on (1) and (2), we have three cases:

Case $\sigma' = \rightharpoonup\langle(\sigma_1', \sigma_2')\rangle$. Applying (n-tycase-elim-1) to (2), we have (3)

$$\mathsf{tycase}[\rightharpoonup](\rightharpoonup\langle(\sigma_1', \sigma_2')\rangle; \boldsymbol{x}.\sigma_1; \sigma_2) \mapsto_{\mathcal{A}} [(\sigma_1', \sigma_2')/\boldsymbol{x}]\sigma_1$$

We can conclude by (I) on (3).

Case $\sigma' = \texttt{TC}\langle\sigma_{\text{tyidx}}\rangle$. We have (3) $\rightharpoonup\ \neq \texttt{TC}$. Applying (n-tycase-elim-2) to (2) and (3), we have (4)

$$\mathsf{tycase}[\rightharpoonup](\texttt{TC}\langle\sigma_{\text{tyidx}}\rangle; \boldsymbol{x}.\sigma_1; \sigma_2) \mapsto_{\mathcal{A}} \sigma_2$$

We can conclude by (I) on (4).

Case $\sigma' = \mathsf{other}[m; \kappa']\langle(\sigma_{\text{data}}, \blacktriangleright(\hat{\tau}))\rangle$. We have (3) $\rightharpoonup\ \neq \mathsf{other}[m; \kappa']$. Applying (n-tycase-elim-2) to (2) and (3), we have (4)

$$\mathsf{tycase}[\rightharpoonup](\mathsf{other}[m; \kappa']\langle(\sigma_{\text{data}}, \blacktriangleright(\hat{\tau}))\rangle; \boldsymbol{x}.\sigma_1; \sigma_2) \mapsto_{\mathcal{A}} \sigma_2$$

We can conclude by (I) on (4).

Case (2) $\sigma'\ \texttt{err}_{\mathcal{A}}$. Applying (n-tycase-err-prop) to (2), we have (3)

$$\mathsf{tycase}[\rightharpoonup](\sigma'; \boldsymbol{x}.\sigma_1; \sigma_2)\ \texttt{err}_{\mathcal{A}}$$

We can conclude by (III) on (3).

Case (k-tycase-ext). In this case, we have $\sigma = \mathsf{tycase}[\texttt{TC}](\sigma'; \boldsymbol{x}.\sigma_1; \sigma_2)$ and (1) $\emptyset\ \emptyset \vdash^n_\Phi \sigma' :: \mathsf{Ty}$. Applying the IH to (1) and (b), we have three cases:

Case (2) $\sigma' \mapsto_{\mathcal{A}} \sigma''$. Applying (n-tycase-step) to (2), we have (3)

$$\mathsf{tycase}[\texttt{TC}](\sigma'; \boldsymbol{x}.\sigma_1; \sigma_2) \mapsto_{\mathcal{A}} \mathsf{tycase}[\texttt{TC}](\sigma''; \boldsymbol{x}.\sigma_1; \sigma_2)$$

We can conclude by (I) on (3).

Case (2) $\sigma'\ \texttt{val}_{\mathcal{A}}$. By Lemma 17.7 on (1) and (2), we have three cases:

Case $\sigma' = \rightharpoonup\langle(\sigma_1', \sigma_2')\rangle$. We have (3) $\textsc{tc} \neq \rightharpoonup$. Applying (n-tycase-elim-2) to (2) and (3), we have (4)

$$\mathsf{tycase}[\textsc{tc}](\rightharpoonup\langle(\sigma_1', \sigma_2')\rangle; \boldsymbol{x}.\sigma_1; \sigma_2) \mapsto_{\mathcal{A}} \sigma_2$$

We can conclude by (I) on (4).

Case $\sigma' = \textsc{tc}'\langle\sigma_{\text{tyidx}}\rangle$ and (3) $\sigma_{\text{tyidx}}\ \mathtt{val}_{\mathcal{A}}$. There are two cases:

Case $\textsc{tc} = \textsc{tc}'$. Applying (n-tycase-elim-1) to (3), we have (4)

$$\mathsf{tycase}[\textsc{tc}](\textsc{tc}\langle\sigma_{\text{tyidx}}\rangle; \boldsymbol{x}.\sigma_1; \sigma_2) \mapsto_{\mathcal{A}} [\sigma_{\text{tyidx}}/\boldsymbol{x}]\sigma_1$$

We can conclude by (I) on (4).

Case (4) $\textsc{tc} \neq \textsc{tc}'$. Applying (n-tycase-elim-2) to (3) and (4), we have (5)

$$\mathsf{tycase}[\textsc{tc}](\textsc{tc}'\langle\sigma_{\text{tyidx}}\rangle; \boldsymbol{x}.\sigma_1; \sigma_2) \mapsto_{\mathcal{A}} \sigma_2$$

We can conclude by (I) on (5).

Case $\sigma' = \mathsf{other}[m; \kappa']\langle(\sigma_{\text{data}}, \blacktriangleright(\hat{\tau}))\rangle$. We have (3) $\textsc{tc} \neq \mathsf{other}[m; \kappa']$. Applying (n-tycase-elim-2) to (2) and (3), we have (4)

$$\mathsf{tycase}[\rightharpoonup](\mathsf{other}[m; \kappa']\langle(\sigma_{\text{data}}, \blacktriangleright(\hat{\tau}))\rangle; \boldsymbol{x}.\sigma_1; \sigma_2) \mapsto_{\mathcal{A}} \sigma_2$$

We can conclude by (I) on (4).

Case (2) $\sigma'\ \mathtt{err}_{\mathcal{A}}$. Applying (n-tycase-err-prop) to (2), we have (3)

$$\mathsf{tycase}[\rightharpoonup](\sigma'; \boldsymbol{x}.\sigma_1; \sigma_2)\ \mathtt{err}_{\mathcal{A}}$$

We can conclude by (III) on (3).

Case (k-qity-unquote). We have $\sigma = \blacktriangleright(\blacktriangleleft(\sigma'))$ and (1) $\emptyset\ \emptyset \vdash_{\Phi}^{n} \sigma' :: \mathsf{ITy}$. Applying the IH to (1) and (b), we have three cases:

Case (2) $\sigma' \mapsto_{\mathcal{A}} \sigma''$. Applying (n-q-t-unquote-step) to (2), we have (3) $\blacktriangleright(\blacktriangleleft(\sigma')) \mapsto_{\mathcal{A}} \blacktriangleright(\blacktriangleleft(\sigma''))$. We can conclude by (I) on (3).

Case (2) $\sigma'\ \mathtt{val}_{\mathcal{A}}$. Applying Lemma 17.8 to (1) and (2), we have that (3) $\sigma' = \blacktriangleright(\hat{\tau})$. Applying (n-q-t-unquote-elim) to (3), we have (4) $\blacktriangleright(\blacktriangleleft(\blacktriangleright(\hat{\tau}))) \mapsto_{\mathcal{A}} \blacktriangleright(\hat{\tau})$. We can conclude by (I) on (4).

Case (2) $\sigma'\ \mathtt{err}_{\mathcal{A}}$. Applying (n-q-t-unquote-err-prop) on (2), we have (3) $\blacktriangleright(\blacktriangleleft(\sigma'))\ \mathtt{err}_{\mathcal{A}}$. We can conclude by (III) on (3).

Case (k-qity-trans). We have $\sigma = \blacktriangleright(\mathsf{trans}(\sigma'))$ and (1) $\emptyset\ \emptyset \vdash_{\Phi}^{n} \sigma' :: \mathsf{Ty}$. Applying the IH to (1) and (b), we have three cases:

Case (2) $\sigma' \mapsto_{\mathcal{A}} \sigma''$. Applying (n-q-tytrans-step) to (2), we have (3) $\blacktriangleright(\mathsf{trans}(\sigma')) \mapsto_{\mathcal{A}} \blacktriangleright(\mathsf{trans}(\sigma''))$. We can conclude by (I) on (3).

Case (2) $\sigma'\ \mathtt{val}_{\mathcal{A}}$. Applying (n-q-tytrans-val) to (2), we have (3) $\blacktriangleright(\mathsf{trans}(\sigma'))\ \mathtt{val}_{\mathcal{A}}$. We can conclude by (I) on (3).

Case (2) $\sigma'\ \mathtt{err}_{\mathcal{A}}$. Applying (n-q-tytrans-err-prop) on (2), we have (3) $\blacktriangleright(\blacktriangleleft(\sigma'))\ \mathtt{err}_{\mathcal{A}}$. We can conclude by (III) on (3).

Cases [All rules in Figure 19]. We give an example base case and inductive cases with 0, 1 and 2 sub-terms. All rules with the same number of sub-terms follow the corresponding script, replacing only the forms and rule names:

Case (k-qity-alpha). We have that $\sigma = \blacktriangleright(\alpha)$. Applying (n-q-alpha-val), we have (1) $\blacktriangleright(\alpha)\ \mathtt{val}_{\mathcal{A}}$. We can conclude by (II) on (1).

Case (k-qity-forall). We have that $\sigma = \blacktriangleright(\forall(\alpha.\hat{\tau}))$ and (1) $\emptyset\ \emptyset \vdash^{n}_{\Phi} \blacktriangleright(\hat{\tau}) :: \mathsf{ITy}$. Applying the IH to (1) and (b), we have three cases:

Case (2) $\blacktriangleright(\hat{\tau}) \mapsto_{\mathcal{A}} \sigma'$. By Lemma 15 on (2), we have $\sigma' = \blacktriangleright(\hat{\tau}')$. Applying (n-q-forall-step) to (2), we have (3) $\blacktriangleright(\forall(\alpha.\hat{\tau})) \mapsto_{\mathcal{A}} \blacktriangleright(\forall(\alpha.\hat{\tau}'))$. We can conclude by (I) on (3).

Case (2) $\blacktriangleright(\hat{\tau})\ \mathtt{val}_{\mathcal{A}}$. Applying (n-q-forall-val) to (2), we have (3) $\blacktriangleright(\forall(\alpha.\hat{\tau}))\ \mathtt{val}_{\mathcal{A}}$. We can conclude by (II) on (3).

Case (2) $\blacktriangleright(\hat{\tau})\ \mathtt{err}_{\mathcal{A}}$. Applying (n-q-forall-err-prop) to (2), we have (3) $\blacktriangleright(\forall(\alpha.\hat{\tau}))\ \mathtt{err}_{\mathcal{A}}$. We can conclude by (III) on (3).

Case (k-qity-parr). We have that $\sigma = \blacktriangleright(\hat{\tau}_1 \rightharpoonup \hat{\tau}_2)$ and (1) $\emptyset\ \emptyset \vdash^{n}_{\Phi} \blacktriangleright(\hat{\tau}_1) :: \mathsf{ITy}$ and (2) $\emptyset\ \emptyset \vdash^{n}_{\Phi} \blacktriangleright(\hat{\tau}_2) :: \mathsf{ITy}$. Applying the IH to (1), there are three possibilities:

Case (3) $\blacktriangleright(\hat{\tau}_1) \mapsto_{\mathcal{A}} \sigma'_1$. By Lemma 15 on (3), $\sigma'_1 = \blacktriangleright(\hat{\tau}'_1)$. Applying (n-q-parr-step-1) to (3), we have (4) $\blacktriangleright(\hat{\tau}_1 \rightharpoonup \hat{\tau}_2) \mapsto_{\mathcal{A}} \blacktriangleright(\hat{\tau}'_1 \rightharpoonup \hat{\tau}_2)$. We can conclude by (I) on (4).

Case (3) $\blacktriangleright(\hat{\tau}_1)\ \mathtt{val}_{\mathcal{A}}$. Applying the IH to (2), there are three possibilities:

Case (4) $\blacktriangleright(\hat{\tau}_2) \mapsto_{\mathcal{A}} \sigma'_2$. By Lemma 15 on (4), $\sigma'_2 = \blacktriangleright(\hat{\tau}'_2)$. By (n-q-parr-step-2) on (3) and (4), we have (5) $\blacktriangleright(\hat{\tau}_1 \rightharpoonup \hat{\tau}_2) \mapsto_{\mathcal{A}} \blacktriangleright(\hat{\tau}_1 \rightharpoonup \hat{\tau}'_2)$. We can conclude by (I) on (5).

Case (4) $\blacktriangleright(\hat{\tau}_2)\ \mathtt{val}_{\mathcal{A}}$. Applying (n-q-parr-val) to (3) and (4), we have (5) $\blacktriangleright(\hat{\tau}_1 \rightharpoonup \hat{\tau}_2)\ \mathtt{val}_{\mathcal{A}}$. We can conclude by (II) on (5).

Case (4) $\blacktriangleright(\hat{\tau}_2)\ \mathtt{err}_{\mathcal{A}}$. Applying (n-q-parr-err-prop-2) to (4), we have (5) $\blacktriangleright(\hat{\tau}_1 \rightharpoonup \hat{\tau}_2)\ \mathtt{err}_{\mathcal{A}}$. We can conclude by (III) on (5).

Case (3) $\blacktriangleright(\hat{\tau}_1)\ \mathtt{err}_{\mathcal{A}}$. Applying (n-q-parr-err-prop-1) to (3), we have (4) $\blacktriangleright(\hat{\tau}_1 \rightharpoonup \hat{\tau}_2)\ \mathtt{err}_{\mathcal{A}}$. We can conclude by (III) on (4).

Case (k-qitm-unquote). We have $\sigma = \triangleright(\triangleleft(\sigma'))$ and (1) $\emptyset\,\emptyset \vdash^n_\Phi \sigma' :: \mathsf{ITm}$. Applying the IH to (1) and (b), we have three cases:

Case (2) $\sigma' \mapsto_\mathcal{A} \sigma''$. Applying (n-q-unquote-step) to (2), we have (3) $\triangleright(\triangleleft(\sigma')) \mapsto_\mathcal{A} \triangleright(\triangleleft(\sigma''))$. We can conclude by (I) on (3).

Case (2) $\sigma'\ \texttt{val}_\mathcal{A}$. Applying Lemma 17.9 to (1) and (2), we have that (3) $\sigma' = \triangleright(\hat{\iota})$. Applying (n-q-unquote-elim) to (3), we have (4) $\triangleright(\triangleleft(\triangleright(\hat{\iota}))) \mapsto_\mathcal{A} \triangleright(\hat{\iota})$. We can conclude by (I) on (4).

Case (2) $\sigma'\ \texttt{err}_\mathcal{A}$. Applying (n-q-unquote-err-prop) on (2), we have (3) $\triangleright(\triangleleft(\sigma'))\ \texttt{err}_\mathcal{A}$. We can conclude by (III) on (3).

Case (k-qitm-anatrans). We have $\sigma = \triangleright(\mathsf{anatrans}[n'](\sigma'))$ and (1) $n' < n$ and (2) $\emptyset\,\emptyset \vdash^n_\Phi \sigma' :: \mathsf{Ty}$. Applying the IH to (2), we have three cases:

Case (3) $\sigma' \mapsto_\mathcal{A} \sigma''$. Applying (n-q-anatrans-step) to (3), we have (4) $\triangleright(\mathsf{anatrans}[n'](\sigma')) \mapsto_\mathcal{A} \triangleright(\mathsf{anatrans}[n'](\sigma''))$. We can conclude by (I) on (4).

Case (3) $\sigma'\ \texttt{val}_\mathcal{A}$. Applying (n-q-anatrans-val) to (2), (b) and (1), we have (4) $\triangleright(\mathsf{anatrans}[n'](\sigma'))\ \texttt{val}_\mathcal{A}$. We can conclude by (II) on (4).

Case (3) $\sigma'\ \texttt{err}_\mathcal{A}$. Applying (n-q-anatrans-err-prop) to (3), we have (4) $\triangleright(\mathsf{anatrans}[n'](\sigma'))\ \texttt{err}_\mathcal{A}$. We can conclude by (III) on (4).

Case (k-qitm-syntrans). We have $\sigma = \triangleright(\mathsf{syntrans}[n'])$ and (1) $n' < n$. Applying (n-q-syntrans-val) to (b) and (1), we have (2) $\triangleright(\mathsf{syntrans}[n'])\ \texttt{val}_\mathcal{A}$. We can conclude by (II) on (2).

Cases [All rules in Figure 21]. We give an example base case and inductive cases with 0, 1, 2 and 3 sub-terms. All rules with the same number of sub-terms follow the corresponding script, replacing only the forms and rule names:

Case [k-qitm-var]. In this case, $\sigma = \triangleright(x)$. Applying (n-q-x-val), we have (1) $\triangleright(x)\ \texttt{val}_\mathcal{A}$. We can conclude by (II) on (1).

Case (k-qitm-tabs). We have that $\sigma = \triangleright(\Lambda(\alpha.\hat{\iota}))$ and (1) $\emptyset\ \emptyset \vdash^n_\Phi \triangleright(\hat{\iota}) :: \mathsf{ITm}$. Applying the IH to (1) and (b), we have three cases:

Case (2) $\triangleright(\hat{\iota}) \mapsto_\mathcal{A} \sigma'$. By Lemma 16 on (2), we have $\sigma' = \triangleright(\hat{\iota}')$. Applying (n-q-tabs-step) to (2), we have (3) $\triangleright(\Lambda(\alpha.\hat{\iota})) \mapsto_\mathcal{A} \triangleright(\Lambda(\alpha.\hat{\iota}'))$. We can conclude by (I) on (3).

Case (2) $\triangleright(\hat{\iota})\ \texttt{val}_\mathcal{A}$. Applying (n-q-tabs-val) to (2), we have (3) $\triangleright(\Lambda(\alpha.\hat{\iota}))\ \texttt{val}_\mathcal{A}$. We can conclude by (II) on (3).

Case (2) $\triangleright(\hat{\iota})\ \texttt{err}_\mathcal{A}$. Applying (n-q-tabs-err-prop) to (2), we have (3) $\triangleright(\Lambda(\alpha.\hat{\iota}))\ \texttt{err}_\mathcal{A}$. We can conclude by (III) on (3).

Case [k-qitm-abs]. In this case, $\sigma = \triangleright(\lambda x{:}\hat{\tau}.\hat{\iota})$ and (1) $\emptyset\ \emptyset \vdash^n_\Phi \blacktriangleright(\hat{\tau}) :: \mathsf{ITy}$ and (2) $\emptyset\,\emptyset \vdash^n_\Phi \triangleright(\hat{\iota}) :: \mathsf{ITm}$. Applying the IH to (1) and (b), we have three cases:

Case (3) $\blacktriangleright(\hat{\tau}) \mapsto_{\mathcal{A}} \sigma_1'$. By Lemma 15 on (3), $\sigma_1' = \blacktriangleright(\hat{\tau}')$. Applying (n-q-abs-step-1) on (3), we have (4) $\triangleright(\lambda x{:}\hat{\tau}.\hat{\iota}) \mapsto_{\mathcal{A}} \triangleright(\lambda x{:}\hat{\tau}'.\hat{\iota})$. We can conclude by (I) on (4).

Case (3) $\blacktriangleright(\hat{\tau})$ $\texttt{val}_{\mathcal{A}}$. Applying the IH (2) and (b), we have three cases:

Case (4) $\triangleright(\hat{\iota}) \mapsto_{\mathcal{A}} \sigma_2'$. By Lemma 16 on (4), we have $\sigma_2' = \triangleright(\hat{\iota}')$. Applying (n-q-abs-step-2) to (3) and (4), we have (5) $\triangleright(\lambda x{:}\hat{\tau}.\hat{\iota}) \mapsto_{\mathcal{A}} \triangleright(\lambda x{:}\hat{\tau}.\hat{\iota}')$. We can conclude by (I) on (5).

Case (4) $\triangleright(\hat{\iota})$ $\texttt{val}_{\mathcal{A}}$. Applying (n-q-abs-val) to (3) and (4), we have (5) $\triangleright(\lambda x{:}\hat{\tau}.\hat{\iota})$ $\texttt{val}_{\mathcal{A}}$. We can conclude by (II) on (5).

Case (4) $\triangleright(\hat{\iota})$ $\texttt{err}_{\mathcal{A}}$. Applying (n-q-abs-err-prop-2) to (4), we have (5) $\triangleright(\lambda x{:}\hat{\tau}.\hat{\iota})$ $\texttt{err}_{\mathcal{A}}$. We can conclude by (III) on (5).

Case (3) $\blacktriangleright(\hat{\tau})$ $\texttt{err}_{\mathcal{A}}$. Applying (n-q-abs-err-prop-2) to (3), we have (4) $\triangleright(\lambda x{:}\hat{\tau}.\hat{\iota})$ $\texttt{err}_{\mathcal{A}}$. We can conclude by (III) on (4).

Case (k-qitm-case). In this case $\sigma = \triangleright(\mathsf{case}(\hat{\iota}; x.\hat{\iota}_1; x.\hat{\iota}_2))$ and (1) $\emptyset\;\emptyset \vdash^n_\Phi \triangleright(\hat{\iota}) :: \mathsf{ITm}$ and (2) $\emptyset\;\emptyset \vdash^n_\Phi \triangleright(\hat{\iota}_1) :: \mathsf{ITm}$ and (3) $\emptyset\;\emptyset \vdash^n_\Phi \triangleright(\hat{\iota}_2) :: \mathsf{ITm}$. Applying the IH to (1) and (b), we have three cases:

Case (4) $\triangleright(\hat{\iota}) \mapsto_{\mathcal{A}} \sigma'$. By Lemma 16 on (4), $\sigma' = \triangleright(\hat{\iota}')$. Applying (n-q-case-step-1) to (4), we have (5) $\triangleright(\mathsf{case}(\hat{\iota}; x.\hat{\iota}_1; x.\hat{\iota}_2)) \mapsto_{\mathcal{A}} \triangleright(\mathsf{case}(\hat{\iota}'; x.\hat{\iota}_1; x.\hat{\iota}_2))$. We can conclude by (I) on (5).

Case (4) $\triangleright(\hat{\iota})$ $\texttt{val}_{\mathcal{A}}$. Applying the IH to (2) and (b), we have three cases:

Case (5) $\triangleright(\hat{\iota}_1) \mapsto_{\mathcal{A}} \sigma_1'$. By Lemma 16 on (5), $\sigma_1' = \triangleright(\hat{\iota}_1')$. Applying (n-q-case-step-2) to (4) and (5), we have (6) $\triangleright(\mathsf{case}(\hat{\iota}; x.\hat{\iota}_1; x.\hat{\iota}_2)) \mapsto_{\mathcal{A}} \triangleright(\mathsf{case}(\hat{\iota}; x.\hat{\iota}_1'; x.\hat{\iota}_2))$. We can conclude by (I) on (6).

Case (5) $\triangleright(\hat{\iota}_1)$ $\texttt{val}_{\mathcal{A}}$. Applying the IH to (3) and (b), we have three cases:

Case (6) $\triangleright(\hat{\iota}_2) \mapsto_{\mathcal{A}} \sigma_2'$. By Lemma 16 on (6), $\sigma_2' = \triangleright(\hat{\iota}_2')$. Applying (n-q-case-step-3) to (4), (5) and (6), we have (7) $\triangleright(\mathsf{case}(\hat{\iota}; x.\hat{\iota}_1; x.\hat{\iota}_2)) \mapsto_{\mathcal{A}} \triangleright(\mathsf{case}(\hat{\iota}; x.\hat{\iota}_1; x.\hat{\iota}_2'))$. We can conclude by (I) on (7).

Case (6) $\triangleright(\hat{\iota}_2)$ $\texttt{val}_{\mathcal{A}}$. Applying (n-q-case-val) to (4), (5) and (6), we have (7) $\triangleright(\mathsf{case}(\hat{\iota}; x.\hat{\iota}_1; x.\hat{\iota}_2))$ $\texttt{val}_{\mathcal{A}}$. We can conclude by (II) on (7).

Case (6) $\triangleright(\hat{\iota}_2)$ $\texttt{err}_{\mathcal{A}}$. Applyig (n-q-case-err-prop-3) to (6), we have (7) $\triangleright(\mathsf{case}(\hat{\iota}; x.\hat{\iota}_1; x.\hat{\iota}_2))$ $\texttt{err}_{\mathcal{A}}$. We can conclude by (III) on (7).

Case (5) $\triangleright(\hat{\iota}_1)$ $\texttt{err}_{\mathcal{A}}$. Applying (n-q-case-err-prop-2) to (5), we have (6) $\triangleright(\mathsf{case}(\hat{\iota}; x.\hat{\iota}_1; x.\hat{\iota}_2))$ $\texttt{err}_{\mathcal{A}}$. We can conclude by (III) on (6).

Case (4) $\rhd(\hat{\iota})\ \texttt{err}_{\mathcal{A}}$. Applying (n-q-case-err-prop-1) to (4), we have (5) $\rhd(\mathsf{case}(\hat{\iota}; x.\hat{\iota}_1; x.\hat{\iota}_2))\ \texttt{err}_{\mathcal{A}}$. We can conclude by (III) on (5).

Case (k-ana). We have $\sigma = \mathsf{ana}[n'](\sigma')$ and (1) $n' < n$ and (2) $\emptyset\ \emptyset \vdash^n_\Phi \sigma' :: \mathsf{Ty}$. Applying the IH to (2) and (b), we have three cases:

Case (3) $\sigma' \mapsto_{\mathcal{A}} \sigma''$. Applying (n-ana-step) to (3), we have (4) $\mathsf{ana}[n'](\sigma') \mapsto_{\mathcal{A}} \mathsf{ana}[n'](\sigma'')$. We can conclude by (I) on (4).

Case (3) $\sigma'\ \texttt{val}_{\mathcal{A}}$. By (1) and (b), we have (4) $\mathsf{nth}[n'](\overline{e}) = e$.

Because our semantics are non-deterministic (the bracketed premise of (n-ana-fail) is assumed to be only advisory to implementors), by applying (n-ana-fail) to (3) and (4), we have (5) $\mathsf{ana}[n'](\sigma')\ \texttt{err}_{\mathcal{A}}$. We can conclude by (III) on (5).

Case (3) $\sigma'\ \texttt{err}_{\mathcal{A}}$. Applying (n-ana-err-prop) to (3), we have (4) $\mathsf{ana}[n'](\sigma')\ \texttt{err}_{\mathcal{A}}$. We can conclude by (III) on (4).

Case (k-syn). We have $\sigma = \mathsf{syn}[n']$ and (1) $n' < n$. By (1) and (b), we have (2) $\mathsf{nth}[n](\overline{e}) = e$. Because our semantics are non-deterministic (the bracketed premise of (n-syn-fail) is assumed to be only advisory to implementors), by applying (n-syn-fail) to (2), we have (3) $\mathsf{syn}[n']\ \texttt{err}_{\mathcal{A}}$. We can conclude by (III) on (3). □

Theorem 2 (Static Preservation)**.** *Letting $\mathcal{A} = \overline{e}; \Upsilon; \Phi$, if (a) $\sigma \mapsto_{\overline{e};\Upsilon;\Phi} \sigma'$ and (b) $\emptyset\ \emptyset \vdash^n_\Phi \sigma :: \kappa$ and (c) $\vdash \Phi$ and (d) $\Upsilon\ \texttt{ctx}_\Phi$ then $\emptyset\ \emptyset \vdash^n_\Phi \sigma' :: \kappa$.*

Proof. Rule induction on (a).

Case [All rules in Figure 23]. The cases for the core language follow the standard script and are omitted for concision.

Case (n-ty-step). In this case, we have $\sigma = c\langle\sigma_{\text{tyidx}}\rangle$ and $\sigma' = c\langle\sigma'_{\text{tyidx}}\rangle$ and $\kappa = \mathsf{Ty}$ and (1) $\sigma_{\text{tyidx}} \mapsto_{\mathcal{A}} \sigma'_{\text{tyidx}}$. Rule induction on (b). Only three rules apply:

Case (k-ty-parr). In this case, we have $c = \rightharpoonup$ and (2) $\emptyset\ \emptyset \vdash^n_\Phi \sigma_{\text{tyidx}} :: \mathsf{Ty} \times \mathsf{Ty}$. Applying the IH to (1), (2), (c) and (d), we have (3) $\emptyset\ \emptyset \vdash^n_\Phi \sigma'_{\text{tyidx}} :: \mathsf{Ty} \times \mathsf{Ty}$. Apply (k-ty-parr) to (3).

Case (k-ty-ext). In this case, we have $c = \textsc{tc}$ and (2) $\mathsf{tycon}\ \textsc{tc}\ \{\theta\} \sim \mathsf{tcsig}[\kappa_{\text{tyidx}}]\ \{\chi\} \in \Phi$ and (3) $\emptyset\ \emptyset \vdash^n_\Phi \sigma_{\text{tyidx}} :: \kappa_{\text{tyidx}}$. Applying the IH to (1), (3), (c) and (d), we have (4) $\emptyset\ \emptyset \vdash^n_\Phi \sigma'_{\text{tyidx}} :: \kappa_{\text{tyidx}}$. Apply (k-ty-ext) to (2) and (4).

Case (k-ty-other). In this case, we have $c = \mathsf{other}[m; \kappa']$ and (2) $\vdash \kappa'\ \mathsf{eq}$ and (3) $\emptyset\ \emptyset \vdash^n_\Phi \sigma_{\text{tyidx}} :: \kappa' \times \mathsf{ITy}$. Applying the IH to (1) and (3), we have (4) $\emptyset\ \emptyset \vdash^n_\Phi \sigma'_{\text{tyidx}} :: \kappa' \times \mathsf{ITy}$. Apply (k-ty-other) to (2) and (4).

Case (n-tycase-step). In this case, we have $\sigma = \mathsf{tycase}[c](\sigma''; \boldsymbol{x}.\sigma_1; \boldsymbol{x})\sigma_2$ and $\sigma' = \mathsf{tycase}[c](\sigma'''; \boldsymbol{x}.\sigma_1; \boldsymbol{x})\sigma_2$ and (1) $\sigma'' \mapsto_{\mathcal{A}} \sigma'''$. Rule induction on (b). Only two rules apply:

Case (k-tycase-parr). In this case, we have $c = \rightharpoonup$ and (2) $\emptyset\ \emptyset \vdash^n_\Phi \sigma'' :: \mathsf{Ty}$ and (3) $\emptyset\ \emptyset, \boldsymbol{x} :: \mathsf{Ty} \times \mathsf{Ty} \vdash^n_\Phi \sigma_1 :: \kappa$ and (4) $\emptyset\ \emptyset \vdash^n_\Phi \sigma_2 :: \kappa$. Applying the IH to (1) and (2), we have that (5) $\emptyset\ \emptyset \vdash^n_\Phi \sigma''' :: \mathsf{Ty}$. Apply (k-tycase-parr) to (5), (3) and (4).

Case (k-tycase-ext). In this case, have $c = \textsc{tc}$ and (2) $\emptyset\ \emptyset \vdash^n_\Phi \sigma'' :: \mathsf{Ty}$ and (3) $\mathsf{tycon}\ \textsc{tc}\ \{\theta\} \sim \mathsf{tcsig}[\kappa_{\text{tyidx}}]\ \{\chi\} \in \Phi$ and (4) $\emptyset\ \emptyset, \boldsymbol{x} :: \kappa_{\text{tyidx}} \vdash^n_\Phi \sigma_1 :: \kappa$ and (5) $\emptyset\ \emptyset \vdash^n_\Phi \sigma_2 :: \kappa$. Applying the IH to (1) and (2), we have that (6) $\emptyset\ \emptyset \vdash^n_\Phi \sigma''' :: \mathsf{Ty}$. Apply (k-tycase-ext) to (6), (3), (4) and (5).

Case (n-tycase-elim-1). In this case, we have $\sigma = \mathsf{tycase}[c](c\langle\sigma_{\text{tyidx}}\rangle; \boldsymbol{x}.\sigma_1; \sigma_2)$ and $\sigma' = [\sigma_{\text{tyidx}}/\boldsymbol{x}]\sigma_1$ and (1) $c\langle\sigma_{\text{tyidx}}\rangle\ \mathtt{val}_{\mathcal{A}}$. Rule induction on (b). Only two rules apply:

Case (k-tycase-parr). In this case, we have $c = \rightharpoonup$ and (2) $\emptyset\ \emptyset \vdash^n_\Phi \rightharpoonup\langle\sigma_{\text{tyidx}}\rangle :: \mathsf{Ty}$ and (3) $\emptyset\ \emptyset, \boldsymbol{x} :: \mathsf{Ty} \times \mathsf{Ty} \vdash^n_\Phi \sigma_1 :: \kappa$. Applying Lemma 17.7 to (1) and (2), we have that (4) $\sigma_{\text{tyidx}} = (\sigma_1, \sigma_2)$ and (5) $\emptyset\ \emptyset \vdash^n_\Phi \sigma_{\text{tyidx}} :: \mathsf{Ty} \times \mathsf{Ty}$. Apply Assumption 10 to (3) and (5).

Case (k-tycase-ext). In this case, we have $c = \textsc{tc}$ and (2) $\emptyset\ \emptyset \vdash^n_\Phi \textsc{tc}\langle\sigma_{\text{tyidx}}\rangle :: \mathsf{Ty}$ and (3) $\mathsf{tycon}\ \textsc{tc}\ \{\theta\} \sim \mathsf{tcsig}[\kappa_{\text{tyidx}}]\ \{\chi\}$ and (4) $\emptyset\ \emptyset, \boldsymbol{x} :: \kappa_{\text{tyidx}} \vdash^n_{\sigma_1} \kappa ::$. Applying Lemma 18.7 to (1) and (2), we have that (5) $\emptyset\ \emptyset \vdash^n_\Phi \sigma_{\text{tyidx}} :: \kappa_{\text{tyidx}}$. Apply Assumption 10 to (4) and (5).

Case (n-tycase-elim-2). In this case, we have $\sigma = \mathsf{tycase}[c](c'\langle\sigma_{\text{tyidx}}\rangle; \boldsymbol{x}.\sigma_1; \sigma_2)$ and $\sigma' = \sigma_2$. Rule induction on (b). Only two rules apply:

Case (k-tycase-parr). In this case, we have (1) $\emptyset\ \emptyset \vdash^n_\Phi \sigma_2 :: \kappa$. We can conclude by (1).

Case (k-tycase-ext). In this case, we have (1) $\emptyset\ \emptyset \vdash^n_\Phi \sigma_2 :: \kappa$. We can conclude by (1).

Case (n-q-t-unquote-step). In this case, we have $\sigma = \blacktriangleright(\blacktriangleleft(\sigma''))$ and $\sigma' = \blacktriangleright(\blacktriangleleft(\sigma'''))$ and $\kappa = \mathsf{ITy}$ and (1) $\sigma'' \mapsto_{\mathcal{A}} \sigma'''$. Rule induction on (b). Only rule (k-qity-unquote) applies, so (2) $\emptyset\ \emptyset \vdash^n_\Phi \sigma'' :: \mathsf{ITy}$. Applying the IH to (1) and (2), we have (3) $\emptyset\ \emptyset \vdash^n_\Phi \sigma''' :: \mathsf{ITy}$. Apply (k-qity-unquote) to (3).

Case (n-q-t-unquote-elim). In this case, we have $\sigma = \blacktriangleright(\blacktriangleleft(\blacktriangleright(\hat{\tau})))$ and $\sigma' = \blacktriangleright(\hat{\tau})$ and $\kappa = \mathsf{ITy}$. Rule induction on (b). Only rule (k-qity-unquote) applies, so (1) $\emptyset\ \emptyset \vdash^n_\Phi \blacktriangleright(\hat{\tau}) :: \mathsf{ITy}$. We can conclude by (1).

Case (n-q-tytrans-step). In this case, we have $\sigma = \blacktriangleright(\mathsf{trans}(\sigma''))$ and $\sigma' = \blacktriangleright(\mathsf{trans}(\sigma'''))$ and $\kappa = \mathsf{ITy}$ and (1) $\sigma'' \mapsto_{\mathcal{A}} \sigma'''$. Rule induction on (b). Only rule (k-qity-trans) applies, so (2) $\emptyset\ \emptyset \vdash^n_\Phi \sigma'' :: \mathsf{ITy}$. Applying the IH to (1) and (2), we have (3) $\emptyset\ \emptyset \vdash^n_\Phi \sigma''' :: \mathsf{ITy}$. Apply (k-qity-trans) to (3).

Cases [All rules in Figure 26]. We give an example for forms with 1 and 2 sub-terms. All rules with the same number of sub-terms follow the corresponding script, replacing only the forms and rule names.

Case (n-q-forall-step). In this case, we have $\sigma = \blacktriangleright(\forall(\alpha.\hat{\tau}))$ and $\sigma' = \blacktriangleright(\forall(\alpha.\hat{\tau}'))$ and (1) $\blacktriangleright(\hat{\tau}) \mapsto_{\mathcal{A}} \blacktriangleright(\hat{\tau}')$. Rule induction on (b). Only rule (k-qity-forall) applies, so (2)

$\emptyset\ \emptyset \vdash^n_\Phi \blacktriangleright(\hat{\tau}) :: \mathsf{ITy}$. Applying the IH to (1) and (2), we have (3) $\emptyset\ \emptyset \vdash^n_\Phi \blacktriangleright(\hat{\tau}') :: \mathsf{ITy}$. Apply (k-qity-forall) to (3).

Case (n-q-parr-step-1). In this case, we have $\sigma = \blacktriangleright(\hat{\tau}_1 \rightharpoonup \hat{\tau}_2)$ and $\sigma' = \blacktriangleright(\hat{\tau}_1' \rightharpoonup \hat{\tau}_2)$ and (1) $\blacktriangleright(\hat{\tau}_1) \mapsto_{\mathcal{A}} \blacktriangleright(\hat{\tau}_1')$. Rule induction on (b). Only rule (k-qity-parr) applies, so (2) $\emptyset\ \emptyset \vdash^n_\Phi \blacktriangleright(\hat{\tau}_1) :: \mathsf{ITy}$ and (3) $\emptyset\ \emptyset \vdash^n_\Phi \blacktriangleright(\hat{\tau}_2) :: \mathsf{ITy}$. Applying the IH to (1) and (2), we have (4) $\emptyset\ \emptyset \vdash^n_\Phi \blacktriangleright(\hat{\tau}_1') :: \mathsf{ITy}$. Apply (k-qity-parr) to (4) and (3).

Case (n-q-parr-step-2). In this case, we have $\sigma = \blacktriangleright(\hat{\tau}_1 \rightharpoonup \hat{\tau}_2)$ and $\sigma' = \blacktriangleright(\hat{\tau}_1 \rightharpoonup \hat{\tau}_2')$ and (1) $\blacktriangleright(\hat{\tau}_2) \mapsto_{\mathcal{A}} \blacktriangleright(\hat{\tau}_2')$. Rule induction on (b). Only rule (k-qity-parr) applies, so (2) $\emptyset\ \emptyset \vdash^n_\Phi \blacktriangleright(\hat{\tau}_1) :: \mathsf{ITy}$ and (3) $\emptyset\ \emptyset \vdash^n_\Phi \blacktriangleright(\hat{\tau}_2) :: \mathsf{ITy}$. Applying the IH to (1) and (3), we have (4) $\emptyset\ \emptyset \vdash^n_\Phi \blacktriangleright(\hat{\tau}_2') :: \mathsf{ITy}$. Apply (k-qity-parr) to (2) and (4).

Case (n-q-unquote-step). In this case, we have $\sigma = \triangleright(\triangleleft(\sigma''))$ and $\sigma' = \triangleright(\triangleleft(\sigma'''))$ and $\kappa = \mathsf{ITm}$ and (1) $\sigma'' \mapsto_{\mathcal{A}} \sigma'''$. Rule induction on (b). Only rule (k-qitm-unquote) applies, so (2) $\emptyset\ \emptyset \vdash^n_\Phi \sigma'' :: \mathsf{ITm}$. Applying the IH to (1) and (2), we have (3) $\emptyset\ \emptyset \vdash^n_\Phi \sigma''' :: \mathsf{ITm}$. Apply (k-qitm-unquote) to (3).

Case (n-q-unquote-elim). In this case, we have $\sigma = \triangleright(\triangleleft(\triangleright(\hat{\iota})))$ and $\sigma' = \triangleright(\hat{\iota})$ and $\kappa = \mathsf{ITm}$. Rule induction on (b). Only rule (k-qitm-unquote) applies, so (1) $\emptyset\ \emptyset \vdash^n_\Phi \triangleright(\hat{\iota}) :: \mathsf{ITm}$. We can conclude by (1).

Case (n-q-anatrans-step). In this case, we have $\sigma = \triangleright(\mathsf{anatrans}[n'](\sigma''))$ and $\sigma' = \triangleright(\mathsf{anatrans}[n'](\sigma'''))$ and $\kappa = \mathsf{ITm}$ and (1) $\sigma'' \mapsto_{\mathcal{A}} \sigma'''$. Rule induction on (b). Only rule (k-qitm-anatrans) applies, so (2) $\emptyset\ \emptyset \vdash^n_\Phi \sigma'' :: \mathsf{ITm}$ and (3) $n' < n$. Applying the IH to (1) and (2), we have (4) $\emptyset\ \emptyset \vdash^n_\Phi \sigma''' :: \mathsf{ITm}$. Apply (k-qitm-trans) to (3) and (4).

Cases [All rules in Figures 28 and 29]. We give an example for forms with 1, 2 and 3 subterms. All rules with the same number of sub-terms follow the corresponding script, replacing only the forms and rule names.

Case (n-q-tabs-step). In this case, we have $\sigma = \triangleright(\Lambda(\alpha.\hat{\iota}))$ and $\sigma' = \triangleright(\Lambda(\alpha.\hat{\iota}'))$ and (1) $\blacktriangleright(\hat{\iota}) \mapsto_{\mathcal{A}} \blacktriangleright(\hat{\iota}')$. Rule induction on (b). Only rule (k-qitm-tabs) applies, so (2) $\emptyset\ \emptyset \vdash^n_\Phi \triangleright(\hat{\iota}) :: \mathsf{ITm}$. Applying the IH to (1) and (2), we have (3) $\emptyset\ \emptyset \vdash^n_\Phi \triangleright(\hat{\iota}') :: \mathsf{ITm}$. Apply (k-qitm-tabs) to (3).

Case (n-q-abs-step-1). In this case, we have $\sigma = \triangleright(\lambda x{:}\hat{\tau}_1.\hat{\iota}_2)$ and $\sigma' = \triangleright(\lambda x{:}\hat{\tau}_1'.\hat{\iota}_2)$ and (1) $\blacktriangleright(\hat{\tau}_1) \mapsto_{\mathcal{A}} \blacktriangleright(\hat{\tau}_1')$. Rule induction on (b). Only rule (k-qitm-abs) applies, so (2) $\emptyset\ \emptyset \vdash^n_\Phi \blacktriangleright(\hat{\tau}_1) :: \mathsf{ITy}$ and (3) $\emptyset\ \emptyset \vdash^n_\Phi \triangleright(\hat{\iota}_2) :: \mathsf{ITm}$. Applying the IH to (1) and (2), we have (4) $\emptyset\ \emptyset \vdash^n_\Phi \blacktriangleright(\hat{\tau}_1') :: \mathsf{ITy}$. Apply (k-qitm-abs) to (4) and (3).

Case (n-q-abs-step-2). In this case, we have $\sigma = \triangleright(\lambda x{:}\hat{\tau}_1.\hat{\iota}_2)$ and $\sigma' = \triangleright(\lambda x{:}\hat{\tau}_1.\hat{\iota}_2')$ and (1) $\triangleright(\hat{\iota}_2) \mapsto_{\mathcal{A}} \triangleright(\hat{\iota}_2')$. Rule induction on (b). Only rule (k-qitm-abs) applies, so (2) $\emptyset\ \emptyset \vdash^n_\Phi \blacktriangleright(\hat{\tau}_1) :: \mathsf{ITy}$ and (3) $\emptyset\ \emptyset \vdash^n_\Phi \triangleright(\hat{\iota}_2) :: \mathsf{ITm}$. Applying the IH to (1) and (3), we have (4) $\emptyset\ \emptyset \vdash^n_\Phi \triangleright(\hat{\iota}_2') :: \mathsf{ITm}$. Apply (k-qitm-abs) to (2) and (4).

Case (n-q-case-step-1). In this case, we have $\sigma = \triangleright(\mathsf{case}(\hat{\iota}; x.\hat{\iota}_1; x.\hat{\iota}_2))$ and $\sigma' = \triangleright(\mathsf{case}(\hat{\iota}'; x.\hat{\iota}_1; x.\hat{\iota}_2))$ and (1) $\triangleright(\hat{\iota}) \mapsto_{\mathcal{A}} \triangleright(\hat{\iota}')$. Rule induction on (b). Only rule (k-qitm-case) applies, so (2) $\emptyset\ \emptyset \vdash^n_\Phi \triangleright(\hat{\iota}) :: \mathsf{ITm}$ and (3) $\emptyset\ \emptyset \vdash^n_\Phi \triangleright(\hat{\iota}_1) :: \mathsf{ITm}$ and (4)

$\emptyset\;\emptyset \vdash^n_\Phi \rhd(\hat{\iota}_2) :: \mathsf{ITm}$. Applying the IH to (1) and (2), we have (5) $\emptyset\;\emptyset \vdash^n_\Phi \rhd(\hat{\iota}') :: \mathsf{ITm}$. Apply (k-qitm-case) to (5), (3) and (4).

Case (n-q-case-step-2). In this case, we have $\sigma = \rhd(\mathsf{case}(\hat{\iota}; x.\hat{\iota}_1; x.\hat{\iota}_2))$ and $\sigma' = \rhd(\mathsf{case}(\hat{\iota}; x.\hat{\iota}_1'; x.\hat{\iota}_2))$ and (1) $\rhd(\hat{\iota}_1) \mapsto_{\mathcal{A}} \rhd(\hat{\iota}_1')$. Rule induction on (b). Only rule (k-qitm-case) applies, so (2) $\emptyset\;\emptyset \vdash^n_\Phi \rhd(\hat{\iota}) :: \mathsf{ITm}$ and (3) $\emptyset\;\emptyset \vdash^n_\Phi \rhd(\hat{\iota}_1) :: \mathsf{ITm}$ and (4) $\emptyset\;\emptyset \vdash^n_\Phi \rhd(\hat{\iota}_2) :: \mathsf{ITm}$. Applying the IH to (1) and (3), we have (5) $\emptyset\;\emptyset \vdash^n_\Phi \rhd(\hat{\iota}') :: \mathsf{ITm}$. Apply (k-qitm-case) to (2), (5) and (4).

Case (n-q-case-step-3). In this case, we have $\sigma = \rhd(\mathsf{case}(\hat{\iota}; x.\hat{\iota}_1; x.\hat{\iota}_2))$ and $\sigma' = \rhd(\mathsf{case}(\hat{\iota}; x.\hat{\iota}_1; x.\hat{\iota}_2'))$ and (1) $\rhd(\hat{\iota}_2) \mapsto_{\mathcal{A}} \rhd(\hat{\iota}_2')$. Rule induction on (b). Only rule (k-qitm-case) applies, so (2) $\emptyset\;\emptyset \vdash^n_\Phi \rhd(\hat{\iota}) :: \mathsf{ITm}$ and (3) $\emptyset\;\emptyset \vdash^n_\Phi \rhd(\hat{\iota}_1) :: \mathsf{ITm}$ and (4) $\emptyset\;\emptyset \vdash^n_\Phi \rhd(\hat{\iota}_2) :: \mathsf{ITm}$. Applying the IH to (1) and (4), we have (5) $\emptyset\;\emptyset \vdash^n_\Phi \rhd(\hat{\iota}_2') :: \mathsf{ITm}$. Apply (k-qitm-case) to (2), (3) and (5).

Case (n-ana-step). In this case, we have $\sigma = \mathsf{ana}[n'](\sigma'')$ and $\sigma' = \mathsf{ana}[n'](\sigma''')$ and (1) $\sigma'' \mapsto_{\mathcal{A}} \sigma'''$. Rule induction on (b). Only rule (k-ana) applies, so (2) $n' < n$ and (3) $\emptyset\;\emptyset \vdash^n_\Phi \sigma'' :: \mathsf{Ty}$. Applying the IH to (1) and (3), we have (4) $\emptyset\;\emptyset \vdash^n_\Phi \sigma''' :: \mathsf{Ty}$. Apply (k-ana) to (2) and (4).

Case (n-ana-success). In this case, we have $\sigma = \mathsf{ana}[n'](\sigma'')$ and $\sigma' = \rhd(\mathsf{anatrans}[n'](\sigma''))$. Rule induction on (b). Only rule (k-ana) applies, so (2) $n' < n$ and (3) $\emptyset\;\emptyset \vdash^n_\Phi \sigma'' :: \mathsf{Ty}$. Apply (k-qitm-anatrans) to (2) and (3).

Case (n-syn-success). In this case, we have $\sigma = \mathsf{syn}[n']$ and (1) $\mathsf{nth}[n'](\overline{e}) = e$ and (2) $\Upsilon \vdash_\Phi e \Leftarrow \sigma'' \rightsquigarrow \iota$ and $\sigma' = (\sigma'', \rhd(\mathsf{syntrans}[n']))$. Rule induction on (b). Only rule (k-syn) applies, so (3) $n' < n$. Applying Theorem 3 to (c), (d), (2), we have that (4) $\emptyset\;\emptyset \vdash^n_\Phi \sigma'' :: \mathsf{Ty}$. Applying (k-qitm-syntrans) to (3), we have (5) $\emptyset\;\emptyset \vdash^n_\Phi \rhd(\mathsf{syntrans}[n']) :: \mathsf{ITm}$. Applying (k-prod) to (4) and (5), we have that $\emptyset\;\emptyset \vdash^n_\Phi (\sigma'', \rhd(\mathsf{syntrans}[n'])) :: \mathsf{Ty} \times \mathsf{ITm}$.

□

The final case in the proof of Theorem 2, for (n-syn-success), requires that Theorem 3, shown in Sec. 4.4 be mutually defined (we give a metric there that strictly decreases, to ensure that the mutual induction is well-founded).

Preservation extends straightforwardly to the multistep judgement $\sigma \mapsto^*_{\mathcal{A}} \sigma'$ in Figure 31, which is simply the reflexive, transitive closure of the stepping judgement.

Lemma 20 (Static Multistep Preservation)**.** *Letting $\mathcal{A} = \overline{e}; \Upsilon; \Phi$, if (a) $\sigma \mapsto^*_{\mathcal{A}} \sigma'$ and (b) $\emptyset\;\emptyset \vdash^n_\Phi \sigma :: \kappa$ and (c) $\vdash \Phi$ and (d) $\Upsilon\ \mathtt{ctx}_\Phi$ then $\emptyset\;\emptyset \vdash^n_\Phi \sigma :: \kappa$.*

Proof. Rule induction on (a).

Case (n-multi-refl). In this case, $\sigma' = \sigma$, so the conclusion is assumption (b).

Case (n-multi-step). In this case, (1) $\sigma \mapsto^*_{\mathcal{A}} \sigma''$ and (2) $\sigma'' \mapsto_{\mathcal{A}} \sigma'$. Applying the IH to (1), (b), (c) and (d), we have that (3) $\emptyset\;\emptyset \vdash^n_\Phi \sigma'' :: \kappa$. Applying Theorem 2 to (b), (3), (c) and (d), we have our conclusion. □

The normalization judgement is defined as follows:

Definition 2 (Static Normalization)**.** *$\sigma \Downarrow_{\mathcal{A}} \sigma'$ iff $\sigma \mapsto^*_{\mathcal{A}} \sigma'$ and $\sigma'\ \mathtt{val}_{\mathcal{A}}$.*

Corollary 1 (Static Normalization Preservation)**.** *If (a) $\sigma \Downarrow_{\mathcal{A}} \sigma'$ and (b) $\emptyset\ \emptyset \vdash^n_\Phi \sigma :: \kappa$ and (c) $\vdash \Phi$ and (d) $\Upsilon\ \mathtt{ctx}_\Phi$ then $\emptyset\ \emptyset \vdash^n_\Phi \sigma' :: \kappa$.*

Proof. By Definition 2 on (a), we have (1) $\sigma \mapsto^*_{\mathcal{A}} \sigma'$. By Lemma 20 on (1), (b), (c) and (d), we have our conclusion. □

3.6 Stability

Lemma 21 (Stability of Kinding)**.** *If (a) $\vdash \Phi\ \mathtt{sigsok}$ and (b) $\mathbf{\Delta}\ \mathbf{\Gamma} \vdash^n_\Phi \sigma :: \kappa$ and (c) $\vdash \Phi'\Phi\ \mathtt{sigsok}$ then $\mathbf{\Delta}\ \mathbf{\Gamma} \vdash^n_{\Phi'\Phi} \sigma :: \kappa$.*

Proof. By rule induction on (c). All cases except the following proceed by application of the IH to all kinding premises and re-application of the same rule, because they are defined for all Φ and all premises are checked under the same Φ.

Case (k-ty-ext). We have $\sigma = \textsc{tc}\langle\sigma_{\text{tyidx}}\rangle$ and (1) $\mathsf{tycon}\ \textsc{tc}\ \{\theta\} \sim \mathsf{tcsig}[\kappa_{\text{tyidx}}]\,\{\chi\} \in \Phi$ and (2) $\mathbf{\Delta}\ \mathbf{\Gamma} \vdash^n_\Phi \sigma_{\text{tyidx}} :: \kappa_{\text{tyidx}}$. By the definition of prepending on ordered mappings, we have (3) $\mathsf{tycon}\ \textsc{tc}\ \{\theta\} \sim \mathsf{tcsig}[\kappa_{\text{tyidx}}]\,\{\chi\} \in \Phi'\Phi$. By the IH on (a), (2) and (c), we have (4) $\mathbf{\Delta}\ \mathbf{\Gamma} \vdash^n_{\Phi'\Phi} \sigma_{\text{tyidx}} :: \kappa_{\text{tyidx}}$. Apply (k-ty-ext) on (3) and (4).

Case (k-tycase-ext). We have $\sigma = \mathsf{tycase}[\textsc{tc}](\sigma';\boldsymbol{x}.\sigma_1;\sigma_2)$ and (1) $\mathbf{\Delta}\ \mathbf{\Gamma} \vdash^n_\Phi \sigma' :: \mathsf{Ty}$ and (2) $\mathsf{tycon}\ \textsc{tc}\ \{\theta\} \sim \mathsf{tcsig}[\kappa_{\text{tyidx}}]\,\{\chi\} \in \Phi$ and (3) $\mathbf{\Delta}\ \mathbf{\Gamma}, \boldsymbol{x} :: \kappa_{\text{tyidx}} \vdash^n_\Phi \sigma_1 :: \kappa$ and (4) $\mathbf{\Delta}\ \mathbf{\Gamma} \vdash^n_\Phi \sigma_2 :: \kappa$.

By the IH on (a), (1) and (c), we have (5) $\mathbf{\Delta}\ \mathbf{\Gamma} \vdash^n_{\Phi'\Phi} \sigma' :: \mathsf{Ty}$.

By the definition of prepending on ordered mappings, we have (6) $\mathsf{tycon}\ \textsc{tc}\ \{\theta\} \sim \mathsf{tcsig}[\kappa_{\text{tyidx}}]\,\{\chi\} \in \Phi'\Phi$.

By the IH on (a), (3) and (c), we have (7) $\mathbf{\Delta}\ \mathbf{\Gamma}, \boldsymbol{x} :: \kappa_{\text{tyidx}} \vdash^n_{\Phi'\Phi} \sigma_1 :: \kappa$.

By the IH on (a), (4) and (c), we have (8) $\mathbf{\Delta}\ \mathbf{\Gamma} \vdash^n_{\Phi'\Phi} \sigma_2 :: \kappa$.

Apply (k-tycase-ext) to (5), (6), (7) and (8). □

Lemma 22 (Stability of Kinding 2)**.** *If (a) $\vdash \Phi\ \mathtt{sigsok}$ and (b) $\mathbf{\Delta}\ \mathbf{\Gamma} \vdash^n_\Phi \sigma :: \kappa$ and (c) $\vdash \Phi\Phi'\ \mathtt{sigsok}$ then $\mathbf{\Delta}\ \mathbf{\Gamma} \vdash^n_{\Phi\Phi'} \sigma :: \kappa$.*

Proof. By rule induction on (c). All cases except the following proceed by application of the IH to all kinding premises and re-application of the same rule, because they are defined for all Φ and all premises are checked under the same Φ.

Case (k-ty-ext). We have $\sigma = \textsc{tc}\langle\sigma_{\text{tyidx}}\rangle$ and (1) $\mathsf{tycon}\ \textsc{tc}\ \{\theta\} \sim \mathsf{tcsig}[\kappa_{\text{tyidx}}]\,\{\chi\} \in \Phi$ and (2) $\mathbf{\Delta}\ \mathbf{\Gamma} \vdash^n_\Phi \sigma_{\text{tyidx}} :: \kappa_{\text{tyidx}}$. By the definition of appending ordered mappings, we have (3) $\mathsf{tycon}\ \textsc{tc}\ \{\theta\} \sim \mathsf{tcsig}[\kappa_{\text{tyidx}}]\,\{\chi\} \in \Phi\Phi'$. By the IH on (a), (2) and (c), we have (4) $\mathbf{\Delta}\ \mathbf{\Gamma} \vdash^n_{\Phi\Phi'} \sigma_{\text{tyidx}} :: \kappa_{\text{tyidx}}$. Apply (k-ty-ext) on (3) and (4).

Case (k-tycase-ext). We have $\sigma = \mathsf{tycase}[\textsc{tc}](\sigma';\boldsymbol{x}.\sigma_1;\sigma_2)$ and (1) $\mathbf{\Delta}\ \mathbf{\Gamma} \vdash^n_\Phi \sigma' :: \mathsf{Ty}$ and (2) $\mathsf{tycon}\ \textsc{tc}\ \{\theta\} \sim \mathsf{tcsig}[\kappa_{\text{tyidx}}]\,\{\chi\} \in \Phi$ and (3) $\mathbf{\Delta}\ \mathbf{\Gamma}, \boldsymbol{x} :: \kappa_{\text{tyidx}} \vdash^n_\Phi \sigma_1 :: \kappa$ and (4) $\mathbf{\Delta}\ \mathbf{\Gamma} \vdash^n_\Phi \sigma_2 :: \kappa$.

By the IH on (a), (1) and (c), we have (5) $\mathbf{\Delta}\ \mathbf{\Gamma} \vdash^n_{\Phi\Phi'} \sigma' :: \mathsf{Ty}$.

By the definition of appending ordered mappings, we have (6) $\mathsf{tycon}\ \textsc{tc}\ \{\theta\} \sim \mathsf{tcsig}[\kappa_{\text{tyidx}}]\,\{\chi\} \in \Phi\Phi'$.

By the IH on (a), (3) and (c), we have (7) $\mathbf{\Delta}\ \mathbf{\Gamma}, \boldsymbol{x} :: \kappa_{\text{tyidx}} \vdash^n_{\Phi\Phi'} \sigma_1 :: \kappa$.

By the IH on (a), (4) and (c), we have (8) $\mathbf{\Delta}\ \mathbf{\Gamma} \vdash^n_{\Phi\Phi'} \sigma_2 :: \kappa$.

Apply (k-tycase-ext) to (5), (6), (7) and (8). □

Corollary 2 (Stability of Kinding 3). *If (a)* $\vdash \Phi$ *and (b)* $\mathbf{\Delta}\ \mathbf{\Gamma} \vdash^n_\Phi \sigma :: \kappa$ *and (c)* $\vdash \Phi\Phi'$ *then* $\mathbf{\Delta}\ \mathbf{\Gamma} \vdash^n_{\Phi\Phi'} \sigma :: \kappa$.

Proof. Applying Lemma 6 on (a), we have (1) $\vdash \Phi$ `sigsok`.

Applying Lemma 6 on (c), we have (2) $\vdash \Phi\Phi'$ `sigsok`.

Apply Lemma 22 to (1), (b) and (2). □

We can now give corresponding stability lemmas about tycon context extension.

Lemma 23 (Stability of Opcon Structure Validity). *If (a)* $\vdash \Phi$ `sigsok` *and (b)* $\vdash_\Phi \omega \sim \psi$ *and (c)* $\vdash \Phi'\Phi$ `sigsok` *then* $\vdash_{\Phi'\Phi} \omega \sim \psi$.

Proof. Rule induction on (b).

Case (ocstruct-intro). We have $\omega = \mathsf{ana\ intro} = \sigma_{\text{def}}$ and (1) $\emptyset \vdash \kappa_{\text{tmidx}}$ and (2) $\emptyset\ \emptyset \vdash^0_\Phi \sigma_{\text{def}} :: \kappa_{\text{tyidx}} \to \kappa_{\text{tmidx}} \to \mathsf{List[Arg]} \to \mathsf{ITm}$.

By Lemma 21 on (a), (2) and (c), we have (3) $\emptyset\ \emptyset \vdash^0_{\Phi'\Phi} \sigma_{\text{def}} :: \kappa_{\text{tyidx}} \to \kappa_{\text{tmidx}} \to \mathsf{List[Arg]} \to \mathsf{ITm}$.

Apply (ocstruct-intro) to (1) and (3).

Case (ocstruct-targ). We have $\omega = \omega'; \mathsf{syn}\ \mathbf{op} = \sigma_{\text{def}}$ and $\psi = \mathsf{tcsig}[\kappa_{\text{tyidx}}]\ \{\chi; \mathbf{op}[\kappa_{\text{tmidx}}]\}$ and (1) $\vdash_\Phi \omega' \sim \mathsf{tcsig}[\kappa_{\text{tyidx}}]\ \{\chi\}$ and (2) $\emptyset \vdash \kappa_{\text{tmidx}}$ and (3) $\emptyset\ \emptyset \vdash^0_\Phi \sigma_{\text{def}} :: \kappa_{\text{tyidx}} \to \kappa_{\text{tmidx}} \to \mathsf{List[Arg]} \to (\mathsf{Ty} \times \mathsf{ITm})$.

By the IH on (a), (1) and (c), we have (4) $\vdash_{\Phi'\Phi} \omega' \sim \mathsf{tcsig}[\kappa_{\text{tyidx}}]\ \{\chi\}$.

By Lemma 21 on (a), (3) and (c), we have (5) $\emptyset\ \emptyset \vdash^0_{\Phi'\Phi} \sigma_{\text{def}} :: \kappa_{\text{tyidx}} \to \kappa_{\text{tmidx}} \to \mathsf{List[Arg]} \to (\mathsf{Ty} \times \mathsf{ITm})$.

Apply (ocstruct-targ) to (4), (2) and (5). □

Lemma 24 (Stability of Opcon Structure Validity 2). *If (a)* $\vdash \Phi$ `sigsok` *and (b)* $\vdash_\Phi \omega \sim \psi$ *and (c)* $\vdash \Phi\Phi'$ `sigsok` *then* $\vdash_{\Phi\Phi'} \omega \sim \psi$.

Proof. Rule induction on (b).

Case (ocstruct-intro). We have $\omega = \mathsf{ana\ intro} = \sigma_{\text{def}}$ and (1) $\emptyset \vdash \kappa_{\text{tmidx}}$ and (2) $\emptyset\ \emptyset \vdash^0_\Phi \sigma_{\text{def}} :: \kappa_{\text{tyidx}} \to \kappa_{\text{tmidx}} \to \mathsf{List[Arg]} \to \mathsf{ITm}$.

By Lemma 22 on (a), (2) and (c), we have (3) $\emptyset\ \emptyset \vdash^0_{\Phi\Phi'} \sigma_{\text{def}} :: \kappa_{\text{tyidx}} \to \kappa_{\text{tmidx}} \to \mathsf{List[Arg]} \to \mathsf{ITm}$.

Apply (ocstruct-intro) to (1) and (3).

Case (ocstruct-targ). We have $\omega = \omega'; \mathsf{syn}\ \mathbf{op} = \sigma_{\text{def}}$ and $\psi = \mathsf{tcsig}[\kappa_{\text{tyidx}}]\ \{\chi; \mathbf{op}[\kappa_{\text{tmidx}}]\}$ and (1) $\vdash_\Phi \omega' \sim \mathsf{tcsig}[\kappa_{\text{tyidx}}]\ \{\chi\}$ and (2) $\emptyset \vdash \kappa_{\text{tmidx}}$ and (3) $\emptyset\ \emptyset \vdash^0_\Phi \sigma_{\text{def}} :: \kappa_{\text{tyidx}} \to \kappa_{\text{tmidx}} \to \mathsf{List[Arg]} \to (\mathsf{Ty} \times \mathsf{ITm})$.

By the IH on (a), (1) and (c), we have (4) $\vdash_{\Phi\Phi'} \omega' \sim \mathsf{tcsig}[\kappa_{\text{tyidx}}]\ \{\chi\}$.

By Lemma 21 on (a), (3) and (c), we have (5) $\emptyset\ \emptyset \vdash^0_{\Phi\Phi'} \sigma_{\text{def}} :: \kappa_{\text{tyidx}} \to \kappa_{\text{tmidx}} \to \mathsf{List[Arg]} \to (\mathsf{Ty} \times \mathsf{ITm})$.

Apply (ocstruct-targ) to (4), (2) and (5). □

Corollary 3 (Stability of Opcon Structure Validity 3). *If (a)* $\vdash \Phi$ *and (b)* $\vdash_\Phi \omega \sim \psi$ *and (c)* $\vdash \Phi\Phi'$ *then* $\vdash_{\Phi\Phi'} \omega \sim \psi$.

Proof. Applying Lemma 6 to (a), we have (1) $\vdash \Phi\ \texttt{sigsok}$.

Applying Lemma 6 to (c), we have (2) $\vdash \Phi\Phi'\ \texttt{sigsok}$.

Apply Lemma 24 to (1), (b) and (2). $\square$

Lemma 25 (Tycon Context Extension)**.** *If (a)* $\vdash \Phi$ *and (b)* $\vdash \Phi'$ *and (c) dom*$(\Phi) \cap$ *dom*$(\Phi') = \emptyset$ *then* $\vdash \Phi\Phi'$.

Proof. Rule induction on (b).

Case (tcc-emp). Here, $\Phi' = \emptyset$, so $\Phi\Phi' = \Phi$ and $\vdash \Phi$ by (a).

Case (tcc-ext). Here, $\Phi' = \Phi'', \mathsf{tycon}\ \textsc{tc}\ \{\mathsf{trans} = \sigma_{\text{schema}}\ \mathsf{in}\ \omega\} \sim \mathsf{tcsig}[\kappa_{\text{tyidx}}]\ \{\chi\}$ and (1) $\vdash \Phi''$ and (2) $\vdash \kappa_{\text{tyidx}}\ \mathsf{eq}$ and (3) $\emptyset\ \emptyset \vdash^{0}_{\Phi''} \sigma_{\text{schema}} :: \kappa_{\text{tyidx}} \to \mathsf{ITy}$ and (4) $\vdash_{\Phi'', \mathsf{tycon}\ \textsc{tc}\ \{\mathsf{trans} = \sigma_{\text{schema}}\ \mathsf{in}\ \omega\} \sim \mathsf{tcsig}[\kappa_{\text{tyidx}}]\ \{\chi\}} \omega \sim \mathsf{tcsig}[\kappa_{\text{tyidx}}]\ \{\chi\}$.

By the fact that Φ'' is a prefix of Φ' and (c), we have that (5) $\text{dom}(\Phi) \cap \text{dom}(\Phi'')$.

By the IH on (a), (1) and (5), we have (6) $\vdash \Phi\Phi''$.

By Lemma 6 on (6), we have (7) $\vdash \Phi\Phi''\ \texttt{sigsok}$.

By Lemma 6 on (1), we have (8) $\vdash \Phi''\ \texttt{sigsok}$.

By Lemma 21 on (8), (3) and (7), we have (9) $\emptyset\ \emptyset \vdash^{0}_{\Phi\Phi''} \sigma_{\text{schema}} :: \kappa_{\text{tyidx}} \to \mathsf{ITy}$.

By Lemma 5 on (4), we have (10) $\vdash \chi$.

By (tcsig-ok) on (2) and (10), we have (11) $\vdash \mathsf{tcsig}[\kappa_{\text{tyidx}}]\ \{\chi\}$.

By (tcc-sigok-ext) on (7) and (11), we have (12) $\vdash \Phi\Phi'', \mathsf{tycon}\ \textsc{tc}\ \{\mathsf{trans} = \sigma_{\text{schema}}\ \mathsf{in}\ \omega\} \sim \mathsf{tcsig}[\kappa_{\text{tyidx}}]\ \{\chi\}\ \texttt{sigsok}$.

By (tcc-sigok-ext) on (8) and (11), we have (13) $\vdash \Phi'', \mathsf{tycon}\ \textsc{tc}\ \{\mathsf{trans} = \sigma_{\text{schema}}\ \mathsf{in}\ \omega\} \sim \mathsf{tcsig}[\kappa_{\text{tyidx}}]\ \{\chi\}\ \texttt{sigsok}$.

By Lemma 23 on (13), (4) and (12), we have (14) $\vdash_{\Phi\Phi'', \mathsf{tycon}\ \textsc{tc}\ \{\mathsf{trans} = \sigma_{\text{schema}}\ \mathsf{in}\ \omega\} \sim \mathsf{tcsig}[\kappa_{\text{tyidx}}]\ \{\chi\}} \omega \sim \mathsf{tcsig}[\kappa_{\text{tyidx}}]\ \{\chi\}$.

By (tcc-ext) on (6), (2), (9) and (14), we have our conclusion. $\square$

4 External Language

4.1 Syntax

We now turn to the external language. The abstract syntax of the external language is given in Figure 4 and Figures 5 and 6 give various desugarings. Although we do not detail it here, we expect a technique based on [3] could be introduced to allow new desugarings to be introduced composably.

Argument lists, $\overline{e}$, are assumed to be defined as metatheoretic lists, so we assume standard operations like computing the length, written $|\overline{e}| = n$, or retrieving an element, written $\mathsf{nth}[n](\overline{e}) = e$, are already defined and obey standard metatheoretic properties.

4.2 Typing

The typing rules are given in Figure 44. We will cover their premises in the next several subsections, then return to the metatheory after that.

4.3 Argument Interfaces

Definition 3 (Argument Interface Kind)**.** *The kind abbreviated* Arg *is defined as*

$$\mathsf{Arg} := (\mathsf{Ty} \to \mathsf{ITm}) \times (1 \to \mathsf{Ty} \times \mathsf{ITm})$$

Lemma 26 (Argument Interface Kind is a Kind)**.** *We have that* $\emptyset \vdash \mathsf{Arg}$.

Proof. We apply the following sequences of rules to construct the derivation in a bottom up, left-to-right manner:

Apply (kf-prod). Apply (kf-arrow). Apply (kf-ty). Apply (kf-itm). Apply (kf-arrow). Apply (kf-unit). Apply (kf-prod). Apply (kf-ty). Apply (kf-itm). □

The argument interfaces that populate the list provided to opcon definitions is derived from the argument list $\overline{e}$ by the judgement $\mathsf{args}(n) = \sigma_{\text{args}}$, where the n we use is the length of the argument list (including the target argument when applicable). This judgement is defined in Figure 33 with an auxiliary judgement $\mathsf{args}(n, m) = \sigma_{\text{args}}$, where m is used to count up to n so that the argument indices are in the correct order.

Lemma 27 (Argument Interface Lists (Auxiliary))**.** *If* $\mathsf{args}(n, m) = \sigma_{args}$ *then (i)* σ_{args} $\mathtt{val}_{\mathcal{A}}$ *and (ii)* $\emptyset\ \emptyset \vdash^{n+m}_{\Phi} \sigma_{args} :: \mathsf{List}[\mathsf{Arg}]$.

Proof. Rule induction on the assumption.

Case (mk-arg-interface-nil). In this case, $n = 0$ and thus $n + m = m$, and $\sigma_{\text{args}} = \mathsf{nil}[\mathsf{Arg}]$. Applying (n-nil-val), we have (i) $\mathsf{nil}[\mathsf{Arg}]\ \mathtt{val}_{\mathcal{A}}$. By Lemma 26, we have (1) $\emptyset \vdash \mathsf{Arg}$. Applying (k-nil) to (1), we have (ii) $\emptyset\ \emptyset \vdash^{m}_{\Phi} \mathsf{nil}[\mathsf{Arg}] :: \mathsf{List}[\mathsf{Arg}]$.

Case (mk-arg-interface-cons). In this case, $n = n' + 1$ and (1) $\mathsf{args}(n', m + 1) = \sigma'_{\text{args}}$ and

$$\sigma_{\text{args}} = \mathsf{cons}((\lambda \mathsf{Ty}{::}\boldsymbol{ty}.\mathsf{ana}[m](\boldsymbol{ty}), \lambda\_{::}1.\mathsf{syn}[m]); \sigma'_{\text{args}})$$

Applying the IH to (1), we have that (2) σ'_{args} $\texttt{val}_{\mathcal{A}}$ and (3) $\emptyset\ \emptyset \vdash_{\Phi}^{n'+(m+1)} \sigma'_{\text{args}} :: \mathsf{List}[\mathsf{Arg}]$. Noting that $(n'+1)+m = n'+(m+1)$ by the usual properties of arithmetic, we thus have that (4) $\emptyset\ \emptyset \vdash_{\Phi}^{(n'+1)+m} \sigma'_{\text{args}} :: \mathsf{List}[\mathsf{Arg}]$.

Applying (kf-ty), we have (5) $\emptyset \vdash \mathsf{Ty}$.

Applying (k-var), we have that (6) $\emptyset\ \emptyset, \boldsymbol{ty} :: \mathsf{Ty} \vdash_{\Phi}^{(n'+1)+m} \boldsymbol{ty} :: \mathsf{Ty}$.

We have that (7) $m < (n'+1)+m$ by the usual properties of arithmetic inequalities.

Applying (k-ana) to (7) and (6), we have (8) $\emptyset\ \emptyset, \boldsymbol{ty} :: \mathsf{Ty} \vdash_{\Phi}^{(n'+1)+m} \mathsf{ana}[m](\boldsymbol{ty}) :: \mathsf{ITm}$.

Applying (k-abs) to (5) and (8), we have (9) $\emptyset\ \emptyset \vdash_{\Phi}^{(n'+1)+m} \lambda\boldsymbol{ty}::\mathsf{Ty}.\mathsf{ana}[m](\boldsymbol{ty}) :: \mathsf{Ty} \rightarrow \mathsf{ITm}$.

Applying (k-syn) to (7), we have (10) $\emptyset\ \emptyset, \_ :: 1 \vdash_{\Phi}^{(n'+1)+m} \mathsf{syn}[m] :: \mathsf{Ty} \times \mathsf{ITm}$.

Applying (kf-unit), we have (11) $\emptyset \vdash 1$.

Applying (k-abs) to (11) and (1), we have (12) $\emptyset\ \emptyset \vdash_{\Phi}^{(n'+1)+m} \lambda\_::1.\mathsf{syn}[m] :: 1 \rightarrow \mathsf{Ty} \times \mathsf{ITm}$.

Applying (k-pair) to (9) and (12), we have (13)

$$\emptyset\ \emptyset \vdash_{\Phi}^{(n'+1)+m} (\lambda\mathsf{Ty}::\boldsymbol{ty}.\mathsf{ana}[m](\boldsymbol{ty}), \lambda\_::1.\mathsf{syn}[m]) :: \mathsf{Arg}$$

Applying (k-cons) to (12) and (4), we have (ii)

$$\emptyset\ \emptyset \vdash_{\Phi}^{(n'+1)+m} \mathsf{cons}((\lambda\mathsf{Ty}::\boldsymbol{ty}.\mathsf{ana}[m](\boldsymbol{ty}), \lambda\_::1.\mathsf{syn}[m]); \sigma'_{\text{args}}) :: \mathsf{List}[\mathsf{Arg}]$$

Applying (n-abs-val), we have (14) $\lambda\mathsf{Ty}::\boldsymbol{ty}.\mathsf{ana}[m](\boldsymbol{ty})$ $\texttt{val}_{\mathcal{A}}$.

Applying (n-abs-val), we have (15) $\lambda\_::1.\mathsf{syn}[m]$ $\texttt{val}_{\mathcal{A}}$.

Applying (n-pair-val) to (14) and (15), we have (16) $(\lambda\mathsf{Ty}::\boldsymbol{ty}.\mathsf{ana}[m](\boldsymbol{ty}), \lambda\_::1.\mathsf{syn}[m])$ $\texttt{val}_{\mathcal{A}}$.

Applying (n-cons-val) to (16) and (2), we have (i)

$$\mathsf{cons}((\lambda\mathsf{Ty}::\boldsymbol{ty}.\mathsf{ana}[m](\boldsymbol{ty}), \lambda\_::1.\mathsf{syn}[m]); \sigma'_{\text{args}})\ \texttt{val}_{\mathcal{A}}$$

□

Lemma 28 (Argument Interface Lists). *If* $\mathsf{args}(n) = \sigma_{args}$ *then (i)* σ_{args} $\texttt{val}_{\mathcal{A}}$ *and (ii)* $\emptyset\ \emptyset \vdash_{\Phi}^{n} \sigma_{args} :: \mathsf{List}[\mathsf{Arg}]$.

Proof. Rule induction on the assumption. There is only one rule, (mk-arg-interface), so we have that (1) $\mathsf{args}(n, 0) = \sigma_{\text{args}}$. Applying Lemma 27 to (1) and noting that $n + 0 = n$ by definition, we have (i) σ_{args} $\texttt{val}_{\mathcal{A}}$ and (ii) $\emptyset\ \emptyset \vdash_{\Phi}^{n} \sigma_{\text{args}} :: \mathsf{List}[\mathsf{Arg}]$. □

4.4 Type Synthesis and Static Preservation

To prove the type synthesis theorem below, we need a notion of external typing context validity, defined by the judgement Υ $\texttt{ctx}_{\Phi}$ in Figure 32. It simply ensures that all bindings in Υ map to types.

Lemma 29 (Typing Context Validity). *If (a)* Υ $\texttt{ctx}_{\Phi}$ *and (b)* $x : \sigma \in \Upsilon$ *then* σ $\texttt{type}_{\Phi}$.

Proof. Rule induction on (a) and (b).

Case (tctx-ok-emp). Does not apply by (b).

Case (tctx-ok-ext). In this case, $\Upsilon = \Upsilon', x' : \sigma'$ and (1) $\Upsilon\ \texttt{ctx}'_{\Phi}$ and (2) $\sigma'\ \texttt{type}_{\Phi}$. We have two cases by (b):

> **Case** $x = x'$ and $\sigma = \sigma'$. In this case, we have our conclusion by (2).
>
> **Case** $x \neq x'$ and (3) $x : \sigma \in \Upsilon'$. In this case, we apply our IH to (1) and (3) to conclude.

□

Theorem 3 (Type Synthesis)**.** *If (a)* $\vdash \Phi$ *and (b)* $\Upsilon\ \texttt{ctx}_{\Phi}$ *and (c)* $\Upsilon \vdash_{\Phi} e \Rightarrow \sigma \leadsto \iota$ *then* $\sigma\ \texttt{type}_{\Phi}$.

Proof. Rule induction on (c).

Case (ascribe). In this case, we have $e = e' : \sigma'$ and (1) $\emptyset\ \emptyset \vdash^0_{\Phi} \sigma' :: \mathsf{Ty}$ and (2) $\sigma' \Downarrow_{\cdot;\emptyset;\Phi} \sigma$. By Corollary 1 on (2), (1), (a) and (b), we have (3) $\emptyset\ \emptyset \vdash^0_{\Phi} \sigma :: \mathsf{Ty}$. By Definition 2 on (2), we have (4) $\sigma\ \texttt{val}_{\cdot;\emptyset;\Phi}$. Apply Definition 1 to (3) and (4).

Case (var). In this case, we have (1) $x : \sigma \in \Upsilon$. Apply Lemma 29 to (a) and (1).

Case (syn-lam). In this case, we have $e = \lambda x{:}\sigma_1.e'$ and $\sigma = \rightharpoonup\langle(\sigma'_1, \sigma_2)\rangle$ and (1) $\emptyset\ \emptyset \vdash^0_{\Phi} \sigma_1 :: \mathsf{Ty}$ and (2) $\sigma_1 \Downarrow_{\cdot;\emptyset;\Phi} \sigma'_1$ and (3) $\Upsilon, x : \sigma'_1 \vdash_{\Phi} e' \Rightarrow \sigma_2 \leadsto \iota'$.

By Corollary 1 on (2), (1), (a) and (b), we have (3) $\emptyset\ \emptyset \vdash^0_{\Phi} \sigma'_1 :: \mathsf{Ty}$. By Definition 2 on (2), we have (4) $\sigma'_1\ \texttt{val}_{\cdot;\emptyset;\Phi}$. By Definition 1 on (3) and (4), we have (5) $\sigma'_1\ \texttt{type}_{\Phi}$.

By (tctx-ok-ext) to (b) and (5), we have (6) $\Upsilon, x : \sigma'_1\ \texttt{ctx}_{\Phi}$.

Applying the IH to (a), (6) and (3), we have (7) $\sigma_2\ \texttt{type}_{\Phi}$ and so by Definition 1 on (7), we have (8) $\emptyset\ \emptyset \vdash^0_{\Phi} \sigma_2 :: \mathsf{Ty}$ and (9) $\sigma_2\ \texttt{val}_{\cdot;\emptyset;\Phi}$.

Applying (k-pair) to (3) and (8), we have (10) $\emptyset\ \emptyset \vdash^0_{\Phi} (\sigma'_1, \sigma_2) :: \mathsf{Ty} \times \mathsf{Ty}$.

Applying (k-ty-parr) to (10), we have (11) $\emptyset\ \emptyset \vdash^0_{\Phi} \rightharpoonup\langle(\sigma'_1, \sigma_2)\rangle :: \mathsf{Ty}$.

Applying (n-pair-val) to (4) and (9), we have (12) $(\sigma'_1, \sigma_2)\ \texttt{val}_{\cdot;\emptyset;\Phi}$.

Applying (n-ty-val) to (12), we have $\rightharpoonup\langle(\sigma'_1, \sigma_2)\rangle\ \texttt{val}_{\cdot;\emptyset;\Phi}$.

Apply Definition 1 to (11) and (13).

Case (syn-ap). In this case, $e = e_1(e_2)$ and (1) $\Upsilon \vdash_{\Phi} e_1 \Rightarrow \rightharpoonup\langle(\sigma_1, \sigma)\rangle \leadsto \iota_1$. Applying the IH to (a), (b) and (1), we have (2) $\rightharpoonup\langle(\sigma_1, \sigma)\rangle\ \texttt{type}_{\Phi}$. By Definition 1 on (2), we have (3) $\emptyset\ \emptyset \vdash^0_{\Phi} \rightharpoonup\langle(\sigma_1, \sigma)\rangle :: \mathsf{Ty}$ and (4) $\rightharpoonup\langle(\sigma_1, \sigma)\rangle\ \texttt{val}_{\mathcal{A}}$. By Lemma 17 on (4) and (3), we have $\sigma\ \texttt{type}_{\Phi}$.

Case (syn-targ). In this case, $e = \mathsf{targop}[\mathbf{op}; \sigma_{\text{tmidx}}](e_{\text{targ}}; \overline{e})$ and

(1) $\Upsilon \vdash_{\Phi} e_{\text{targ}} \Rightarrow \textsc{tc}\langle\sigma_{\text{tyidx}}\rangle \leadsto \iota_{\text{targ}}$ and

(2) $\mathsf{tycon}\ \textsc{tc}\ \{\mathsf{trans} = \sigma_{\text{schema}}\ \mathsf{in}\ \omega\} \sim \mathsf{tcsig}[\kappa_{\text{tyidx}}]\ \{\chi\} \in \Phi$ and

(3) $\mathbf{op}[\kappa_{\text{tmidx}}] \in \chi$ and

(4) $\emptyset\ \emptyset \vdash^0_{\Phi} \sigma_{\text{tmidx}} :: \kappa_{\text{tmidx}}$ and

(5) $\mathsf{syn}\ \mathbf{op} = \sigma_{\text{def}} \in \omega$ and

(6) $|e_{\text{targ}}; \overline{e}| = n$ and

(7) $\mathsf{args}(n) = \sigma_{\text{args}}$ and

(8) $\sigma_{\text{def}}(\sigma_{\text{tyidx}})(\sigma_{\text{tmidx}})(\sigma_{\text{args}}) \Downarrow_{(e_{\text{targ}};\overline{e});\Upsilon;\Phi} (\sigma, \rhd(\hat{\iota}))$.

Applying the IH to (a), (b) and (1), we have (9) $\texttt{TC}\langle\sigma_{\text{tyidx}}\rangle\ \texttt{type}_\Phi$. By Lemma 17.7 on (9), we have (10) $\emptyset\ \emptyset \vdash^0_\Phi \sigma_{\text{tyidx}} :: \kappa_{\text{tyidx}}$. Applying Lemma 14 to (10), we have (11) $\emptyset\ \emptyset \vdash^n_\Phi \sigma_{\text{tyidx}} :: \kappa_{\text{tyidx}}$.

Applying Lemma 1 to (2), we have that there is a prefix Φ' of Φ such that (12) $\vdash \Phi'$ and (13) $\vdash \Phi', \mathsf{tycon}\ \texttt{TC}\ \{\mathsf{trans} = \sigma_{\text{schema}}\ \mathsf{in}\ \omega\} \sim \mathsf{tcsig}[\kappa_{\text{tyidx}}]\ \{\chi\}$ and (14) $\vdash_{\Phi', \mathsf{tycon}\ \texttt{TC}\ \{\mathsf{trans} = \sigma_{\text{schema}}\ \mathsf{in}\ \omega\} \sim \mathsf{tcsig}[\kappa_{\text{tyidx}}]\ \{\chi\}}\ \omega \sim \mathsf{tcsig}[\kappa_{\text{tyidx}}]\ \{\chi\}$.

Applying Lemma 6 to (12), we have (15) $\vdash \Phi', \mathsf{tycon}\ \texttt{TC}\ \{\mathsf{trans} = \sigma_{\text{schema}}\ \mathsf{in}\ \omega\} \sim \mathsf{tcsig}[\kappa_{\text{tyidx}}]\ \{\chi\}\ \texttt{sigsok}$. Applying Lemma 6 to (a), taking by the definition of prefixing that $\Phi = (\Phi', \mathsf{tycon}\ \texttt{TC}\ \{\mathsf{trans} = \sigma_{\text{schema}}\ \mathsf{in}\ \omega\} \sim \mathsf{tcsig}[\kappa_{\text{tyidx}}]\ \{\chi\})\Phi''$, we have (16) $\vdash (\Phi', \mathsf{tycon}\ \texttt{TC}\ \{\mathsf{trans} = \sigma_{\text{schema}}\ \mathsf{in}\ \omega\} \sim \mathsf{tcsig}[\kappa_{\text{tyidx}}]\ \{\chi\})\Phi''\ \texttt{sigsok}$. Applying Lemma 23 to (15), (14) and (16), we have that (17) $\vdash_\Phi \omega \sim \mathsf{tcsig}[\kappa_{\text{tyidx}}]\ \{\chi\}$.

Applying Lemma 3 to (17) and (5), we have that (18) $\emptyset\ \emptyset \vdash^0_\Phi \sigma_{\text{def}} :: \kappa_{\text{tyidx}} \to \kappa_{\text{tmidx}} \to \mathsf{List}[\mathsf{Arg}] \to (\mathsf{Ty} \times \mathsf{ITm})$. Applying Lemma 14 to (18), we have (19) $\emptyset\ \emptyset \vdash^n_\Phi \sigma_{\text{def}} :: \kappa_{\text{tyidx}} \to \kappa_{\text{tmidx}} \to \mathsf{List}[\mathsf{Arg}] \to (\mathsf{Ty} \times \mathsf{ITm})$.

Applying Lemma 28 to (7), we have that (20) $\emptyset\ \emptyset \vdash^n_\Phi \sigma_{\text{args}} :: \mathsf{List}[\mathsf{Arg}]$.

Applying (k-ap) three times with (19), (11), (4) and (20), we have that (21) $\emptyset\ \emptyset \vdash^n_\Phi \sigma_{\text{def}}(\sigma_{\text{tyidx}})(\sigma_{\text{tmidx}})(\sigma_{\text{args}}) :: \mathsf{Ty} \times \mathsf{ITm}$.

Applying Corollary 1 (see note on well-foundedness below) to (8), (21), (a) and (b), we have that (22) $\emptyset\ \emptyset \vdash^n_\Phi (\sigma, \rhd(\hat{\iota})) :: \mathsf{Ty} \times \mathsf{ITm}$. By Definition 2 on (8), we have (23) $(\sigma, \rhd(\hat{\iota}))\ \texttt{val}_{(e_{\text{targ}};\overline{e});\Upsilon;\Phi}$.

Applying Lemma 17.5 to (23) and (22), we have that (24) $\emptyset\ \emptyset \vdash^n_\Phi \sigma :: \mathsf{Ty}$ and (25) $\sigma\ \texttt{val}_{(e_{\text{targ}};\overline{e});\Upsilon;\Phi}$. Apply Definition 1 to (24) and (25).

Case (syn-targ-other). In this case, we have that $\sigma = \mathsf{targop}[\mathbf{op}; (\sigma, \rhd(\hat{\iota}))](e_{\text{targ}}; \overline{e})$ and (1) $\sigma\ \texttt{type}_\Phi$, so we can conclude by (1). □

The mutual induction is well-founded because the total number of intro and targeted terms that remain to be typechecked strictly decreases. In particular, we define the following metrics:

Definition 4 (Extension Operation Count). *The extension operation count, written $\|e\| = n$ on external terms and $\|\overline{e}\| = n$ on argument lists, is mutually defined as follows:*

$$\begin{array}{rcl}
\|x\| & = & 0 \\
\|\lambda x.e\| & = & \|e\| \\
\|\lambda x{:}\sigma.e\| & = & \|e\| \\
\|e_1(e_2)\| & = & \|e_1\| + \|e_2\| \\
\|\mathsf{fix}(x.e)\| & = & \|e\| \\
\|e : \sigma\| & = & \|e\| \\
\|\mathsf{intro}[\sigma](\overline{e})\| & = & \|\overline{e}\| + 1 \\
\|\mathsf{targop}[\boldsymbol{op}; \sigma](e_{targ}; \overline{e})\| & = & \|e_{targ}\| + \|\overline{e}\| + 1 \\
\\
\| \cdot \| & = & 0 \\
\|\overline{e}; e\| & = & \|\overline{e}\| + \|e\|
\end{array}$$

In the proof of Theorem 3 for an external term e such that $\|e\| = n$, Corollary 1 (of Theorem 2) is invoked only when $e = \mathsf{targop}[\mathbf{op}; \sigma'](e_{\text{targ}}; \overline{e}')$ with an argument list $\overline{e} = e_{\text{targ}}; \overline{e}'$. By definition, $\|e\| = \|e_{\text{targ}}\| + \|\overline{e}'\| + 1$ and $\|\overline{e}\| = \|e_{\text{targ}}\| + \|\overline{e}'\|$, so $\|\overline{e}\| < \|e\|$.

In the proof of Theorem 2 with an argument list $\overline{e}$ such that $\|\overline{e}\| = n$, Theorem 3 is invoked only on a term e extracted from $\overline{e}$, so by definition $\|e\| \leq \|\overline{e}\|$.

This metric strictly decreases every time we pass into Theorem 2 and does not increase in passing back to Theorem 3, so mutual induction is well-founded.

4.5 Type and Typing Context Translations

The type translation judgement $\vdash_\Phi \sigma\ \mathtt{type} \leadsto \tau$ specified in Figure 34 simply checks that σ is a type, then generates any selectively abstracted type translation and applies the type substitution it implies, discussed below.

Lemma 30 (Types with Translations are Types)**.** *If* $\vdash_\Phi \sigma\ \mathtt{type} \leadsto \tau$ *then* $\sigma\ \mathtt{type}_\Phi$.

Proof. Rule induction on the assumption. The conclusion is the first premise of the only rule. □

Because typing contexts map variables to types, we need an analagous context translation judgement, $\vdash_\Phi \Upsilon\ \mathtt{ctx} \leadsto \Gamma$, specified in Figure 35.

Lemma 31 (Typing Contexts with Translations are Valid Typing Contexts)**.** *If* $\vdash_\Phi \Upsilon\ \mathtt{ctx} \leadsto \Gamma$ *then* $\vdash_\Phi \Upsilon$.

Proof. Rule induction on the assumption.

Case (ctx-trans-emp). Apply (tctx-ok-emp).

Case (ctx-trans-ext). We have that $\Upsilon = \Upsilon', x : \sigma$ and (1) $\vdash_\Phi \Upsilon'\ \mathtt{ctx} \leadsto \Gamma'$ and (2) $\vdash_\Phi \sigma\ \mathtt{type} \leadsto \tau$.

Applying the IH to (1), we have that (3) $\vdash_\Phi \Upsilon'$. Applying Lemma 30 to (2), we have (4) $\sigma\ \mathtt{type}_\Phi$. Apply (tctx-ok-ext) to (3) and (4). □

4.5.1 Selectively Abstracted Type Translations

The rules for generating selectively abstracted type translations are given in Figure 36. They make use of a type translation store, $\mathcal{D}$. For our metatheory, we need a notion of a valid type translation store under tycon context Φ for delegated tycon c, written $\vdash_\Phi^c \mathcal{D}$ and specified in Figure 37. Each type translation store implies a type variable substitution and internal type formation context, Δ, by the internalization judgement $\mathcal{D} \leadsto \delta : \Delta$, specified in Figure 38.

Lemma 32 (Type Translation Store Validity)**.** *If (a)* $\vdash_\Phi^c \mathcal{D}$ *and (b)* $c'\langle\sigma_{tyidx}\rangle \leftrightarrow \tau/\alpha \in \mathcal{D}$ *then (i)* $c \neq c'$ *and (ii)* $\vdash_\Phi c'\langle\sigma_{tyidx}\rangle\ \mathtt{type} \leadsto \tau$ *and (iii)* $\emptyset \vdash \tau$.

Proof. Rule induction on (a) and (b).

Case (tyts-ok-emp). Does not apply by (b).

Case (tyts-ok-ext). We have two cases by (b):

Case $\mathcal{D} = \mathcal{D}', c'\langle\sigma_{\text{tyidx}}\rangle$ and (1) $c \neq c'$ and (2) $\vdash_\Phi c'\langle\sigma_{\text{tyidx}}\rangle\ \mathtt{type} \rightsquigarrow \tau$. and (3) $\emptyset \vdash \tau$. We can conclude with (1) and (2).

Case $\mathcal{D} = \mathcal{D}', c''\langle\sigma'_{\text{tyidx}}\rangle \leftrightarrow \tau/\alpha$ for $c' \neq c''$ and (1) $c'\langle\sigma_{\text{tyidx}}\rangle \leftrightarrow \tau/\alpha \in \mathcal{D}$ and (2) $\vdash^c_\Phi \mathcal{D}'$. Apply the IH to (2) and (1).

□

Lemma 33 (Selective Type Translation Abstraction Caching). *If (a)* $\emptyset\ \emptyset \vdash^n_\Phi \blacktriangleright(\hat{\tau}) :: \mathsf{ITy}$ *and (b)* $\blacktriangleright(\hat{\tau})\ \mathtt{val}_{\overline{e};\Upsilon;\Phi}$ *and (c)* $\vdash \Phi$ *and (d)* $\vdash^c_\Phi \mathcal{D}_1$ *and (e)* $\hat{\tau} \parallel \mathcal{D}_1 \hookrightarrow^c_\Phi \tau \parallel \mathcal{D}_2$ *then (i)* $\hat{\tau} \parallel \mathcal{D}_2 \hookrightarrow^c_\Phi \tau \parallel \mathcal{D}_2$ *and (ii)* $\hat{\tau} \parallel \emptyset \hookrightarrow^c_\Phi \tau \parallel \mathcal{D}_2$.

Proof. Rule induction on (e).

The proof follows by a straightforward application of the IH and re-application of the same rule in all cases except the following.

Case (abs-ext-not-delegated-new), (abs-other-not-delegated-new). In these cases, we can apply (abs-ty-stored) to arrive at (i). Conclusion (ii) follows by the IH.

Case (abs-ty-stored). In this case, conclusion (i) follows by re-applying (abs-ty-stored). Conclusion (ii) follows by applying Lemma 32, then inverting (ty-trans) and applying the IH. □

Lemma 34 (Selective Type Translation Abstraction). *If (a)* $\emptyset\ \emptyset \vdash^n_\Phi \blacktriangleright(\hat{\tau}) :: \mathsf{ITy}$ *and (b)* $\blacktriangleright(\hat{\tau})\ \mathtt{val}_{\overline{e};\Upsilon;\Phi}$ *and (c)* $\vdash \Phi$ *and (d)* $\vdash^c_\Phi \mathcal{D}_1$ *and (e)* $\hat{\tau} \parallel \mathcal{D}_1 \hookrightarrow^c_\Phi \tau \parallel \mathcal{D}_2$ *then (i)* $\vdash^c_\Phi \mathcal{D}_2$ *and given* $\mathcal{D}_1 \rightsquigarrow \delta_1 : \Delta_1$, *(ii)* $\mathcal{D}_2 \rightsquigarrow \delta_1\delta_2 : \Delta_1\Delta_2$ *and (iii)* $\emptyset \vdash \delta_1\delta_2 : \Delta_1\Delta_2$ *and (iv) if* $\hat{\tau} = \mathsf{trans}(\sigma)$ *then* $\Delta_1\Delta_2 \vdash \tau$.

Proof. Rule induction on (e).

Case (abs-parr). In this case, we have $\hat{\tau} = \mathsf{trans}(\rightharpoonup\langle(\sigma_1, \sigma_2)\rangle)$ and $\tau = \tau_1 \rightharpoonup \tau_2$ and

(1) $\mathsf{trans}(\sigma_1) \parallel \mathcal{D}_1 \hookrightarrow^c_\Phi \tau_1 \parallel \mathcal{D}'$

(2) $\mathsf{trans}(\sigma_2) \parallel \mathcal{D}' \hookrightarrow^c_\Phi \tau_2 \parallel \mathcal{D}_2$

By Lemma 17.8 on (a) and (b), we have (3) $\rightharpoonup\langle(\sigma_1, \sigma_2)\rangle\ \mathtt{type}_\Phi$. By Lemma 17.7 on (3), we have (4) $\sigma_1\ \mathtt{type}_\Phi$ and (5) $\sigma_2\ \mathtt{type}_\Phi$.

By Definition 1 on (4), we have (6) $\emptyset\ \emptyset \vdash^n_\Phi \sigma_1 :: \mathsf{Ty}$ and (7) $\sigma_1\ \mathtt{val}_{\overline{e};\Upsilon;\Phi}$.

By Definition 1 on (5), we have (8) $\emptyset\ \emptyset \vdash^n_\Phi \sigma_2 :: \mathsf{Ty}$ and (9) $\sigma_2\ \mathtt{val}_{\overline{e};\Upsilon;\Phi}$.

By (k-qity-parr) on (6), we have (10) $\emptyset\ \emptyset \vdash^n_\Phi \blacktriangleright(\mathsf{trans}(\sigma_1)) :: \mathsf{ITy}$. By (n-qity-parr) on (7), we have (11) $\blacktriangleright(\mathsf{trans}(\sigma_1))\ \mathtt{val}_{\overline{e};\Upsilon;\Phi}$.

By (k-qity-parr) on (8), we have (12) $\emptyset\ \emptyset \vdash^n_\Phi \blacktriangleright(\mathsf{trans}(\sigma_2)) :: \mathsf{ITy}$. By (n-qity-parr) on (9), we have (13) $\blacktriangleright(\mathsf{trans}(\sigma_2))\ \mathtt{val}_{\overline{e};\Upsilon;\Phi}$.

By the IH on (10), (11), (c), (d) and (1), we have (14) $\vdash^c_\Phi \mathcal{D}'$ and (15) $\mathcal{D}' \rightsquigarrow \delta_1\delta' : \Delta_1\Delta'$ and (16) $\emptyset \vdash \delta_1\delta' : \Delta_1\Delta'$ and (17) $\Delta_1\Delta \vdash \tau_1$.

By the IH on (12), (13), (c), (14) and (2), we have (i) $\vdash^c_\Phi \mathcal{D}_2$ and (ii) $\mathcal{D}_2 \rightsquigarrow \delta_1\delta'\delta_2 : \Delta_1\Delta'\Delta_2$ and (iii) $\emptyset \vdash \delta_1\delta'\delta_2 : \Delta_1\Delta'\Delta_2$ and (18) $\Delta_1\Delta'\Delta_2 \vdash \tau_2$.

By Weakening on (17), we have that (19) $\Delta_1\Delta'\Delta_2 \vdash \tau_1$ and so by (18) and (19) and the usual rules of internal type formation, we have (iv) $\Delta_1\Delta'\Delta_2 \vdash \tau_1 \rightharpoonup \tau_2$.

Case (abs-ext-delegated). In this case, we have $\hat{\tau} = \mathsf{trans}(\mathtt{TC}\langle\sigma_{\text{tyidx}}\rangle)$ and

(1) $\mathsf{tycon}\ \textsc{tc}\ \{\mathsf{trans} = \sigma_{\text{schema}}\ \mathsf{in}\ \omega\} \sim \psi \in \Phi$

(2) $\sigma_{\text{schema}}(\sigma_{\text{tyidx}}) \Downarrow_{\cdot;\emptyset;\Phi} \blacktriangleright(\hat{\tau}')$

(3) $\hat{\tau}' \parallel \mathcal{D}_1 \looparrowright^{\textsc{tc}}_{\Phi} \tau \parallel \mathcal{D}_2$

(4) $\mathcal{D}_2 \leadsto \delta : \Delta$

(5) $\Delta \vdash \tau$

By Lemma 6 on (1), we have that there exists a prefix Φ' of Φ such that (6) $\vdash \Phi'$ and (7) $\emptyset\ \emptyset \vdash^{0}_{\Phi'} \sigma_{\text{schema}} :: \kappa_{\text{tyidx}} \to \mathsf{ITy}$. By Corollary 2 on (6), (7) and (c), we have (8) $\emptyset\ \emptyset \vdash^{0}_{\Phi} \sigma_{\text{schema}} :: \kappa_{\text{tyidx}} \to \mathsf{ITy}$.

By Lemma 17.8 on (a) and (b), we have (9) $\textsc{tc}\langle\sigma_{\text{tyidx}}\rangle\ \mathtt{type}_{\Phi}$. By Lemma 19 on (9), we have (10) $\emptyset\ \emptyset \vdash^{0}_{\Phi} \sigma_{\text{tyidx}} :: \kappa_{\text{tyidx}}$ and (11) $\sigma_{\text{tyidx}}\ \mathtt{val}_{\overline{e};\Upsilon;\Phi}$.

By (k-ap) on (8) and (10), we have (12) $\emptyset\ \emptyset \vdash^{0}_{\Phi} \sigma_{\text{schema}}(\sigma_{\text{tyidx}}) :: \mathsf{ITy}$.

By (tctx-ok-emp), we have (13) $\vdash_{\Phi} \emptyset$.

By Corollary 1 on (2), (12), (c) and (13), we have (14) $\emptyset\ \emptyset \vdash^{0}_{\Phi} \blacktriangleright(\hat{\tau}') :: \mathsf{ITy}$.

By Definition 2 on (2), we have (15) $\blacktriangleright(\hat{\tau}')\ \mathtt{val}_{\overline{e};\Upsilon;\Phi}$.

Applying the IH to (14), (15), (c), (d) and (3), which is well-founded by the metric defined below, we have (i), (ii) and (iii). By (5), we have (iv).

Case (abs-ext-not-delegated-new). In this case, we have $\hat{\tau} = \mathsf{trans}(\textsc{tc}\langle\sigma_{\text{tyidx}}\rangle)$ and $\mathcal{D}_2 = \mathcal{D}', \textsc{tc}\langle\sigma_{\text{tyidx}}\rangle \leftrightarrow [\delta]\tau'/\alpha$ and

(1) $c \neq \textsc{tc}$

(2) $\textsc{tc}\langle\sigma_{\text{tyidx}}\rangle \notin \text{dom}(\mathcal{D}_1)$

(3) $\mathsf{tycon}\ \textsc{tc}\ \{\mathsf{trans} = \sigma_{\text{schema}}\ \mathsf{in}\ \omega\} \sim \psi \in \Phi$

(4) $\sigma_{\text{schema}}(\sigma_{\text{tyidx}}) \Downarrow_{\cdot;\emptyset;\Phi} \blacktriangleright(\hat{\tau}')$

(5) $\hat{\tau} \parallel \mathcal{D}_1 \looparrowright^{c}_{\Phi} \tau' \parallel \mathcal{D}'$

(6) $\mathcal{D}' \leadsto \delta : \Delta$

(7) $\Delta \vdash \tau'$

By Lemma 6 on (1), we have that there exists a prefix Φ' of Φ such that (6) $\vdash \Phi'$ and (7) $\emptyset\ \emptyset \vdash^{0}_{\Phi'} \sigma_{\text{schema}} :: \kappa_{\text{tyidx}} \to \mathsf{ITy}$. By Corollary 2 on (6), (7) and (c), we have (8) $\emptyset\ \emptyset \vdash^{0}_{\Phi} \sigma_{\text{schema}} :: \kappa_{\text{tyidx}} \to \mathsf{ITy}$.

By Lemma 17.8 on (a) and (b), we have (9) $\textsc{tc}\langle\sigma_{\text{tyidx}}\rangle\ \mathtt{type}_{\Phi}$. By Lemma 19 on (9), we have (10) $\emptyset\ \emptyset \vdash^{0}_{\Phi} \sigma_{\text{tyidx}} :: \kappa_{\text{tyidx}}$ and (11) $\sigma_{\text{tyidx}}\ \mathtt{val}_{\overline{e};\Upsilon;\Phi}$.

By (k-ap) on (8) and (10), we have (12) $\emptyset\ \emptyset \vdash^{0}_{\Phi} \sigma_{\text{schema}}(\sigma_{\text{tyidx}}) :: \mathsf{ITy}$.

By (tctx-ok-emp), we have (13) $\vdash_{\Phi} \emptyset$.

By Corollary 1 on (2), (12), (c) and (13), we have (14) $\emptyset\ \emptyset \vdash^{0}_{\Phi} \blacktriangleright(\hat{\tau}') :: \mathsf{ITy}$.

By Definition 2 on (2), we have (15) $\blacktriangleright(\hat{\tau}')\ \mathtt{val}_{\overline{e};\Upsilon;\Phi}$.

Applying the IH to (14), (15), (c), (d) and (3), which is well-founded by the metric defined below, we have (16) $\vdash^{c}_{\Phi} \mathcal{D}'$ and (17) $\mathcal{D}' \leadsto \delta_1\delta' : \Delta_1\Delta'$ and (18) $\emptyset \vdash \delta_1\delta' : \Delta_1\Delta'$.

Applying (ty-store-int-ext) to (6) we have (19) $\mathcal{D}_2 \rightsquigarrow (\delta_1\delta', [\delta_1\delta']\tau/\alpha) : \Delta_1\Delta', \alpha$, i.e. (ii). By (7) and Weakening, we have (iv).

By definition, $[\delta_1\delta', [\delta_1\delta']\tau'/\alpha]\alpha = [\delta_1\delta']\tau'$.

By Lemma 33 on (a-e), we have (20) $\vdash_\Phi \sigma\ \mathtt{type} \rightsquigarrow [\delta_1\delta']\tau'$.

Applying (ty-trans) to (9), (20) and (19), we have (21) $\vdash_\Phi \mathtt{TC}\langle\sigma_{\text{tyidx}}\rangle\ \mathtt{type} \rightsquigarrow [\delta_1\delta']\tau'$.

By Assumption 9 on (7) and (iii), we have (22) $\emptyset \vdash [\delta_1\delta']\tau'$.

Applying (tyts-ok-ext) to (16), (21) and (19) and (22), we have (i). By (tysub-ext) on (18) and (22), we have (iii).

Case (abs-other-delegated). In this case, $\hat{\tau} = \mathsf{trans}(\mathsf{other}[m;\kappa]\langle(\sigma_{\text{data}}, \blacktriangleright(\hat{\tau}'))\rangle)$ and

1. $\hat{\tau}' \parallel \mathcal{D}_1 \looparrowright^{\mathsf{other}[m;\kappa]}_\Phi \tau \parallel \mathcal{D}_2$.
2. $\mathcal{D}_2 \rightsquigarrow \delta : \Delta$
3. $\Delta \vdash \tau$

By Lemma 17 iterated on (a) and (b), we have (4) $\emptyset\ \emptyset \vdash^n_\Phi \blacktriangleright(\hat{\tau}') :: \mathsf{ITy}$ and (5) $\blacktriangleright(\hat{\tau}')\ \mathtt{val}_{\overline{e};\Upsilon;\Phi}$. Apply the IH to (4), (5), (c), (d) and (1).

Case (abs-other-not-delegated-new). In this case, $\hat{\tau} = \mathsf{trans}(\mathsf{other}[m;\kappa]\langle(\sigma_{\text{data}}, \blacktriangleright(\hat{\tau}'))\rangle)$ and $\tau = \alpha$ and $\mathcal{D}_2 = \mathcal{D}', \mathsf{other}[m;\kappa]\langle(\sigma_{\text{data}}, \blacktriangleright(\hat{\tau}'))\rangle \leftrightarrow [\delta]\tau'/\alpha$ and

1. $c \neq \mathsf{other}[m;\kappa]$
2. $\mathsf{other}[m;\kappa]\langle(\sigma_{\text{data}}, \blacktriangleright(\hat{\tau}'))\rangle \notin \text{dom}(\mathcal{D})$
3. $\hat{\tau}' \parallel \mathcal{D}_1 \looparrowright^c_\Phi \tau' \parallel \mathcal{D}'$.
4. $\mathcal{D}' \rightsquigarrow \delta : \Delta$
5. $\Delta \vdash \tau'$

By Lemma 17 iterated on (a) and (b), we have (6) $\mathsf{other}[m;\kappa]\langle(\sigma_{\text{data}}, \blacktriangleright(\hat{\tau}'))\rangle\ \mathtt{type}_\Phi$ and (7) $\emptyset\ \emptyset \vdash^n_\Phi \blacktriangleright(\hat{\tau}') :: \mathsf{ITy}$ and (8) $\blacktriangleright(\hat{\tau}')\ \mathtt{val}_{\overline{e};\Upsilon;\Phi}$. Applying the IH to (7), (8), (c), (d) and (3), we have (9) $\vdash^c_\Phi \mathcal{D}'$ and (10) $\mathcal{D}' \rightsquigarrow \delta_1\delta' : \Delta_1\Delta'$ and (11) $\emptyset \vdash \delta_1\delta' : \Delta_1\Delta'$.

Applying (ty-store-int-ext) to (4) we have (12) $\mathcal{D}_2 \rightsquigarrow (\delta_1\delta', [\delta_1\delta']\tau/\alpha) : \Delta_1\Delta', \alpha$, i.e. (ii). By (5) and Weakening, we have (iv).

By definition, (13) $[\delta_1\delta', [\delta_1\delta']\tau'/\alpha]\alpha = [\delta_1\delta']\tau'$.

Applying Lemma 33 on (a-e), we have its conclusion as (14).

Applying (ty-trans) to (6), (14) and (12), we have (15) $\vdash_\Phi \mathsf{other}[m;\kappa]\langle(\sigma_{\text{data}}, \blacktriangleright(\hat{\tau}'))\rangle\ \mathtt{type} \rightsquigarrow [\delta]\tau'$.

By Assumption 9 on (5) and (iii), we have (16) $\emptyset \vdash [\delta_1\delta']\tau'$.

Applying (tyts-ok-ext) to (6), (1), (15) and (16), we have (i). By (tysub-ext) on (11) and (16), we have (iii).

Case (abs-ty-stored). Apply 32 to (d).

Cases (abs-i-alpha), (abs-i-t), (abs-i-unit). Apply (d).

Cases (abs-i-parr), (abs-i-forall), (abs-i-mu), (abs-i-prod), (abs-i-sum). These cases follow by iterated application of the IH and Lemma 17.8. □

The induction in this case is justified for cases (-not-delegated-new) because the number of sub-terms of the form $c\langle\sigma\rangle$ is strictly decreasing by kinding.

4.6 Translation Validation

4.6.1 Type and Typing Context Translation Validation

Lemma 35 (Type Translation)**.** *If (a)* $\vdash \Phi$ *and (b)* $\vdash_\Phi \sigma \; \texttt{type} \leadsto \tau$ *then* $\emptyset \vdash \tau$*.*

Proof. Rule induction on (b). There is only one rule, so we have $\tau = [\delta]\tau'$

(1) $\sigma \; \texttt{type}_\Phi$

(2) $\vdash^c_\Phi \mathcal{D}$

(3) $\mathsf{trans}(\sigma) \parallel \mathcal{D} \looparrowright^c_\Phi \tau' \parallel \mathcal{D}'$

(4) $\mathcal{D}' \leadsto \delta : \Delta$

Taking $\mathcal{D} = \emptyset$ and applying Lemma 34, we have $\Delta \vdash \tau'$ and $\emptyset \vdash \delta : \Delta$. Applying Assumption 5, we have our conclusion. □

Lemma 36 (Typing Context Translation)**.** *If (a)* $\vdash \Phi$ *and (b)* $\vdash_\Phi \Upsilon \; \texttt{ctx} \leadsto \Gamma$ *then* $\emptyset \vdash \Gamma$*.*

Proof. Rule induction on the assumption.

Case (ctx-trans-emp). We have $\emptyset \vdash \Gamma$ by (ictx-emp).

Case (ctx-trans-ext). We have that $\Upsilon = \Upsilon', x : \sigma$ and $\Gamma = \Gamma', x : \tau$ and (1) $\vdash_\Phi \Upsilon' \; \texttt{ctx} \leadsto \Gamma$ and (2) $\vdash_\Phi \sigma \; \texttt{type} \leadsto \tau$.

Applying the IH to (a) and (1), we have (3) $\emptyset \vdash \Gamma'$.

Applying Lemma 35 to (a) and (2) we have (4) $\emptyset \vdash \tau$. Applying (ictx-ext) to (3) and (4), we have $\emptyset \vdash \Gamma', x : \tau$. □

Lemma 37 (Typing Context Translation Membership)**.** *If (a)* $\vdash \Phi$ *and (b)* $\vdash_\Phi \Upsilon \; \texttt{ctx} \leadsto \Gamma$ *and (c)* $x : c\langle\sigma\rangle \in \Upsilon$ *then (i)* $\vdash_\Phi \sigma \; \texttt{type} \leadsto \tau$ *and (ii)* $x : \tau \in \Gamma$*.*

Proof. Rule induction on (b) and (c).

Case (ctx-trans-emp). Does not apply by (c).

Case (ctx-trans-ext). We have that $\Upsilon = \Upsilon', x' : \sigma'$ and (1) $\vdash_\Phi \Upsilon' \; \texttt{ctx} \leadsto \Gamma'$. We have two cases by (c):

> **Case** $x = x'$. We have our conclusion as the second premise.
>
> **Case** $x \neq x'$ and (2) $x : c\langle\sigma\rangle \in \Upsilon'$. Applying the IH to (1) and (2), we have our conclusion.

□

4.6.2 Selectively Abstracted Term Translations

The rules for selectively abstracted term translations are specified in Figures 41 and 42. The auxiliary judgement for validating a type and term translation store together is $\vdash^{c}_{\overline{e};\Upsilon;\Phi} \mathcal{D}\,\mathcal{G}$, specified in Figure 39. The internalization judgement $\mathcal{G} \leadsto \gamma : \Gamma$ is specified in Figure 40.

Lemma 38. *If* $\vdash^{c}_{\overline{e};\Upsilon;\Phi} \mathcal{D}\,\mathcal{G}$ *then* $\vdash^{c}_{\Phi} \mathcal{D}$.

Proof. The conclusion is a premise of each rule for the assumption. □

Lemma 39 (Argument Translation)**.** *If (a)* $\vdash^{c}_{\overline{e};\Upsilon;\Phi} \mathcal{D}\,\mathcal{G}$ *and (b)* $n : \sigma \leadsto \iota/x : \tau$ *then*

(i) $\vdash^{c}_{\Phi} \mathcal{D}$

(ii) $\mathsf{trans}(\sigma) \parallel \mathcal{D} \hookrightarrow^{c}_{\Phi} \tau \parallel \mathcal{D}$

(iii) $\mathcal{D} \leadsto \delta : \Delta$

(iv) $\vdash_{\Phi} \sigma\ \mathtt{type} \leadsto [\delta]\tau$

(v) $\mathsf{nth}[n](\overline{e}) = e$

(vi) $\Upsilon \vdash_{\Phi} e \Leftarrow \sigma \leadsto \iota$

Proof. Rule induction on (a) and (b).

Case (tmts-ok-emp). Does not apply by (b).

Case (tmts-ok-ext). We have $\mathcal{G} = \mathcal{G}', n' : \sigma' \leadsto \iota'/x' : \tau'$ There are two cases by (b).

Case $n = n'$. The conclusions are the premises.

Case $n \neq n'$. Apply the IH.

□

Lemma 40 (Selectively Abstracted Term Translation)**.** *If (a)* $\emptyset\ \emptyset \vdash^{n}_{\Phi} \triangleright(\hat{\iota}) :: \mathsf{ITm}$ *and (b)* $\triangleright(\hat{\iota})\ \mathtt{val}_{\overline{e};\Upsilon;\Phi}$ *and (c)* $|\overline{e}| = n$ *and (d)* $\vdash \Phi$ *and (e)* $\vdash_{\Phi} \Upsilon\ \mathtt{ctx} \leadsto \Gamma$ *and (f)* $\vdash^{c}_{\overline{e};\Upsilon;\Phi} \mathcal{D}\,\mathcal{G}$ *and (g)* $\hat{\iota} \parallel \mathcal{D}\,\mathcal{G} \hookrightarrow^{c}_{\overline{e};\Upsilon;\Phi} \iota \parallel \mathcal{D}'\,\mathcal{G}'$ *then* $\vdash^{c}_{\overline{e};\Upsilon;\Phi} \mathcal{D}'\,\mathcal{G}'$.

Proof. Rule induction on (g).

Case (abs-anatrans-new). In this case, $\hat{\iota} = \mathsf{anatrans}[n'](\sigma)$ and $\iota = x$ and $\mathcal{G}' = \mathcal{G}, n' : \sigma \leadsto \iota/x : \tau$ and

1. $n' \notin \text{dom}(\mathcal{G})$
2. $\mathsf{nth}[n'](\overline{e}) = e$
3. $\Upsilon \vdash_{\Phi} e \Leftarrow \sigma \leadsto \iota$
4. $\mathsf{trans}(\sigma) \parallel \mathcal{D} \hookrightarrow^{c}_{\Phi} \tau \parallel \mathcal{D}'$

By Lemma 38 on (f), we have (5) $\vdash^c_\Phi \mathcal{D}$.

By iterated application of Lemma 17, we have (6) $\emptyset\ \emptyset \vdash^n_\Phi \blacktriangleright(\mathsf{trans}(\sigma)) :: \mathsf{ITy}$ and (7) $\blacktriangleright(\mathsf{trans}(\sigma))\ \mathtt{val}_{\overline{e};\Upsilon;\Phi}$ and (8) $\sigma\ \mathtt{type}_\Phi$.

By Lemma 34 on (6), (7), (d), (5) and (4) we have (9) $\vdash^c_\Phi \mathcal{D}'$ and (10) $\mathcal{D}' \leadsto \delta : \Delta$.

By Lemma 33 on (6), (7), (d) and (4) we have (11) $\mathsf{trans}(\sigma) \parallel \emptyset \looparrowright^c_\Phi \tau \parallel \mathcal{D}'$ and (12) $\mathsf{trans}(\sigma) \parallel \mathcal{D}' \looparrowright^c_\Phi \tau \parallel \mathcal{D}'$.

By (ty-trans) on (8), (11) and (10), we have (13) $\vdash_\Phi \sigma\ \mathtt{type} \leadsto [\delta]\tau$.

Apply (tmts-ok-ext) to (5), (7), (10), (13), (2) and (3).

Case (abs-anatrans-stored). We have $\mathcal{D}' = \mathcal{D}$ and $\mathcal{G}' = \mathcal{G}$. Apply (e).

Case (abs-syntrans-new). In this case, $\hat{\iota} = \mathsf{syntrans}[n']$ and $\iota = x$ and $\mathcal{G}' = \mathcal{G}, n' : \sigma \leadsto \iota/x : \tau$ and

1. $n' \notin \mathrm{dom}(\mathcal{G})$
2. $\mathsf{nth}[n'](\overline{e}) = e$
3. $\Upsilon \vdash_\Phi e \Rightarrow \sigma \leadsto \iota$
4. $\mathsf{trans}(\sigma) \parallel \mathcal{D} \looparrowright^c_\Phi \tau \parallel \mathcal{D}'$

By Lemma 38 on (f), we have (5) $\vdash^c_\Phi \mathcal{D}$.

By Lemma 3 and kinding rules, we have (6) $\emptyset\ \emptyset \vdash^n_\Phi \blacktriangleright(\mathsf{trans}(\sigma)) :: \mathsf{ITy}$ and (7) $\blacktriangleright(\mathsf{trans}(\sigma))\ \mathtt{val}_{\overline{e};\Upsilon;\Phi}$ and (8) $\sigma\ \mathtt{type}_\Phi$.

By Lemma 34 on (6), (7), (d), (5) and (4) we have (9) $\vdash^c_\Phi \mathcal{D}'$ and (10) $\mathcal{D}' \leadsto \delta : \Delta$.

By Lemma 33 on (6), (7), (d) and (4) we have (11) $\mathsf{trans}(\sigma) \parallel \emptyset \looparrowright^c_\Phi \tau \parallel \mathcal{D}'$ and (12) $\mathsf{trans}(\sigma) \parallel \mathcal{D}' \looparrowright^c_\Phi \tau \parallel \mathcal{D}'$.

By (ty-trans) on (8), (11) and (10), we have (13) $\vdash_\Phi \sigma\ \mathtt{type} \leadsto [\delta]\tau$.

Apply (tmts-ok-ext) to (5), (7), (10), (13), (2) and (3).

Case (abs-syntrans-stored). We have $\mathcal{D}' = \mathcal{D}$ and $\mathcal{G}' = \mathcal{G}$. Apply (e).

Cases (abs-x), (abs-triv). We have $\mathcal{D}' = \mathcal{D}$ and $\mathcal{G}' = \mathcal{G}$. Apply (e).

Cases (all remaining cases). These cases follow by iterative application of the IH and Lemma 17.9.

□

4.6.3 Translation Validation

The translation validation judgement is specified in Figure 43 and discussed together with type-preserving translation below.

4.7 Type-Preserving Translation and Type Safety

Theorem 4 (Type-Preserving Translation)**.**

1) If (a) $\vdash \Phi$ *and (b)* $\vdash_\Phi \Upsilon\ \mathtt{ctx} \leadsto \Gamma$ *and (c)* $\Upsilon \vdash_\Phi e \Rightarrow \sigma \leadsto \iota$ *then (i)* $\vdash_\Phi \sigma\ \mathtt{type} \leadsto \tau$ *and (ii)* $\emptyset\,\Gamma \vdash \iota : \tau$*.*

2) *If (a)* $\vdash \Phi$ *and (b)* $\vdash_\Phi \Upsilon \texttt{ ctx} \rightsquigarrow \Gamma$ *and (c)* $\sigma \texttt{ type}_\Phi$ *and (d)* $\Upsilon \vdash_\Phi e \Leftarrow \sigma \rightsquigarrow \iota$ *then (i)* $\vdash_\Phi \sigma \texttt{ type} \rightsquigarrow \tau$ *and (ii)* $\emptyset\, \Gamma \vdash \iota : \tau$.

Proof. **Case** (subsume). In this case of 2), we have that (1) $\Upsilon \vdash_\Phi e \Rightarrow \sigma \rightsquigarrow \iota$. Apply the IH to (a), (b) and (1).

Case (ascribe). In this case of 1), we have that $e = e' : \sigma'$ and (1) $\emptyset\, \emptyset \vdash^0_\Phi \sigma' :: \mathsf{Ty}$ and (2) $\sigma' \Downarrow_{\cdot;\emptyset;\Phi} \sigma$ and (3) $\Upsilon \vdash_\Phi e \Leftarrow \sigma \rightsquigarrow \iota$.

Applying Definition 2 to (2), we have (4) $\sigma \texttt{ val}_{\cdot;\emptyset;\Phi}$. Applying Corollary 1 to (1) and (2), we have (5) $\emptyset\, \emptyset \vdash^0_\Phi \sigma :: \mathsf{Ty}$. Applying Definition 1 to (4) and (5), we have (6) $\sigma \texttt{ type}_\Phi$. Apply the IH to (a), (b), (6) and (3).

Case (syn-var). In this case of 1), We have $e = x$ and $\iota = x$ and (1) $x : \sigma \in \Upsilon$. Applying Lemma 37 to (a) and (b) and (1), we have (i) and (2) $x : \tau \in \Gamma$. Applying the standard typing variable rule for the IL to (2), we have $\emptyset\Gamma \vdash x : \tau$.

Case (ana-fix). In this case of 2), we have $e = \mathsf{fix}(x.e')$ and $\iota = \mathsf{fix}[\tau](x.\iota')$ and (1) $\Upsilon, x : \sigma \vdash_{e'} \sigma \Leftarrow \iota' \rightsquigarrow$ and (2) $\vdash_\Phi \sigma \texttt{ type} \rightsquigarrow \tau$, i.e. (i).

Applying (ctx-trans-ext) to (b) and (2), we have (3) $\vdash_\Phi \Upsilon, x : \sigma \texttt{ ctx} \rightsquigarrow \Gamma, x : \tau$.

Applying Lemma 30 to (2), we have (4) $\sigma \texttt{ type}_\Phi$.

Applying the IH to (a), (3), (4) and (1), we have (5) $\emptyset\, \Gamma, x : \tau \vdash \iota' : \tau$.

Applying Lemma 35 to (2), we have (6) $\emptyset \vdash \tau$.

By the standard typing rule for fixpoints in the IL on (6) and (5), we have (i) $\emptyset\, \Gamma, x : \tau \vdash \mathsf{fix}[\tau](x.\iota')$.

Case (syn-lam). In this case of 1), we have $e = \lambda x{:}\sigma_1.e'$ and $\sigma = \rightharpoonup\langle(\sigma_1, \sigma_2)\rangle$ and $\iota = \lambda x{:}\tau_1.\iota'$ and (1) $\emptyset\, \emptyset \vdash^0_\Phi \sigma_1 :: \mathsf{Ty}$ and (2) $\sigma_1 \Downarrow_{\cdot;\emptyset;\Phi} \sigma'_1$ and (3) $\Upsilon, x : \sigma'_1 \vdash_{e'} \sigma_2 \Rightarrow \iota' \rightsquigarrow$ and (4) $\vdash_\Phi \sigma'_1 \texttt{ type} \rightsquigarrow \tau_1$.

Applying Definition 2 to (2), we have (5) $\sigma'_1 \texttt{ val}_{\cdot;\emptyset;\Phi}$. Applying Corollary 1 to (1) and (2), we have (6) $\emptyset\, \emptyset \vdash^0_\Phi \sigma'_1 :: \mathsf{Ty}$. Applying Definition 1 to (5) and (6), we have (7) $\sigma'_1 \texttt{ type}_\Phi$.

Applying (ctx-trans-ext) to (b) and (4), we have (8) $\vdash_\Phi \Upsilon, x : \sigma'_1 \texttt{ ctx} \rightsquigarrow \Gamma, x : \tau_1$.

Applying the IH to (a), (8) and (3), we have (9) $\vdash_\Phi \sigma_2 \texttt{ type} \rightsquigarrow \tau_2$ and (10) $\emptyset\, \Gamma, x : \tau_1 \vdash \iota' : \tau_2$.

By Lemma 34 on (9), we have (11) $\emptyset \vdash \tau_2$.

By the standard typing rule for lambdas in the IL on (10) and (9), we have (ii).

Case (syn-ap). In this case of 1), we have $e = e_1(e_2)$ and $\iota = \iota_1(\iota_2)$ and (1) $\Upsilon \vdash_\Phi e_1 \Rightarrow \rightharpoonup\langle(\sigma_1, \sigma_2)\rangle \rightsquigarrow \iota_1$ and (2) $\Upsilon \vdash_\Phi e_2 \Leftarrow \sigma_1 \rightsquigarrow \iota_2$.

Applying the IH to (a), (b) and (1), we have (3) $\vdash_\Phi \sigma_1 \texttt{ type} \rightsquigarrow \tau_1$ and (4) $\emptyset\, \Gamma \vdash \iota_1 : \tau_1$.

Applying the IH to (a), (b) and (2), we have (5) $\vdash_\Phi \sigma_2 \texttt{ type} \rightsquigarrow \tau_2$ and (6) $\emptyset\, \Gamma \vdash \iota_2 : \tau_2$.

By the standard typing rule for application in the IL on (4) and (6), we have (ii).

Case (ana-intro). In this case of 2), we have $e = \mathsf{intro}[\sigma_{\text{tmidx}}](\overline{e})$ and $\sigma = \textsc{tc}\langle\sigma_{\text{tyidx}}\rangle$ and

(1) $\mathsf{tycon}\ \textsc{tc}\ \{\mathsf{trans} = \sigma_{\text{schema}}\ \mathsf{in}\ \omega\} \sim \mathsf{tcsig}[\kappa_{\text{tyidx}}]\ \{\chi\} \in \Phi$

(2) $\mathsf{intro}[\kappa_{\text{tmidx}}] \in \chi$

(3) $\emptyset\, \emptyset \vdash^0_\Phi \sigma_{\text{tmidx}} :: \kappa_{\text{tmidx}}$

(4) $\mathsf{ana\ intro} = \sigma_{\text{def}} \in \omega$

(5) $|\overline{e}| = n$

(6) $\mathsf{args}(n) = \sigma_{\text{args}}$

(7) $\sigma_{\text{def}}(\sigma_{\text{tyidx}})(\sigma_{\text{tmidx}})(\sigma_{\text{args}}) \Downarrow_{\overline{e};\Upsilon;\Phi} \rhd(\hat{\iota})$

(8) $\vdash^{\textsc{tc}}_{\overline{e};\Upsilon;\Phi} \hat{\iota} : \textsc{tc}\langle\sigma_{\text{tyidx}}\rangle \rightsquigarrow \iota$

Apply Lemma 1 to (1), we have that there exists a prefix Φ' of Φ such that (9) $\vdash_{\Phi',\mathsf{tycon}~\textsc{tc}~\{\mathsf{trans}=\sigma_{\text{schema}}~\mathsf{in}~\omega\}\sim\mathsf{tcsig}[\kappa_{\text{tyidx}}]\,\{\chi\}} \omega \sim \psi$ and (10) $\vdash \Phi'$.

Apply Lemma 3 to (10), (9) and (a), we have (11) $\vdash_\Phi \omega \sim \psi$

Applying Lemma 2 to (11) and (4), we have (12) $\emptyset\,\emptyset \vdash^0_\Phi \sigma_{\text{def}} :: \kappa_{\text{tyidx}} \to \kappa_{\text{tmidx}} \to \mathsf{List}[\mathsf{Arg}] \to \mathsf{ITm}$.

Applying Lemma 14 to (12), we have (13) $\emptyset\,\emptyset \vdash^0_\Phi \sigma_{\text{def}} :: \kappa_{\text{tyidx}} \to \kappa_{\text{tmidx}} \to \mathsf{List}[\mathsf{Arg}] \to \mathsf{ITm}$.

Applying Lemma 17 on (c), we have that (14) $\emptyset\,\emptyset \vdash^n_\Phi \sigma_{\text{tyidx}} :: \kappa_{\text{tyidx}}$.

Apply Lemma 28, we have that (15) $\emptyset\,\emptyset \vdash^n_\Phi \sigma_{\text{args}} :: \mathsf{List}[\mathsf{Arg}]$.

Applying (k-ap) three times with (13), (14), (3) and (15), we have that (16) $\emptyset\,\emptyset \vdash^n_\Phi \sigma_{\text{def}}(\sigma_{\text{tyidx}})(\sigma_{\text{tmidx}})(\sigma_{\text{args}}) :: \mathsf{ITm}$.

Applying Lemma 1 to (16) and (7), we have (17) $\emptyset\,\emptyset \vdash^n_\Phi \rhd(\hat{\iota}) :: \mathsf{ITm}$.

Applying Definition 2 to (7), we have (18) $\rhd(\hat{\iota})~\mathtt{val}_{\overline{e};\Upsilon;\Phi}$.

Apply Lemma 41 to (a), (b), (c), (17), (18), (5) and (7).

Case (syn-targ). In this case of 1), we have $e = \mathsf{targop}[\mathbf{op}; \sigma_{\text{tmidx}}](e_{\text{targ}}; \overline{e})$ and

(1) $\Upsilon \vdash_\Phi e_{\text{targ}} \Rightarrow \textsc{tc}\langle\sigma_{\text{tyidx}}\rangle \rightsquigarrow \iota_{\text{targ}}$ and

(2) $\mathsf{tycon}~\textsc{tc}~\{\mathsf{trans} = \sigma_{\text{schema}}~\mathsf{in}~\omega\} \sim \mathsf{tcsig}[\kappa_{\text{tyidx}}]\,\{\chi\} \in \Phi$ and

(3) $\mathbf{op}[\kappa_{\text{tmidx}}] \in \chi$ and

(4) $\emptyset\,\emptyset \vdash^0_\Phi \sigma_{\text{tmidx}} :: \kappa_{\text{tmidx}}$ and

(5) $\mathsf{syn}~\mathbf{op} = \sigma_{\text{def}} \in \omega$ and

(6) $|e_{\text{targ}}; \overline{e}| = n$ and

(7) $\mathsf{args}(n) = \sigma_{\text{args}}$ and

(8) $\sigma_{\text{def}}(\sigma_{\text{tyidx}})(\sigma_{\text{tmidx}})(\sigma_{\text{args}}) \Downarrow_{(e_{\text{targ}};\overline{e});\Upsilon;\Phi} (\sigma, \rhd(\hat{\iota}))$.

Applying Lemma 3 to (a), (b) and (1), we have (9) $\textsc{tc}\langle\sigma_{\text{tyidx}}\rangle~\mathtt{type}_\Phi$. By Lemma 17.7 on (9), we have (10) $\emptyset\,\emptyset \vdash^0_\Phi \sigma_{\text{tyidx}} :: \kappa_{\text{tyidx}}$. Applying Lemma 14 to (10), we have (11) $\emptyset\,\emptyset \vdash^n_\Phi \sigma_{\text{tyidx}} :: \kappa_{\text{tyidx}}$.

Applying Lemma 1 to (2), we have that there is a prefix Φ' of Φ such that (12) $\vdash \Phi'$ and (13) $\vdash \Phi', \mathsf{tycon}~\textsc{tc}~\{\mathsf{trans} = \sigma_{\text{schema}}~\mathsf{in}~\omega\} \sim \mathsf{tcsig}[\kappa_{\text{tyidx}}]\,\{\chi\}$ and (14) $\vdash_{\Phi',\mathsf{tycon}~\textsc{tc}~\{\mathsf{trans}=\sigma_{\text{schema}}~\mathsf{in}~\omega\}\sim\mathsf{tcsig}[\kappa_{\text{tyidx}}]\,\{\chi\}} \omega \sim \mathsf{tcsig}[\kappa_{\text{tyidx}}]\,\{\chi\}$.

Applying Lemma 6 to (12), we have (15) $\vdash \Phi', \mathsf{tycon}~\textsc{tc}~\{\mathsf{trans} = \sigma_{\text{schema}}~\mathsf{in}~\omega\} \sim \mathsf{tcsig}[\kappa_{\text{tyidx}}]\,\{\chi\}~\mathtt{sigsok}$. Applying Lemma 6 to (a), taking by the definition of prefixing that $\Phi = (\Phi', \mathsf{tycon}~\textsc{tc}~\{\mathsf{trans} = \sigma_{\text{schema}}~\mathsf{in}~\omega\} \sim \mathsf{tcsig}[\kappa_{\text{tyidx}}]\,\{\chi\})\Phi''$, we have (16) $\vdash (\Phi', \mathsf{tycon}~\textsc{tc}~\{\mathsf{trans} =$

$\sigma_{\text{schema}}\ \mathsf{in}\ \omega\} \sim \mathsf{tcsig}[\kappa_{\text{tyidx}}]\ \{\chi\})\Phi''\ \mathtt{sigsok}$. Applying Lemma 23 to (15), (14) and (16), we have that (17) $\vdash_\Phi \omega \sim \mathsf{tcsig}[\kappa_{\text{tyidx}}]\ \{\chi\}$.

Applying Lemma 3 to (17) and (5), we have that (18) $\emptyset\ \emptyset \vdash^0_\Phi \sigma_{\text{def}} :: \kappa_{\text{tyidx}} \to \kappa_{\text{tmidx}} \to \mathsf{List}[\mathsf{Arg}] \to (\mathsf{Ty} \times \mathsf{ITm})$. Applying Lemma 14 to (18), we have (19) $\emptyset\ \emptyset \vdash^n_\Phi \sigma_{\text{def}} :: \kappa_{\text{tyidx}} \to \kappa_{\text{tmidx}} \to \mathsf{List}[\mathsf{Arg}] \to (\mathsf{Ty} \times \mathsf{ITm})$.

Applying Lemma 28 to (7), we have that (20) $\emptyset\ \emptyset \vdash^n_\Phi \sigma_{\text{args}} :: \mathsf{List}[\mathsf{Arg}]$.

Applying (k-ap) three times with (19), (11), (4) and (20), we have that (21) $\emptyset\ \emptyset \vdash^n_\Phi \sigma_{\text{def}}(\sigma_{\text{tyidx}})(\sigma_{\text{tmidx}})(\sigma_{\text{args}}) :: \mathsf{Ty} \times \mathsf{ITm}$.

Applying Corollary 1 to (8), (21), (a) and (b), we have that (22) $\emptyset\ \emptyset \vdash^n_\Phi (\sigma, \rhd(\hat{\iota})) :: \mathsf{Ty} \times \mathsf{ITm}$. By Definition 2 on (8), we have (23) $(\sigma, \rhd(\hat{\iota}))\ \mathtt{val}_{(e_{\text{targ}};\overline{e});\Upsilon;\Phi}$.

Applying Lemma 17 to (23) and (22), we have that (24) $\emptyset\ \emptyset \vdash^n_\Phi \sigma :: \mathsf{Ty}$ and (25) $\sigma\ \mathtt{val}_{(e_{\text{targ}};\overline{e});\Upsilon;\Phi}$ and (26) $\emptyset\ \emptyset \vdash^n_\Phi \rhd(\hat{\iota}) :: \mathsf{ITm}$ and (27) $\rhd(\hat{\iota})\ \mathtt{val}_{(e_{\text{targ}};\overline{e});\Upsilon;\Phi}$. Applying Definition 1 to (24) and (25) we have (28) $\sigma\ \mathtt{type}_\Phi$.

Apply Lemma 41 to (a), (b), (28), (26), (27), (6) and (8).

Case (ana-intro-other). In this case, we have that $e = \mathsf{intro}[\rhd(\hat{\iota})](\overline{e})$ and $\sigma = \mathsf{other}[m;\kappa]\langle\sigma_{\text{tyidx}}\rangle$ and

(1) $|\overline{e}| = n$

(2) $\emptyset\ \emptyset \vdash^n_\Phi \rhd(\hat{\iota}) :: \mathsf{ITm}$

(3) $\rhd(\hat{\iota})\ \mathtt{val}_{\overline{e};\Upsilon;\Phi}$

(4) $\vdash^{\mathsf{other}[m;\kappa]}_{\overline{e};\Upsilon;\Phi} \hat{\iota} : \mathsf{other}[m;\kappa]\langle\sigma_{\text{tyidx}}\rangle \rightsquigarrow \iota$

Apply Lemma 41 to (a), (b), (c), (2), (3), (1) and (4).

Case (syn-targ-other). In this case, we have that $e = \mathsf{targop}[\mathbf{op}; \rhd(\hat{\iota})](e_{\text{targ}}; \overline{e})$ and

(1) $\Upsilon \vdash_\Phi e_{\text{targ}} \Rightarrow \mathsf{other}[m;\kappa]\langle\sigma_{\text{tyidx}}\rangle \rightsquigarrow \iota_{\text{targ}}$

(2) $|e_{\text{targ}}; \overline{e}| = n$

(3) $\emptyset\ \emptyset \vdash^n_\Phi \rhd(\hat{\iota}) :: \mathsf{ITm}$

(4) $\rhd(\hat{\iota})\ \mathtt{val}_{(e_{\text{targ}};\overline{e});\Upsilon;\Phi}$

(5) $\sigma\ \mathtt{type}_\Phi$

(6) $\vdash^{\mathsf{other}[m;\kappa]}_{(e_{\text{targ}};\overline{e});\Upsilon;\Phi} \hat{\iota} : \sigma \rightsquigarrow \iota$

Apply Lemma 41 to (a), (b), (9), (3), (4), (2), (5) and (6). □

Lemma 41 (Translation Validation)**.** *If (a)* $\vdash \Phi$ *and (b)* $\vdash_\Phi \Upsilon\ \mathtt{ctx} \rightsquigarrow \Gamma$ *and (c)* $\sigma\ \mathtt{type}_\Phi$ *and (d)* $\emptyset\ \emptyset \vdash^n_\Phi \rhd(\hat{\iota}) :: \mathsf{ITm}$ *and (e)* $\rhd(\hat{\iota})\ \mathtt{val}_{\overline{e};\Upsilon;\Phi}$ *and (f)* $|\overline{e}| = n$ *and (g)* $\vdash^c_{\overline{e};\Upsilon;\Phi} \hat{\iota} : \sigma \rightsquigarrow \iota$ *then (i)* $\vdash_\Phi \sigma\ \mathtt{type} \rightsquigarrow \tau$ *and (ii)* $\emptyset\ \Gamma \vdash \iota : \tau$.

Proof. Rule induction on (g). There is one rule, so we have

(1) $\mathsf{trans}(\sigma) \parallel \emptyset \hookrightarrow^{c}_{\Phi} \tau_{\text{abs}} \parallel \mathcal{D}$

(2) $\hat{\iota} \parallel \mathcal{D}\ \emptyset \hookrightarrow^{c}_{\overline{e};\Upsilon;\Phi} \iota_{\text{abs}} \parallel \mathcal{D}'\ \mathcal{G}$

(3) $\mathcal{D}' \leadsto \delta : \Delta_{\text{abs}}$

(4) $\mathcal{G} \leadsto \gamma : \Gamma_{\text{abs}}$

(5) $\Delta_{\text{abs}}\ \Gamma_{\text{abs}} \vdash \iota_{\text{abs}} : \tau_{\text{abs}}$

Applying Lemma 34 on (1), we have that (6) $\vdash^{c}_{\Phi} \mathcal{D}$.
Applying Lemma 40 to (6), we have (7) $\vdash^{c}_{\overline{e};\Upsilon;\Phi} \mathcal{D}\ \mathcal{G}$.
Applying Lemma 42, we have that (8) $\mathcal{D} \leadsto \delta : \Delta_{\text{abs}}$ and (9) $\mathcal{G} \leadsto \gamma : \Gamma_{\text{abs}}$ and (10) $\emptyset \vdash \delta : \Delta_{\text{abs}}$ and (11) $\Delta\ \Gamma \vdash \gamma : \Gamma_{\text{abs}}$.
By Assumptions 4 and 7 and (5) and (10) and (11), we have that $\emptyset\ \Gamma \vdash [\delta][\gamma]\iota_{\text{abs}} : \tau_{\text{abs}}$. □

Lemma 42. *If (a) $\vdash^{c}_{\overline{e};\Upsilon;\Phi} \mathcal{D}\ \mathcal{G}$ and (b) $\vdash_{\Phi} \Upsilon\ \mathtt{ctx} \leadsto \Gamma$ and (c) $\mathcal{D} \leadsto \delta : \Delta_{abs}$ and (d) $\mathcal{G} \leadsto \gamma : \Gamma_{abs}$ then (i) $\emptyset \vdash \delta : \Delta$ and (ii) $\Delta\ \Gamma \vdash \gamma : \Gamma_{abs}$.*

Proof. Inversion on (a).
(i) follows from Lemma 32. (ii) follows by inductive appeal to Theorem 4, justified by the same metric as Theorem 3. □

4.8 Conservativity

4.8.1 Backprojection

To avoid having to give a large number of rules that do nothing interesting, we define backprojection on static terms definitionally. In the discussion below, let $\Phi' = \Phi, \mathsf{tycon}\ \textsc{tc}'\ \{\mathsf{trans} = \sigma_{\text{schema}}\ \mathsf{in}\ \omega\} \sim \mathsf{tcsig}[\kappa_{\text{tyidx}}]\ \{\chi\}\sigma\kappa$.

Definition 5 (Backprojection (Static values))**.** *We write ${}^{m}\mathcal{B}^{\textsc{tc}'}_{\Phi'}(\sigma) = \sigma'$ for the backprojection of σ when $\emptyset\ \emptyset \vdash^{n}_{\Phi'} \sigma :: \kappa$ and $\sigma\ \mathtt{val}_{\mathcal{A}}$ and $\sigma_{schema}(\sigma_{tyidx}) \Downarrow_{\cdot;\emptyset;\Phi'} \blacktriangleright(\hat{\tau})$. We generate σ' by replacing all sub-terms of the form $\textsc{tc}'\langle\sigma_{tyidx}\rangle$ with a sub-term of the form $\mathsf{other}[m; \kappa_{tyidx}]\langle(\sigma_{tyidx}, \blacktriangleright(\hat{\tau}))\rangle$.*

All interesting properties about static values (in particular, that they are values and their kind) are preserved under backprojection.

Lemma 43. *If $\emptyset\ \emptyset \vdash^{n}_{\Phi'} \sigma :: \kappa$ and $\sigma\ \mathtt{val}_{\mathcal{A}}$ and ${}^{m}\mathcal{B}^{\textsc{tc}'}_{\Phi'}(\sigma) = \sigma'$ then $\emptyset\ \emptyset \vdash^{n}_{\Phi} \sigma' :: \kappa$ and $\sigma'\ \mathtt{val}_{\mathcal{A}}$.*

For types, backprojection preserves the type translation (essentially trivially, since the translational internal type is contained directly in the index) and for all types other than those constructed by $\textsc{tc}$, their tycons.

Lemma 44. *If $\emptyset\ \emptyset \vdash^{n}_{\Phi'} \sigma :: \mathsf{Ty}$ and $\sigma\ \mathtt{val}_{\mathcal{A}}$ and $\vdash_{\Phi'} \sigma\ \mathtt{type} \leadsto \tau$ and ${}^{m}\mathcal{B}^{\textsc{tc}'}_{\Phi'}(\sigma) = \sigma'$ then $\vdash_{\Phi} \sigma\ \mathtt{type} \leadsto \tau$ and if $\sigma = c\langle\sigma_{tyidx}\rangle$ and $c \neq \textsc{tc}'$ then ${}^{m}\mathcal{B}^{\textsc{tc}'}_{\Phi'}(\sigma_{tyidx}) = \sigma'_{tyidx}$ and $\sigma' = c\langle\sigma'_{tyidx}\rangle$.*

We can map backprojection over typing contexts in the same way.

Definition 6 (Backprojection (Typing Contexts))**.** *We write* ${}^{m}\mathcal{B}^{\text{TC}'}_{\Phi'}(\Upsilon) = \Upsilon'$ *when* $\vdash_{\Phi} \Upsilon$ *for the mapping of the backprojection operation over all types in* Υ*.*

Lemma 45. *If* $\vdash_{\Phi'} \Upsilon\ \texttt{ctx} \rightsquigarrow \Gamma$ *and* ${}^{m}\mathcal{B}^{\text{TC}'}_{\Phi'}(\Upsilon) = \Upsilon'$ *then* $\vdash_{\Phi} \Upsilon\ \texttt{ctx} \rightsquigarrow \Gamma$*.*

Definition 7 (Modular Propositions)**.** *We say a proposition* $P(\Gamma, \sigma, \iota)$ *is* modular *under* Φ *if for all* Φ'' *such that* $\vdash \Phi\Phi''$ *and all* $\text{TC} \in dom(\Phi'')$*, we have that if* ${}^{m}\mathcal{B}^{\text{TC}'}_{\Phi\Phi''}(\sigma) = \sigma'$ *and* $P(\Gamma, \sigma', \iota)$ *then* $P(\Gamma, \sigma, \iota)$*.*

Note that for propositions constrained to only hold for $\sigma : \kappa$ such that κ does not contain Ty, the proposition is modular trivially because backprojection is the identity relation in that case. In other cases, the proposition simply must not "know" about future tycons or treat "other" tycons in a special way (e.g. counting how many there are). We expect that this will generally be straightforward to establish. Note that this is a property of the invariant, and can thus be established modularly, not a proof obligation for every composition.

Definition 8 (Backprojection (External Terms))**.** *If* $\vdash \Phi'$ *and* $\vdash_{\Phi'} \Upsilon\ \texttt{ctx} \rightsquigarrow \Gamma$ *and* $\Upsilon \vdash_{\Phi'} e \Leftarrow \sigma \rightsquigarrow \iota$ *or* $\Upsilon \vdash_{\Phi'} e \Rightarrow \sigma \rightsquigarrow \iota$ *then we define* ${}^{m}\mathcal{B}^{\text{TC}'}_{\Phi'}(e) = e'$ *by mapping the following transformation over all sub-terms* e''*:*

1. *If* $e'' = \mathsf{intro}[\sigma_{tmidx}](\overline{e})$ *such that*

   (1) $\mathsf{tycon}\ \text{TC}\ \{\mathsf{trans} = \sigma_{schema}\ \mathsf{in}\ \omega\} \sim \mathsf{tcsig}[\kappa_{tyidx}]\ \{\chi\} \in \Phi$

   (2) $\mathsf{intro}[\kappa_{tmidx}] \in \chi$

   (3) $\emptyset\ \emptyset \vdash^{0}_{\Phi} \sigma_{tmidx} :: \kappa_{tmidx}$

   (4) $\mathsf{ana\ intro} = \sigma_{def} \in \omega$

   (5) $|\overline{e}| = n$

   (6) $\mathsf{args}(n) = \sigma_{args}$

   (7) $\sigma_{def}(\sigma_{tyidx})(\sigma_{tmidx})(\sigma_{args}) \Downarrow_{\overline{e};\Upsilon;\Phi} \rhd(\hat{\iota})$

   (8) $\mathsf{trans}(\sigma) \parallel \emptyset \hookrightarrow^{c}_{\Phi} \tau_{abs} \parallel \mathcal{D}$

   (9) $\hat{\iota} \parallel \mathcal{D}\ \emptyset \hookrightarrow^{c}_{\overline{e};\Upsilon;\Phi} \iota_{abs} \parallel \mathcal{D}'\ \mathcal{G}$

   (10) $\mathcal{D}' \rightsquigarrow \delta : \Delta_{abs}$

   (11) $\mathcal{G} \rightsquigarrow \gamma : \Gamma_{abs}$

   (12) $\Delta_{abs}\ \Gamma_{abs} \vdash \iota_{abs} : \tau_{abs}$

   We generate $\mathsf{intro}[\blacktriangleright(\hat{\tau})](\overline{e}')$ *where* ${}^{m}\mathcal{B}^{\text{TC}'}_{\Phi'}(\overline{e}) = \overline{e}'$ *maps backprojection recursively over all terms in* $\overline{e}$ *for which a type was recorded in* $\mathcal{G}$*. All others can be left alone (they have no bearing on the type or translation).*

2. *If* $e'' = \mathsf{targop}[\boldsymbol{op}; \sigma_{tmidx}](e_{targ}; \overline{e})$ *the procedure is analagous.*

Comparing (ana-intro) and (ana-intro-other), and (syn-targ) and (syn-targ-other), it is straightforward to see that term translations are invariant with respect to backprojection, and types respect backprojection.

Lemma 46. *If* $\vdash \Phi'$ *and* $\vdash_{\Phi'} \Upsilon\ \texttt{ctx} \leadsto \Gamma$ *and* $\Upsilon \vdash_{\Phi'} e \Leftarrow \sigma \leadsto \iota$ *or* $\Upsilon \vdash_{\Phi'} e \Rightarrow \sigma \leadsto \iota$ *then* ${}^{m}\mathcal{B}^{\text{TC}'}_{\Phi'}(e) = e'$ *and* ${}^{m}\mathcal{B}^{\text{TC}'}_{\Phi'}(\Upsilon) = \Upsilon'$ *and* $\vdash_{\Phi} \Upsilon'\ \texttt{ctx} \leadsto \Gamma$ *and* ${}^{m}\mathcal{B}^{\text{TC}'}_{\Phi'}(\sigma) = \sigma'$ *and* $\Upsilon' \vdash_{\Phi} e \Leftarrow \sigma' \leadsto \iota$.

The end result is that we have our conservativity theorem.

Theorem 5 (Conservativity)**.** *If* $\vdash \Phi$ *and* $\text{TC} \in dom(\Phi)$ *and a tycon invariant for* TC *holds under* Φ*:*

- *For all* $\Upsilon, e, \sigma_{tyidx}$*, if* $\Upsilon \vdash_{\Phi} e \Leftarrow \text{TC}\langle\sigma_{tyidx}\rangle \leadsto \iota$ *and* $\vdash_{\Phi} \Upsilon\ \texttt{ctx} \leadsto \Gamma$ *and* $\vdash_{\Phi} \text{TC}\langle\sigma_{tyidx}\rangle \leadsto \tau$ *then* $P(\Gamma, \sigma_{tyidx}, \iota)$.

then for all $\Phi' = \Phi, \mathsf{tycon}\ \text{TC}'\ \{\theta'\} \sim \mathsf{tcsig}[\kappa']\ \{\chi'\}$ *such that* $\vdash \Phi'$*, the same tycon invariant holds under* Φ'*:*

- *For all* $\Upsilon, e, \sigma_{tyidx}$*, if* $\Upsilon \vdash_{\Phi'} e \Leftarrow \text{TC}\langle\sigma_{tyidx}\rangle \leadsto \iota$ *and* $\vdash_{\Phi'} \Upsilon \leadsto \Gamma$ *and* $\vdash_{\Phi'} \text{TC}\langle\sigma_{tyidx}\rangle \leadsto \tau$ *then* $P(\Gamma, \sigma_{tyidx}, \iota)$.

(if proposition $P(\Gamma, \sigma, \iota)$ *is* modular*)*

Proof Sketch. We simply backproject our typing derivation, apply the original tycon invariant, then apply the definition of modular propositions. □

This section has, obviously, been less detailed than the remainder of the supplement, essentially because the lemmas above involve a rather large number of cases where nothing of note is happening, and we simply ran out of time to inductively specify the full details of backprojection and prove all of them. Backprojection is a rather simple conceptual mapping, so we believe that the level of detail above is sufficiently convincing, and we plan on writing out the full details (which we anticipate will be another 20 or so pages) well before the notification date.

5 Discussion

5.1 Record Types

In the paper, we presented labeled products rather than the more conventional record types, both to emphasize that different variants have different trade-offs, and because it made our examples simpler.

One strategy for supporting records themselves would be to specify the intro operator such that records that do not have sorted indices are uninhabited, then provide a helper static function that generates only sorted indices and require that all type construction pass through that. This notion of a “type constructor constructor” could perhaps be integrated into the calculus.

An alternative strategy may have been to use an abstract kind that ensured that such type indices could not have been constructed, but to be compatible with our equality kind restriction, this would require support for abstract equality kinds, analagous to abstract equality types in SML. We chose not to formalize these for simplicity, and to demonstrate the general technique above. This could also have been used for the labeled product example.

5.2 Conjectures

Other properties that we conjecture are true but have not yet proven include decidability of typing (which requires proving strong normalization of the SL), unicity of typing (i.e. that types and translations, if they exist, are unique; this requires proving determinism of normalization), and stability (i.e. that extending the tycon context doesn't change the meaning of already well-typed terms; this is essentially trivial by inspection of the rules, and we proved stability of kinding and tycon context validity above because these were needed for other purposes, but ran out of time and space in the paper to even mention this). Note that because these have no formal development, we did not mention them as contributions in the paper.

5.3 Additional Examples

We have developed additional examples, and developed detailed proof sketches of tycon invariants with them. These will also be available in the final supplement, but due to recent changes in our calculus, and the tedium of writing "code" using TeX macros, these also didn't make the cut in the submitted supplement. There is nothing conceptually novel in these examples, and we did not claim that we had additional complete examples available upon submission in the paper.

Note that because of our simplification related to inductive kinds here, the regular string example cannot be encoded (kind Rx is an inductive sum).

internal types

$\tau \quad ::= \quad \tau \rightharpoonup \tau \mid \alpha \mid \forall(\alpha.\tau) \mid t \mid \mu(t.\tau) \mid \mathsf{1} \mid \tau \times \tau \mid \tau + \tau$

internal terms

$$\begin{array}{lll} \iota & ::= & x \mid \lambda x{:}\tau.\iota \mid \iota(\iota) \mid \mathsf{fix}[\tau](x.\iota) \mid \Lambda(\alpha.\iota) \mid \iota[\tau] \\ & \mid & \mathsf{fold}[t.\tau](\iota) \mid \mathsf{unfold}(\iota) \mid () \mid (\iota, \iota) \mid \mathsf{fst}(\iota) \mid \mathsf{snd}(\iota) \mid \mathsf{inl}[\tau](\iota) \mid \mathsf{inr}[\tau](\iota) \mid \mathsf{case}(\iota; x.\iota; x.\iota) \end{array}$$

internal term variable substitutions $\gamma ::= \emptyset \mid \gamma, \iota/x$

internal typing contexts $\Gamma ::= \emptyset \mid \Gamma, x : \tau$

internal type variable substitutions $\delta ::= \emptyset \mid \delta, \tau/\alpha$

internal type formation contexts $\Delta ::= \emptyset \mid \Delta, \alpha \mid \Delta, t$

Figure 1: Syntax of $\mathcal{L}\{\rightharpoonup \forall\, \mu\, \mathsf{1} \times +\}$, our internal language (IL). Metavariable x ranges over term variables and α and t both range over type variables.

kinds

$\kappa \quad ::= \quad \kappa \rightarrow \kappa \mid \boldsymbol{\alpha} \mid \forall(\boldsymbol{\alpha}.\kappa) \mid \mathsf{List}[\kappa] \mid \mathsf{1} \mid \kappa \times \kappa \mid \kappa + \kappa \mid \mathsf{Ty} \mid \mathsf{ITy} \mid \mathsf{ITm}$

static terms

$$\begin{array}{lll} \sigma & ::= & \boldsymbol{x} \mid \lambda \boldsymbol{x}{::}\kappa.\sigma \mid \sigma(\sigma) \mid \Lambda(\boldsymbol{\alpha}.\sigma) \mid \sigma[\kappa] \mid \mathsf{nil}[\kappa] \mid \mathsf{cons}(\sigma; \sigma) \mid \mathsf{listrec}(\sigma; \sigma; \boldsymbol{x}, \boldsymbol{y}.\sigma) \\ & \mid & () \mid (\sigma, \sigma) \mid \mathsf{fst}(\sigma) \mid \mathsf{snd}(\sigma) \mid \mathsf{inl}[\kappa](\sigma) \mid \mathsf{inr}[\kappa](\sigma) \mid \mathsf{case}(\sigma; \boldsymbol{x}.\sigma; \boldsymbol{x}.\sigma) \mid \mathsf{raise}[\kappa] \\ & \mid & c\langle\sigma\rangle \mid \mathsf{tycase}[c](\sigma; \boldsymbol{x}.\sigma; \sigma) \\ & \mid & \blacktriangleright(\hat{\tau}) \mid \triangleright(\hat{\iota}) \mid \mathsf{ana}[n](\sigma) \mid \mathsf{syn}[n] \end{array}$$

translational internal types and terms

$$\begin{array}{lll} \hat{\tau} & ::= & \blacktriangleleft(\sigma) \mid \mathsf{trans}(\sigma) \mid \hat{\tau} \rightharpoonup \hat{\tau} \mid \alpha \mid \forall(\alpha.\hat{\tau}) \mid t \mid \mu(t.\hat{\tau}) \mid \mathsf{1} \mid \hat{\tau} \times \hat{\tau} \mid \hat{\tau} + \hat{\tau} \\ \hat{\iota} & ::= & \triangleleft(\sigma) \mid \mathsf{anatrans}[n](\sigma) \mid \mathsf{syntrans}[n] \mid x \mid \lambda x{:}\hat{\tau}.\hat{\iota} \mid \hat{\iota}(\hat{\iota}) \mid \mathsf{fix}[\hat{\tau}](x.\hat{\iota}) \mid \Lambda(\alpha.\hat{\iota}) \mid \hat{\iota}[\hat{\tau}] \\ & \mid & \mathsf{fold}[t.\hat{\tau}](\hat{\iota}) \mid \mathsf{unfold}(\hat{\iota}) \mid () \mid (\hat{\iota}, \hat{\iota}) \mid \mathsf{fst}(\hat{\iota}) \mid \mathsf{snd}(\hat{\iota}) \mid \mathsf{inl}[\hat{\tau}](\hat{\iota}) \mid \mathsf{inr}[\hat{\tau}](\hat{\iota}) \mid \mathsf{case}(\iota; x.\hat{\iota}; x.\hat{\iota}) \end{array}$$

kinding contexts $\boldsymbol{\Gamma} ::= \emptyset \mid \boldsymbol{\Gamma}, \boldsymbol{x} :: \kappa$

kind formation contexts $\boldsymbol{\Delta} ::= \emptyset \mid \boldsymbol{\Delta}, \boldsymbol{\alpha}$

Figure 2: Syntax of the SL. Note that we have replaced inductive kinds as in the paper with the only particular example of an inductive kind, $\mathsf{List}[\kappa]$, that is needed by the semantics (see text).

$$\begin{array}{lccl} \textbf{tycons} & c & ::= & \rightharpoonup \ \mid \textsc{tc} \mid \mathsf{other}[m; \kappa] \\ \textbf{tycon contexts} & \Phi & ::= & \cdot \mid \Phi, \mathsf{tycon}\ \textsc{tc}\ \{\theta\} \sim \psi \\ \textbf{tycon structures} & \theta & ::= & \mathsf{trans} = \sigma\ \mathsf{in}\ \omega \\ \textbf{opcon structures} & \omega & ::= & \mathsf{ana\ intro} = \sigma \mid \omega; \mathsf{syn}\ \mathbf{op} = \sigma \\ \textbf{tycon sigs} & \psi & ::= & \mathsf{tcsig}[\kappa]\ \{\chi\} \\ \textbf{opcon sigs} & \chi & ::= & \mathsf{intro}[\kappa] \mid \chi; \mathbf{op}[\kappa] \end{array}$$

Figure 3: Syntax of Tycons and Tycon Contexts. Metavariables TC and **op** range over user-defined tycon and opcon names, respectively, and m ranges over natural numbers.

external terms
$e \quad ::= \quad x \mid \lambda x.e \mid \lambda x{:}\sigma.e \mid e(e) \mid \mathsf{fix}(x.e) \mid e : \sigma \mid \mathsf{intro}[\sigma](\overline{e}) \mid \mathsf{targop}[\mathbf{op}; \sigma](e; \overline{e})$
argument lists $\overline{e} ::= \cdot \mid \overline{e}; e$
external typing contexts $\Upsilon ::= \emptyset \mid \Upsilon, x : \sigma$
argument environments $\mathcal{A} ::= \overline{e}; \Upsilon; \Phi$
type translation stores $\mathcal{D} ::= \emptyset \mid \mathcal{D}, \sigma \leftrightarrow \tau/\alpha$
term translation stores $\mathcal{G} ::= \emptyset \mid \mathcal{G}, n : \sigma \rightsquigarrow \iota/x : \tau$

Figure 4: Syntax of External Language (EL)

Description	**Concrete Form**	**Desugared Form**
sequences	$(e_1, \ldots, e_n)$ or $\texttt{[}e_1, \ldots, e_n\texttt{]}$	$\mathsf{intro}[()](e_1; \ldots; e_n)$
labeled collections	$\{\texttt{lbl}_1 = e_1, \ldots, \texttt{lbl}_n = e_n\}$	$\mathsf{intro}[\texttt{[lbl}_1, \ldots, \texttt{lbl}_n\texttt{]}](e_1; \ldots; e_n)$
label application	$\texttt{lbl}\langle e_1, \ldots, e_n\rangle$	$\mathsf{intro}[\texttt{lbl}](e_1, \ldots, e_n)$
numerals	n	$\mathsf{intro}[n_{\text{SL}}](\cdot)$
labeled numerals	$n\texttt{lbl}$	$\mathsf{intro}[(n, \texttt{lbl})](\cdot)$
strings	$\texttt{"s"}$	$\mathsf{intro}[\texttt{"s"}_{\text{SL}}](\cdot)$

Figure 5: Example Introductory Desugarings. We assume static desugarings as shown for labels (of kind Lbl), lists of labels (of kind List[Lbl]), numbers (of kind Num) and strings (of kind Str). These kinds are definable in the standard way.

Description	**Concrete Form**	**Desugared Form**
explicit invocation	$e_{\text{targ}}\cdot\mathbf{op}[\sigma_{\text{tmidx}}](\overline{e})$	$\mathsf{targop}[\mathbf{op}; \sigma_{\text{tmidx}}](e_{\text{targ}}; \overline{e})$
	$e_{\text{targ}}\cdot\mathbf{op}(\overline{e})$	$\mathsf{targop}[\mathbf{op}; ()](e_{\text{targ}}; \overline{e})$
	$e_{\text{targ}}\cdot\mathbf{op}(\texttt{lbl}_1{=}e_1, \ldots, \texttt{lbl}_n{=}e_n)$	$\mathsf{targop}[\mathbf{op}; \texttt{[lbl}_1, \ldots, \texttt{lbl}_n\texttt{]}](e_{\text{targ}}; e_1; \ldots; e_n)$
index projection	$e_{\text{targ}}\texttt{\#}n$	$\mathsf{targop}[\mathbf{idx}; n](e_{\text{targ}}; \cdot)$
label projection	$e_{\text{targ}}\texttt{\#lbl}$	$\mathsf{targop}[\texttt{\#}; \texttt{lbl}](e_{\text{targ}}; \cdot)$
concatenation	$e_{\text{targ}} \cdot e'$	$\mathsf{targop}[\mathbf{conc}; ()](e_{\text{targ}}; e')$
case analysis	$e_{\text{targ}}\cdot\mathbf{case}[\sigma]\ \{$	$\mathsf{targop}[\mathbf{case}; (\sigma, \texttt{[}\sigma_1, \ldots, \sigma_n\texttt{]})](e_{\text{targ}};$
	$\mid\ \sigma_1\langle x_1, \ldots, x_k\rangle \Rightarrow e_1$	$\lambda x_1. \ldots \lambda x_k.e_1;$
	$\mid\ \ldots$	$\ldots;$
	$\mid\ \sigma_n\langle x_1, \ldots, x_k\rangle \Rightarrow e_n\}$	$\lambda x_1. \ldots \lambda x_k.e_n)$

Figure 6: Example Targeted Desugarings. The case analysis form makes use of unannotated lambda functions, which are included here but were not discussed in the paper. Note that because we only define synthetic targeted operators, the case analysis operator must specify the type the cases must have explicitly (σ). Analytic targeted operators are a natural extension to our calculus, but we do not define them here. Another natural extension is to introduce a syntactic sort for patterns, with positional binding, but again, this is beyond the scope of our present paper.

$\boxed{\Delta \vdash \delta : \Delta}$

(tysub-emp)

$$\frac{}{\Delta \vdash \emptyset : \emptyset}$$

(tysub-ext)

$$\frac{\Delta \vdash \delta : \Delta' \qquad \Delta \vdash \tau}{\Delta \vdash \delta, \tau/\alpha : \Delta', \alpha}$$

Figure 7: Internal Type Substitution Validity

$\boxed{\Delta\ \Gamma \vdash \gamma : \Gamma}$

(tmsub-emp)

$$\frac{}{\Delta\ \Gamma \vdash \emptyset : \emptyset}$$

(tmsub-ext)

$$\frac{\Delta\ \Gamma \vdash \gamma : \Gamma' \qquad \Delta \vdash \tau \qquad \Delta\ \Gamma \vdash \iota : \tau}{\Delta\ \Gamma \vdash \gamma, \iota/x : \Gamma', x : \tau}$$

Figure 8: Internal Term Substitution Validity

$\boxed{\Delta \vdash \Gamma}$

(ictx-emp)

$$\frac{}{\Delta \vdash \emptyset}$$

(ictx-ext)

$$\frac{\Delta \vdash \Gamma \qquad \Delta \vdash \tau}{\Delta \vdash \Gamma, x : \tau}$$

Figure 9: Internal Typing Context Formation

$\boxed{\vdash \Phi}$

(tcc-emp)

$$\frac{}{\vdash \cdot}$$

(tcc-ext)

$$\frac{\begin{array}{c}\vdash \Phi \qquad \vdash \kappa_{\text{tyidx}}\ \mathsf{eq} \qquad \emptyset\ \emptyset \vdash^0_\Phi \sigma_{\text{schema}} :: \kappa_{\text{tyidx}} \to \mathsf{ITy} \\ \vdash_{\Phi, \mathsf{tycon}\ \textsc{tc}\ \{\mathsf{trans}=\sigma_{\text{schema}}\ \mathsf{in}\ \omega\} \sim \mathsf{tcsig}[\kappa_{\text{tyidx}}]\ \{\chi\}}\ \omega \sim \mathsf{tcsig}[\kappa_{\text{tyidx}}]\ \{\chi\}\end{array}}{\vdash \Phi, \mathsf{tycon}\ \textsc{tc}\ \{\mathsf{trans} = \sigma_{\text{schema}}\ \mathsf{in}\ \omega\} \sim \mathsf{tcsig}[\kappa_{\text{tyidx}}]\ \{\chi\}}$$

Figure 10: Tycon Context Validity

$\boxed{\vdash_\Phi \omega \sim \psi}$

(ocstruct-intro)

$$\frac{\emptyset \vdash \kappa_{\text{tmidx}} \qquad \emptyset\ \emptyset \vdash^0_\Phi \sigma_{\text{def}} :: \kappa_{\text{tyidx}} \to \kappa_{\text{tmidx}} \to \mathsf{List}[\mathsf{Arg}] \to \mathsf{ITm}}{\vdash_\Phi \mathsf{ana\ intro} = \sigma_{\text{def}} \sim \mathsf{tcsig}[\kappa_{\text{tyidx}}]\ \{\mathsf{intro}[\kappa_{\text{tmidx}}]\}}$$

(ocstruct-targ)

$$\frac{\begin{array}{c}\vdash_\Phi \omega \sim \mathsf{tcsig}[\kappa_{\text{tyidx}}]\ \{\chi\} \qquad \emptyset \vdash \kappa_{\text{tmidx}} \\ \emptyset\ \emptyset \vdash^0_\Phi \sigma_{\text{def}} :: \kappa_{\text{tyidx}} \to \kappa_{\text{tmidx}} \to \mathsf{List}[\mathsf{Arg}] \to (\mathsf{Ty} \times \mathsf{ITm})\end{array}}{\vdash_\Phi \omega; \mathsf{syn}\ \mathbf{op} = \sigma_{\text{def}} \sim \mathsf{tcsig}[\kappa_{\text{tyidx}}]\ \{\chi; \mathbf{op}[\kappa_{\text{tmidx}}]\}}$$

Figure 11: Opcon Structure Validity

$\boxed{\vdash \psi}$

(tcsig-ok)
$$\frac{\vdash \kappa_{\text{tyidx}}\ \mathsf{eq} \qquad \vdash \chi}{\vdash \mathsf{tcsig}[\kappa_{\text{tyidx}}]\ \{\chi\}}$$

$\boxed{\vdash \chi}$

(ocsig-intro-ok)
$$\frac{\emptyset \vdash \kappa}{\vdash \mathsf{intro}[\kappa]}$$

(ocsig-targ-ok)
$$\frac{\vdash \chi \qquad \emptyset \vdash \kappa}{\vdash \chi; \mathbf{op}[\kappa]}$$

$\boxed{\vdash \Phi\ \texttt{sigsok}}$

(tcc-sigsok-emp)
$$\frac{}{\vdash \emptyset\ \texttt{sigsok}}$$

(tcc-sigsok-ext)
$$\frac{\vdash \Phi\ \texttt{sigsok} \qquad \vdash \psi}{\vdash \Phi, \mathsf{tycon}\ \textsc{tc}\ \{\theta\} \sim \psi\ \texttt{sigsok}}$$

Figure 12: Tycon and Opcon Signature Validity

$\boxed{\boldsymbol{\Delta} \vdash \kappa}$

(kf-arrow)
$$\frac{\boldsymbol{\Delta} \vdash \kappa_1 \qquad \boldsymbol{\Delta} \vdash \kappa_2}{\boldsymbol{\Delta} \vdash \kappa_1 \to \kappa_2}$$

(kf-alpha)
$$\frac{\boldsymbol{\alpha} \in \boldsymbol{\Delta}}{\boldsymbol{\Delta} \vdash \boldsymbol{\alpha}}$$

(kf-forall)
$$\frac{\boldsymbol{\Delta}, \boldsymbol{\alpha} \vdash \kappa}{\boldsymbol{\Delta} \vdash \forall(\boldsymbol{\alpha}.\kappa)}$$

(kf-list)
$$\frac{\boldsymbol{\Delta} \vdash \kappa}{\boldsymbol{\Delta} \vdash \mathsf{List}[\kappa]}$$

(kf-unit)
$$\frac{}{\boldsymbol{\Delta} \vdash 1}$$

(kf-prod)
$$\frac{\boldsymbol{\Delta} \vdash \kappa_1 \qquad \boldsymbol{\Delta} \vdash \kappa_2}{\boldsymbol{\Delta} \vdash \kappa_1 \times \kappa_2}$$

(kf-sum)
$$\frac{\boldsymbol{\Delta} \vdash \kappa_1 \qquad \boldsymbol{\Delta} \vdash \kappa_2}{\boldsymbol{\Delta} \vdash \kappa_1 + \kappa_2}$$

(kf-ty)
$$\frac{}{\boldsymbol{\Delta} \vdash \mathsf{Ty}}$$

(kf-ity)
$$\frac{}{\boldsymbol{\Delta} \vdash \mathsf{ITy}}$$

(kf-itm)
$$\frac{}{\boldsymbol{\Delta} \vdash \mathsf{ITm}}$$

Figure 13: Kind Formation.

$\boxed{\vdash \kappa\ \mathsf{eq}}$

(keq-list)
$$\frac{\vdash \kappa\ \mathsf{eq}}{\vdash \mathsf{List}[\kappa]\ \mathsf{eq}}$$

(keq-unit)
$$\frac{}{\vdash 1\ \mathsf{eq}}$$

(keq-prod)
$$\frac{\vdash \kappa_1\ \mathsf{eq} \qquad \vdash \kappa_2\ \mathsf{eq}}{\vdash \kappa_1 \times \kappa_2\ \mathsf{eq}}$$

(keq-sum)
$$\frac{\vdash \kappa_1\ \mathsf{eq} \qquad \vdash \kappa_2\ \mathsf{eq}}{\vdash \kappa_1 + \kappa_2\ \mathsf{eq}}$$

(keq-ty)
$$\frac{}{\vdash \mathsf{Ty}\ \mathsf{eq}}$$

(keq-ity)
$$\frac{}{\vdash \mathsf{ITy}\ \mathsf{eq}}$$

Figure 14: Equality Kinds

$\boxed{\boldsymbol{\Delta} \vdash \boldsymbol{\Gamma}}$

(kctx-emp)
$$\frac{}{\boldsymbol{\Delta} \vdash \emptyset}$$

(kctx-ext)
$$\frac{\boldsymbol{\Delta} \vdash \boldsymbol{\Gamma} \qquad \boldsymbol{\Delta} \vdash \kappa}{\boldsymbol{\Delta} \vdash \boldsymbol{\Gamma}, \boldsymbol{x} :: \kappa}$$

Figure 15: Kinding Context Formation

$\boxed{\boldsymbol{\Delta}\ \boldsymbol{\Gamma} \vdash^n_\Phi \sigma :: \kappa}$ (continues into Figure 17)

(k-var)
$$\frac{\boldsymbol{x} :: \kappa \in \boldsymbol{\Gamma}}{\boldsymbol{\Delta}\ \boldsymbol{\Gamma} \vdash^n_\Phi \boldsymbol{x} :: \kappa}$$

(k-abs)
$$\frac{\boldsymbol{\Delta} \vdash \kappa_1 \qquad \boldsymbol{\Delta}\ \boldsymbol{\Gamma}, \boldsymbol{x} :: \kappa_1 \vdash^n_\Phi \sigma :: \kappa_2}{\boldsymbol{\Delta}\ \boldsymbol{\Gamma} \vdash^n_\Phi \lambda\boldsymbol{x}{::}\kappa_1.\sigma :: \kappa_1 \rightarrow \kappa_2}$$

(k-ap)
$$\frac{\boldsymbol{\Delta}\ \boldsymbol{\Gamma} \vdash^n_\Phi \sigma_1 :: \kappa_1 \rightarrow \kappa_2 \qquad \boldsymbol{\Delta}\ \boldsymbol{\Gamma} \vdash^n_\Phi \sigma_2 :: \kappa_1}{\boldsymbol{\Delta}\ \boldsymbol{\Gamma} \vdash^n_\Phi \sigma_1(\sigma_2) :: \kappa_2}$$

(k-kabs)
$$\frac{\boldsymbol{\Delta}, \boldsymbol{\alpha}\ \boldsymbol{\Gamma} \vdash^n_\Phi \sigma :: \kappa}{\boldsymbol{\Delta}\ \boldsymbol{\Gamma} \vdash^n_\Phi \Lambda(\boldsymbol{\alpha}.\sigma) :: \forall(\boldsymbol{\alpha}.\kappa)}$$

(k-kap)
$$\frac{\boldsymbol{\Delta}\ \boldsymbol{\Gamma} \vdash^n_\Phi \sigma :: \forall(\boldsymbol{\alpha}.\kappa') \qquad \boldsymbol{\Delta} \vdash \kappa}{\boldsymbol{\Delta}\ \boldsymbol{\Gamma} \vdash^n_\Phi \sigma[\kappa] :: [\kappa/\boldsymbol{\alpha}]\kappa'}$$

(k-nil)
$$\frac{\boldsymbol{\Delta} \vdash \kappa}{\boldsymbol{\Delta}\ \boldsymbol{\Gamma} \vdash^n_\Phi \mathsf{nil}[\kappa] :: \mathsf{List}[\kappa]}$$

(k-cons)
$$\frac{\boldsymbol{\Delta}\ \boldsymbol{\Gamma} \vdash^n_\Phi \sigma_1 :: \kappa \qquad \boldsymbol{\Delta}\ \boldsymbol{\Gamma} \vdash^n_\Phi \sigma_2 :: \mathsf{List}[\kappa]}{\boldsymbol{\Delta}\ \boldsymbol{\Gamma} \vdash^n_\Phi \mathsf{cons}(\sigma_1; \sigma_2) :: \mathsf{List}[\kappa]}$$

(k-listrec)
$$\frac{\begin{array}{c}\boldsymbol{\Delta}\ \boldsymbol{\Gamma} \vdash^n_\Phi \sigma_1 :: \mathsf{List}[\kappa] \qquad \boldsymbol{\Delta}\ \boldsymbol{\Gamma} \vdash^n_\Phi \sigma_2 :: \kappa' \\ \boldsymbol{\Delta}\ \boldsymbol{\Gamma}, \boldsymbol{x} :: \kappa, \boldsymbol{y} :: \kappa' \vdash^n_\Phi \sigma_3 :: \kappa'\end{array}}{\boldsymbol{\Delta}\ \boldsymbol{\Gamma} \vdash^n_\Phi \mathsf{listrec}(\sigma_1; \sigma_2; \boldsymbol{x}, \boldsymbol{y}.\sigma_3) :: \kappa'}$$

(k-triv)
$$\frac{}{\boldsymbol{\Delta}\ \boldsymbol{\Gamma} \vdash^n_\Phi () :: 1}$$

(k-pair)
$$\frac{\boldsymbol{\Delta}\ \boldsymbol{\Gamma} \vdash^n_\Phi \sigma_1 :: \kappa_1 \qquad \boldsymbol{\Delta}\ \boldsymbol{\Gamma} \vdash^n_\Phi \sigma_2 :: \kappa_2}{\boldsymbol{\Delta}\ \boldsymbol{\Gamma} \vdash^n_\Phi (\sigma_1, \sigma_2) :: \kappa_1 \times \kappa_2}$$

(k-fst)
$$\frac{\boldsymbol{\Delta}\ \boldsymbol{\Gamma} \vdash^n_\Phi \sigma :: \kappa_1 \times \kappa_2}{\boldsymbol{\Delta}\ \boldsymbol{\Gamma} \vdash^n_\Phi \mathsf{fst}(\sigma) :: \kappa_1}$$

(k-snd)
$$\frac{\boldsymbol{\Delta}\ \boldsymbol{\Gamma} \vdash^n_\Phi \sigma :: \kappa_1 \times \kappa_2}{\boldsymbol{\Delta}\ \boldsymbol{\Gamma} \vdash^n_\Phi \mathsf{snd}(\sigma) :: \kappa_2}$$

(k-inl)
$$\frac{\boldsymbol{\Delta} \vdash \kappa_1 + \kappa_2 \qquad \boldsymbol{\Delta}\ \boldsymbol{\Gamma} \vdash^n_\Phi \sigma :: \kappa_1}{\boldsymbol{\Delta}\ \boldsymbol{\Gamma} \vdash^n_\Phi \mathsf{inl}[\kappa_1 + \kappa_2](\sigma) :: \kappa_1 + \kappa_2}$$

(k-inr)
$$\frac{\boldsymbol{\Delta} \vdash \kappa_1 + \kappa_2 \qquad \boldsymbol{\Delta}\ \boldsymbol{\Gamma} \vdash^n_\Phi \sigma :: \kappa_2}{\boldsymbol{\Delta}\ \boldsymbol{\Gamma} \vdash^n_\Phi \mathsf{inr}[\kappa_1 + \kappa_2](\sigma) :: \kappa_1 + \kappa_2}$$

(k-case)
$$\frac{\boldsymbol{\Delta}\ \boldsymbol{\Gamma} \vdash^n_\Phi \sigma_1 :: \kappa_1 + \kappa_2 \qquad \boldsymbol{\Delta}\ \boldsymbol{\Gamma}, \boldsymbol{x} :: \kappa_1 \vdash^n_\Phi \sigma_1 :: \kappa \qquad \boldsymbol{\Delta}\ \boldsymbol{\Gamma}, \boldsymbol{x} :: \kappa_2 \vdash^n_\Phi \sigma_2 :: \kappa}{\boldsymbol{\Delta}\ \boldsymbol{\Gamma} \vdash^n_\Phi \mathsf{case}(\sigma_1; \boldsymbol{x}.\sigma_2; \boldsymbol{x}.\sigma_3) :: \kappa}$$

(k-raise)
$$\frac{\boldsymbol{\Delta} \vdash \kappa}{\boldsymbol{\Delta}\ \boldsymbol{\Gamma} \vdash^n_\Phi \mathsf{raise}[\kappa] :: \kappa}$$

Figure 16: Kinding - Core of Static Language. All rules directly follow from [1] with the addition only of Φ and n to each judgement. These are not inspected by the core.

$\boxed{\mathbf{\Delta}\ \mathbf{\Gamma} \vdash^n_\Phi \sigma :: \kappa}$ (continued from Figure 16, continues into Figure 18)

(k-ty-parr)
$$\frac{\mathbf{\Delta}\ \mathbf{\Gamma} \vdash^n_\Phi \sigma :: \mathsf{Ty} \times \mathsf{Ty}}{\mathbf{\Delta}\ \mathbf{\Gamma} \vdash^n_\Phi \rightharpoonup\langle\sigma\rangle :: \mathsf{Ty}}$$

(k-ty-ext)
$$\frac{\mathsf{tycon}\ \textsc{tc}\ \{\theta\} \sim \mathsf{tcsig}[\kappa_{\text{tyidx}}]\ \{\chi\} \in \Phi \qquad \mathbf{\Delta}\ \mathbf{\Gamma} \vdash^n_\Phi \sigma :: \kappa_{\text{tyidx}}}{\mathbf{\Delta}\ \mathbf{\Gamma} \vdash^n_\Phi \textsc{tc}\langle\sigma\rangle :: \mathsf{Ty}}$$

(k-ty-other)
$$\frac{\vdash \kappa\ \mathsf{eq} \qquad \mathbf{\Delta}\ \mathbf{\Gamma} \vdash^n_\Phi \sigma_{\text{tyidx}} :: \kappa \times \mathsf{ITy}}{\mathbf{\Delta}\ \mathbf{\Gamma} \vdash^n_\Phi \mathsf{other}[m;\kappa]\langle\sigma_{\text{tyidx}}\rangle :: \mathsf{Ty}}$$

(k-tycase-parr)
$$\frac{\mathbf{\Delta}\ \mathbf{\Gamma} \vdash^n_\Phi \sigma :: \mathsf{Ty} \qquad \mathbf{\Delta}\ \mathbf{\Gamma}, \boldsymbol{x} :: \mathsf{Ty} \times \mathsf{Ty} \vdash^n_\Phi \sigma_1 :: \kappa \qquad \mathbf{\Delta}\ \mathbf{\Gamma} \vdash^n_\Phi \sigma_2 :: \kappa}{\mathbf{\Delta}\ \mathbf{\Gamma} \vdash^n_\Phi \mathsf{tycase}[\rightharpoonup](\sigma; \boldsymbol{x}.\sigma_1; \sigma_2) :: \kappa}$$

(k-tycase-ext)
$$\frac{\mathbf{\Delta}\ \mathbf{\Gamma} \vdash^n_\Phi \sigma :: \mathsf{Ty} \qquad \mathsf{tycon}\ \textsc{tc}\ \{\theta\} \sim \mathsf{tcsig}[\kappa_{\text{tyidx}}]\ \{\chi\} \in \Phi \\ \mathbf{\Delta}\ \mathbf{\Gamma}, \boldsymbol{x} :: \kappa_{\text{tyidx}} \vdash^n_\Phi \sigma_1 :: \kappa \qquad \mathbf{\Delta}\ \mathbf{\Gamma} \vdash^n_\Phi \sigma_2 :: \kappa}{\mathbf{\Delta}\ \mathbf{\Gamma} \vdash^n_\Phi \mathsf{tycase}[\textsc{tc}](\sigma; \boldsymbol{x}.\sigma_1; \sigma_2) :: \kappa}$$

(no rule for type case analysis where $c = \mathsf{other}[m;\kappa]$)

Figure 17: Kinding - Types

$\boxed{\mathbf{\Delta}\ \mathbf{\Gamma} \vdash^n_\Phi \sigma :: \kappa}$ (continued from Figure 17, continues into Figure 19)

(k-qity-unquote)
$$\frac{\mathbf{\Delta}\ \mathbf{\Gamma} \vdash^n_\Phi \sigma :: \mathsf{ITy}}{\mathbf{\Delta}\ \mathbf{\Gamma} \vdash^n_\Phi \blacktriangleright(\blacktriangleleft(\sigma)) :: \mathsf{ITy}}$$

(k-qity-trans)
$$\frac{\mathbf{\Delta}\ \mathbf{\Gamma} \vdash^n_\Phi \sigma :: \mathsf{Ty}}{\mathbf{\Delta}\ \mathbf{\Gamma} \vdash^n_\Phi \blacktriangleright(\mathsf{trans}(\sigma)) :: \mathsf{ITy}}$$

Figure 18: Kinding - Quoted Translational Internal Types (Unquote and Type Translation References)

$\boxed{\mathbf{\Delta}\ \mathbf{\Gamma} \vdash^n_\Phi \sigma :: \kappa}$ (continued from Figure 18, continues into Figure 20)

(k-qity-parr)
$$\frac{\mathbf{\Delta}\ \mathbf{\Gamma} \vdash^n_\Phi \blacktriangleright(\hat{\tau}_1) :: \mathsf{ITy} \qquad \mathbf{\Delta}\ \mathbf{\Gamma} \vdash^n_\Phi \blacktriangleright(\hat{\tau}_2) :: \mathsf{ITy}}{\mathbf{\Delta}\ \mathbf{\Gamma} \vdash^n_\Phi \blacktriangleright(\hat{\tau}_1 \rightharpoonup \hat{\tau}_2) :: \mathsf{ITy}}$$

(k-qity-alpha)
$$\frac{}{\mathbf{\Delta}\ \mathbf{\Gamma} \vdash^n_\Phi \blacktriangleright(\alpha) :: \mathsf{ITy}}$$

(k-qity-forall)
$$\frac{\mathbf{\Delta}\ \mathbf{\Gamma} \vdash^n_\Phi \blacktriangleright(\hat{\tau}) :: \mathsf{ITy}}{\mathbf{\Delta}\ \mathbf{\Gamma} \vdash^n_\Phi \blacktriangleright(\forall(\alpha.\hat{\tau})) :: \mathsf{ITy}}$$

(k-qity-t)
$$\frac{}{\mathbf{\Delta}\ \mathbf{\Gamma} \vdash^n_\Phi \blacktriangleright(t) :: \mathsf{ITy}}$$

(k-qity-mu)
$$\frac{\mathbf{\Delta}\ \mathbf{\Gamma} \vdash^n_\Phi \blacktriangleright(\hat{\tau}) :: \mathsf{ITy}}{\mathbf{\Delta}\ \mathbf{\Gamma} \vdash^n_\Phi \blacktriangleright(\mu(t.\hat{\tau})) :: \mathsf{ITy}}$$

(k-qity-unit)
$$\frac{}{\mathbf{\Delta}\ \mathbf{\Gamma} \vdash^n_\Phi \blacktriangleright(1) :: \mathsf{ITy}}$$

(k-qity-prod)
$$\frac{\mathbf{\Delta}\ \mathbf{\Gamma} \vdash^n_\Phi \blacktriangleright(\hat{\tau}_1) :: \mathsf{ITy} \qquad \mathbf{\Delta}\ \mathbf{\Gamma} \vdash^n_\Phi \blacktriangleright(\hat{\tau}_2) :: \mathsf{ITy}}{\mathbf{\Delta}\ \mathbf{\Gamma} \vdash^n_\Phi \blacktriangleright(\hat{\tau}_1 \times \hat{\tau}_2) :: \mathsf{ITy}}$$

(k-qity-sum)
$$\frac{\mathbf{\Delta}\ \mathbf{\Gamma} \vdash^n_\Phi \blacktriangleright(\hat{\tau}_1) :: \mathsf{ITy} \qquad \mathbf{\Delta}\ \mathbf{\Gamma} \vdash^n_\Phi \blacktriangleright(\hat{\tau}_2) :: \mathsf{ITy}}{\mathbf{\Delta}\ \mathbf{\Gamma} \vdash^n_\Phi \blacktriangleright(\hat{\tau}_1 + \hat{\tau}_2) :: \mathsf{ITy}}$$

Figure 19: Kinding - Quoted Translational Internal Types (Shared Forms). All shared forms simply recursively quote and then kind check all sub-terms against ITy.

$\boxed{\boldsymbol{\Delta}\;\boldsymbol{\Gamma} \vdash^n_\Phi \sigma :: \kappa}$ (continued from Figure 19, continues into Figure 21)

(k-qitm-unquote)
$$\frac{\boldsymbol{\Delta}\;\boldsymbol{\Gamma} \vdash^n_\Phi \sigma :: \mathsf{ITm}}{\boldsymbol{\Delta}\;\boldsymbol{\Gamma} \vdash^n_\Phi \rhd(\lhd(\sigma)) :: \mathsf{ITm}}$$

(k-qitm-anatrans)
$$\frac{n' < n \qquad \boldsymbol{\Delta}\;\boldsymbol{\Gamma} \vdash^n_\Phi \sigma :: \mathsf{Ty}}{\boldsymbol{\Delta}\;\boldsymbol{\Gamma} \vdash^n_\Phi \rhd(\mathsf{anatrans}[n'](\sigma)) :: \mathsf{ITm}}$$

(k-qitm-syntrans)
$$\frac{n' < n}{\boldsymbol{\Delta}\;\boldsymbol{\Gamma} \vdash^n_\Phi \rhd(\mathsf{syntrans}[n']) :: \mathsf{ITm}}$$

Figure 20: Kinding - Quoted Translational Internal Terms (Unquote and Argument Translation References)

$\boxed{\boldsymbol{\Delta}\;\boldsymbol{\Gamma} \vdash^n_\Phi \sigma :: \kappa}$ (continued from Figure 20, continues into Figure 22)

(k-qitm-var)
$$\frac{}{\boldsymbol{\Delta}\;\boldsymbol{\Gamma} \vdash^n_\Phi \rhd(x) :: \mathsf{ITm}}$$

(k-qitm-abs)
$$\frac{\boldsymbol{\Delta}\;\boldsymbol{\Gamma} \vdash^n_\Phi \blacktriangleright(\hat{\tau}) :: \mathsf{ITy} \qquad \boldsymbol{\Delta}\;\boldsymbol{\Gamma} \vdash^n_\Phi \rhd(\hat{\iota}) :: \mathsf{ITm}}{\boldsymbol{\Delta}\;\boldsymbol{\Gamma} \vdash^n_\Phi \rhd(\lambda x{:}\hat{\tau}.\hat{\iota}) :: \mathsf{ITm}}$$

(k-qitm-ap)
$$\frac{\boldsymbol{\Delta}\;\boldsymbol{\Gamma} \vdash^n_\Phi \rhd(\hat{\iota}_1) :: \mathsf{ITm} \qquad \boldsymbol{\Delta}\;\boldsymbol{\Gamma} \vdash^n_\Phi \rhd(\hat{\iota}_2) :: \mathsf{ITm}}{\boldsymbol{\Delta}\;\boldsymbol{\Gamma} \vdash^n_\Phi \rhd(\hat{\iota}_1(\hat{\iota}_2)) :: \mathsf{ITm}}$$

(k-qitm-tabs)
$$\frac{\boldsymbol{\Delta}\;\boldsymbol{\Gamma} \vdash^n_\Phi \rhd(\hat{\iota}) :: \mathsf{ITm}}{\boldsymbol{\Delta}\;\boldsymbol{\Gamma} \vdash^n_\Phi \rhd(\Lambda(\alpha.\hat{\iota})) :: \mathsf{ITm}}$$

(k-qitm-tap)
$$\frac{\boldsymbol{\Delta}\;\boldsymbol{\Gamma} \vdash^n_\Phi \rhd(\hat{\iota}) :: \mathsf{ITm} \qquad \boldsymbol{\Delta}\;\boldsymbol{\Gamma} \vdash^n_\Phi \blacktriangleright(\hat{\tau}) :: \mathsf{ITy}}{\boldsymbol{\Delta}\;\boldsymbol{\Gamma} \vdash^n_\Phi \rhd(\hat{\iota}[\hat{\tau}]) :: \mathsf{ITm}}$$

(k-qitm-fold)
$$\frac{\boldsymbol{\Delta}\;\boldsymbol{\Gamma} \vdash^n_\Phi \blacktriangleright(\hat{\tau}) :: \mathsf{ITy} \qquad \boldsymbol{\Delta}\;\boldsymbol{\Gamma} \vdash^n_\Phi \rhd(\hat{\iota}) :: \mathsf{ITm}}{\boldsymbol{\Delta}\;\boldsymbol{\Gamma} \vdash^n_\Phi \rhd(\mathsf{fold}[t.\hat{\tau}](\hat{\iota})) :: \mathsf{ITm}}$$

(k-qitm-unfold)
$$\frac{\boldsymbol{\Delta}\;\boldsymbol{\Gamma} \vdash^n_\Phi \rhd(\hat{\iota}) :: \mathsf{ITm}}{\boldsymbol{\Delta}\;\boldsymbol{\Gamma} \vdash^n_\Phi \rhd(\mathsf{unfold}(\hat{\iota})) :: \mathsf{ITm}}$$

(k-qitm-triv)
$$\frac{}{\boldsymbol{\Delta}\;\boldsymbol{\Gamma} \vdash^n_\Phi \rhd(()) :: \mathsf{ITm}}$$

(k-qitm-pair)
$$\frac{\boldsymbol{\Delta}\;\boldsymbol{\Gamma} \vdash^n_\Phi \rhd(\hat{\iota}_1) :: \mathsf{ITm} \qquad \boldsymbol{\Delta}\;\boldsymbol{\Gamma} \vdash^n_\Phi \rhd(\hat{\iota}_2) :: \mathsf{ITm}}{\boldsymbol{\Delta}\;\boldsymbol{\Gamma} \vdash^n_\Phi \rhd((\hat{\iota}_1, \hat{\iota}_2)) :: \mathsf{ITm}}$$

(k-qitm-fst)
$$\frac{\boldsymbol{\Delta}\;\boldsymbol{\Gamma} \vdash^n_\Phi \rhd(\hat{\iota}) :: \mathsf{ITm}}{\boldsymbol{\Delta}\;\boldsymbol{\Gamma} \vdash^n_\Phi \rhd(\mathsf{fst}(\hat{\iota})) :: \mathsf{ITm}}$$

(k-qitm-snd)
$$\frac{\boldsymbol{\Delta}\;\boldsymbol{\Gamma} \vdash^n_\Phi \rhd(\hat{\iota}) :: \mathsf{ITm}}{\boldsymbol{\Delta}\;\boldsymbol{\Gamma} \vdash^n_\Phi \rhd(\mathsf{snd}(\hat{\iota})) :: \mathsf{ITm}}$$

(k-qitm-inl)
$$\frac{\boldsymbol{\Delta}\;\boldsymbol{\Gamma} \vdash^n_\Phi \blacktriangleright(\hat{\tau}) :: \mathsf{ITy} \qquad \boldsymbol{\Delta}\;\boldsymbol{\Gamma} \vdash^n_\Phi \rhd(\hat{\iota}) :: \mathsf{ITm}}{\boldsymbol{\Delta}\;\boldsymbol{\Gamma} \vdash^n_\Phi \rhd(\mathsf{inl}[\hat{\tau}](\hat{\iota})) :: \mathsf{ITm}}$$

(k-qitm-inr)
$$\frac{\boldsymbol{\Delta}\;\boldsymbol{\Gamma} \vdash^n_\Phi \blacktriangleright(\hat{\tau}) :: \mathsf{ITy} \qquad \boldsymbol{\Delta}\;\boldsymbol{\Gamma} \vdash^n_\Phi \rhd(\hat{\iota}) :: \mathsf{ITm}}{\boldsymbol{\Delta}\;\boldsymbol{\Gamma} \vdash^n_\Phi \rhd(\mathsf{inr}[\hat{\tau}](\hat{\iota})) :: \mathsf{ITm}}$$

(k-qitm-case)
$$\frac{\boldsymbol{\Delta}\;\boldsymbol{\Gamma} \vdash^n_\Phi \rhd(\hat{\iota}_1) :: \mathsf{ITm} \qquad \boldsymbol{\Delta}\;\boldsymbol{\Gamma} \vdash^n_\Phi \rhd(\hat{\iota}_2) :: \mathsf{ITm} \qquad \boldsymbol{\Delta}\;\boldsymbol{\Gamma} \vdash^n_\Phi \rhd(\hat{\iota}_3) :: \mathsf{ITm}}{\boldsymbol{\Delta}\;\boldsymbol{\Gamma} \vdash^n_\Phi \rhd(\mathsf{case}(\hat{\iota}_1; x.\hat{\iota}_2; x.\hat{\iota}_3)) :: \mathsf{ITm}}$$

Figure 21: Kinding - Quoted Translational Internal Terms (Shared Forms). All shared forms simply recursively quote and then kind check all sub-terms against ITm or ITy as appropriate.

$\boxed{\mathbf{\Delta}\ \mathbf{\Gamma} \vdash^{n}_{\Phi} \sigma :: \kappa}$ (continued from Figure 21)

(k-ana)

$$\frac{n' < n \qquad \mathbf{\Delta}\ \mathbf{\Gamma} \vdash^{n}_{\Phi} \sigma :: \mathsf{Ty}}{\mathbf{\Delta}\ \mathbf{\Gamma} \vdash^{n}_{\Phi} \mathsf{ana}[n'](\sigma) :: \mathsf{ITm}}$$

(k-syn)

$$\frac{n' < n}{\mathbf{\Delta}\ \mathbf{\Gamma} \vdash^{n}_{\Phi} \mathsf{syn}[n'] :: \mathsf{Ty} \times \mathsf{ITm}}$$

Figure 22: Kinding - SL-EL Interface

$\boxed{\sigma \mapsto_{\mathcal{A}} \sigma'}$ $\boxed{\sigma\ \mathtt{val}_{\mathcal{A}}}$ $\boxed{\sigma\ \mathtt{err}_{\mathcal{A}}}$ (continues into Figure 24)

(n-abs-val)
$$\frac{}{\lambda\boldsymbol{x}{::}\kappa.\sigma\ \mathtt{val}_{\mathcal{A}}}$$

(n-ap-step-1)
$$\frac{\sigma_1 \mapsto_{\mathcal{A}} \sigma_1'}{\sigma_1(\sigma_2) \mapsto_{\mathcal{A}} \sigma_1'(\sigma_2)}$$

(n-ap-step-2)
$$\frac{\sigma_1\ \mathtt{val}_{\mathcal{A}} \qquad \sigma_2 \mapsto_{\mathcal{A}} \sigma_2'}{\sigma_1(\sigma_2) \mapsto_{\mathcal{A}} \sigma_1(\sigma_2')}$$

(n-ap-elim)
$$\frac{\sigma_2\ \mathtt{val}_{\mathcal{A}}}{(\lambda\boldsymbol{x}{::}\kappa.\sigma_1)(\sigma_2) \mapsto_{\mathcal{A}} [\sigma_2/\boldsymbol{x}]\sigma_1}$$

(n-ap-err-prop-1)
$$\frac{\sigma_1\ \mathtt{err}_{\mathcal{A}}}{\sigma_1(\sigma_2)\ \mathtt{err}_{\mathcal{A}}}$$

(n-ap-err-prop-2)
$$\frac{\sigma_2\ \mathtt{err}_{\mathcal{A}}}{\sigma_1(\sigma_2)\ \mathtt{err}_{\mathcal{A}}}$$

(n-tabs-val)
$$\frac{}{\Lambda(\boldsymbol{\alpha}.\sigma)\ \mathtt{val}_{\mathcal{A}}}$$

(n-tap-step)
$$\frac{\sigma \mapsto_{\mathcal{A}} \sigma'}{\sigma\mathtt{[}\kappa\mathtt{]} \mapsto_{\mathcal{A}} \sigma'\mathtt{[}\kappa\mathtt{]}}$$

(n-tap-elim)
$$\frac{}{(\Lambda(\boldsymbol{\alpha}.\sigma))\mathtt{[}\kappa\mathtt{]} \mapsto_{\mathcal{A}} [\kappa/\boldsymbol{\alpha}]\sigma}$$

(n-tap-err-prop)
$$\frac{\sigma\ \mathtt{err}_{\mathcal{A}}}{\sigma\mathtt{[}\kappa\mathtt{]}\ \mathtt{err}_{\mathcal{A}}}$$

(n-nil-val)
$$\frac{}{\mathsf{nil}[\kappa]\ \mathtt{val}_{\mathcal{A}}}$$

(n-cons-step-1)
$$\frac{\sigma_1 \mapsto_{\mathcal{A}} \sigma_1'}{\mathsf{cons}(\sigma_1;\sigma_2) \mapsto_{\mathcal{A}} \mathsf{cons}(\sigma_1';\sigma_2)}$$

(n-cons-step-2)
$$\frac{\sigma_1\ \mathtt{val}_{\mathcal{A}} \qquad \sigma_2 \mapsto_{\mathcal{A}} \sigma_2'}{\mathsf{cons}(\sigma_1;\sigma_2) \mapsto_{\mathcal{A}} \mathsf{cons}(\sigma_1;\sigma_2')}$$

(n-cons-val)
$$\frac{\sigma_1\ \mathtt{val}_{\mathcal{A}} \qquad \sigma_2\ \mathtt{val}_{\mathcal{A}}}{\mathsf{cons}(\sigma_1;\sigma_2)\ \mathtt{val}_{\mathcal{A}}}$$

(n-cons-err-prop-1)
$$\frac{\sigma_1\ \mathtt{err}_{\mathcal{A}}}{\mathsf{cons}(\sigma_1;\sigma_2)\ \mathtt{err}_{\mathcal{A}}}$$

(n-cons-err-prop-2)
$$\frac{\sigma_2\ \mathtt{err}_{\mathcal{A}}}{\mathsf{cons}(\sigma_1;\sigma_2)\ \mathtt{err}_{\mathcal{A}}}$$

(n-listrec-step)
$$\frac{\sigma \mapsto_{\mathcal{A}} \sigma'}{\mathsf{listrec}(\sigma;\sigma_1;\boldsymbol{x},\boldsymbol{y}.\sigma_2) \mapsto_{\mathcal{A}} \mathsf{listrec}(\sigma';\sigma_1;\boldsymbol{x},\boldsymbol{y}.\sigma_2)}$$

(n-listrec-elim-1)
$$\frac{}{\mathsf{listrec}(\mathsf{nil}[\kappa];\sigma_1;\boldsymbol{x},\boldsymbol{y}.\sigma_2) \mapsto_{\mathcal{A}} \sigma_1}$$

(n-listrec-elim-2)
$$\frac{\mathsf{cons}(\sigma_{\mathrm{hd}};\sigma_{\mathrm{tl}})\ \mathtt{val}_{\mathcal{A}}}{\mathsf{listrec}(\mathsf{cons}(\sigma_{\mathrm{hd}};\sigma_{\mathrm{tl}});\sigma_1;\boldsymbol{x},\boldsymbol{y}.\sigma_2) \mapsto_{\mathcal{A}} [\sigma_{\mathrm{head}}/\boldsymbol{x}, \mathsf{listrec}(\sigma_{\mathrm{tl}};\sigma_1;\boldsymbol{x},\boldsymbol{y}.\sigma_2)/\boldsymbol{y}]\sigma_2}$$

(n-listrec-err-prop)
$$\frac{\sigma\ \mathtt{err}_{\mathcal{A}}}{\mathsf{listrec}(\sigma;\sigma_1;\boldsymbol{x},\boldsymbol{y}.\sigma_2)\ \mathtt{err}_{\mathcal{A}}}$$

(n-triv-val)
$$\frac{}{()\ \mathtt{val}_{\mathcal{A}}}$$

(n-pair-step-1)
$$\frac{\sigma_1 \mapsto_{\mathcal{A}} \sigma_1'}{(\sigma_1,\sigma_2) \mapsto_{\mathcal{A}} (\sigma_1',\sigma_2)}$$

(n-pair-step-2)
$$\frac{\sigma_1\ \mathtt{val}_{\mathcal{A}} \qquad \sigma_2 \mapsto_{\sigma_2'}}{(\sigma_1,\sigma_2) \mapsto_{\mathcal{A}} (\sigma_1,\sigma_2')}$$

(n-pair-val)
$$\frac{\sigma_1\ \mathtt{val}_{\mathcal{A}} \qquad \sigma_2\ \mathtt{val}_{\mathcal{A}}}{(\sigma_1,\sigma_2)\ \mathtt{val}_{\mathcal{A}}}$$

(n-pair-err-prop-1)
$$\frac{\sigma_1\ \mathtt{err}_{\mathcal{A}}}{(\sigma_1,\sigma_2)\ \mathtt{err}_{\mathcal{A}}}$$

(n-pair-err-prop-2)
$$\frac{\sigma_2\ \mathtt{err}_{\mathcal{A}}}{(\sigma_1,\sigma_2)\ \mathtt{err}_{\mathcal{A}}}$$

(n-fst-step)
$$\frac{\sigma \mapsto_{\mathcal{A}} \sigma'}{\mathsf{fst}(\sigma) \mapsto_{\mathcal{A}} \mathsf{fst}(\sigma')}$$

(n-fst-elim)
$$\frac{(\sigma_1,\sigma_2)\ \mathtt{val}_{\mathcal{A}}}{\mathsf{fst}((\sigma_1,\sigma_2)) \mapsto_{\mathcal{A}} \sigma_1}$$

(n-fst-err-prop)
$$\frac{\sigma\ \mathtt{err}_{\mathcal{A}}}{\mathsf{fst}(\sigma)\ \mathtt{err}_{\mathcal{A}}}$$

(n-snd-step)
$$\frac{\sigma \mapsto_{\mathcal{A}} \sigma'}{\mathsf{snd}(\sigma) \mapsto_{\mathcal{A}} \mathsf{snd}(\sigma')}$$

(n-snd-elim)
$$\frac{(\sigma_1,\sigma_2)\ \mathtt{val}_{\mathcal{A}}}{\mathsf{snd}((\sigma_1,\sigma_2)) \mapsto_{\mathcal{A}} \sigma_2}$$

(n-snd-err-prop)
$$\frac{\sigma\ \mathtt{err}_{\mathcal{A}}}{\mathsf{snd}(\sigma)\ \mathtt{err}_{\mathcal{A}}}$$

(n-inl-step)
$$\frac{\sigma \mapsto_{\mathcal{A}} \sigma'}{\mathsf{inl}[\kappa](\sigma) \mapsto_{\mathcal{A}} \mathsf{inl}[\kappa](\sigma')}$$

(n-inl-val)
$$\frac{\sigma\ \mathtt{val}_{\mathcal{A}}}{\mathsf{inl}[\kappa](\sigma)\ \mathtt{val}_{\mathcal{A}}}$$

(n-inl-err-prop)
$$\frac{\sigma\ \mathtt{err}_{\mathcal{A}}}{\mathsf{inl}[\kappa](\sigma)\ \mathtt{err}_{\mathcal{A}}}$$

(n-inr-step)
$$\frac{\sigma \mapsto_{\mathcal{A}} \sigma'}{\mathsf{inr}[\kappa](\sigma) \mapsto_{\mathcal{A}} \mathsf{inr}[\kappa](\sigma')}$$

(n-inr-val)
$$\frac{\sigma\ \mathtt{val}_{\mathcal{A}}}{\mathsf{inr}[\kappa](\sigma)\ \mathtt{val}_{\mathcal{A}}}$$

(n-inr-err-prop)
$$\frac{\sigma\ \mathtt{err}_{\mathcal{A}}}{\mathsf{inr}[\kappa](\sigma)\ \mathtt{err}_{\mathcal{A}}}$$

(n-case-step)
$$\frac{\sigma \mapsto_{\mathcal{A}} \sigma'}{\mathsf{case}(\sigma;\boldsymbol{x}.\sigma_1;\boldsymbol{x}.\sigma_2) \mapsto_{\mathcal{A}} \mathsf{case}(\sigma';\boldsymbol{x}.\sigma_1;\boldsymbol{x}.\sigma_2)}$$

(n-case-elim-1)
$$\frac{\mathsf{inl}[\kappa](\sigma)\ \mathtt{val}_{\mathcal{A}}}{\mathsf{case}(\mathsf{inl}[\kappa](\sigma);\boldsymbol{x}.\sigma_1;\boldsymbol{x}.\sigma_2) \mapsto_{\mathcal{A}} [\sigma/\boldsymbol{x}]\sigma_1}$$

(n-case-elim-2)
$$\frac{\mathsf{inr}[\kappa](\sigma)\ \mathtt{val}_{\mathcal{A}}}{\mathsf{case}(\mathsf{inr}[\kappa](\sigma);\boldsymbol{x}.\sigma_1;\boldsymbol{x}.\sigma_2) \mapsto_{\mathcal{A}} [\sigma/\boldsymbol{x}]\sigma_2}$$

(n-case-err-prop)
$$\frac{\sigma\ \mathtt{err}_{\mathcal{A}}}{\mathsf{case}(\sigma;\boldsymbol{x}.\sigma_1;\boldsymbol{x}.\sigma_2)\ \mathtt{err}_{\mathcal{A}}}$$

(n-raise)
$$\frac{}{\mathsf{raise}[\kappa]\ \mathtt{err}_{\mathcal{A}}}$$

Figure 23: Static Dynamics - Core

$\boxed{\sigma \mapsto_{\mathcal{A}} \sigma'}$ $\boxed{\sigma\ \mathtt{val}_{\mathcal{A}}}$ $\boxed{\sigma\ \mathtt{err}_{\mathcal{A}}}$ (continued from Figure 23, continues into Figure 25)

(n-ty-step)
$$\frac{\sigma \mapsto_{\mathcal{A}} \sigma'}{c\langle\sigma\rangle \mapsto_{\mathcal{A}} c\langle\sigma'\rangle}$$

(n-ty-val)
$$\frac{\sigma\ \mathtt{val}_{\mathcal{A}}}{c\langle\sigma\rangle\ \mathtt{val}_{\mathcal{A}}}$$

(n-ty-err-prop)
$$\frac{\sigma\ \mathtt{err}_{\mathcal{A}}}{c\langle\sigma\rangle\ \mathtt{err}_{\mathcal{A}}}$$

(n-tycase-step)
$$\frac{\sigma \mapsto_{\mathcal{A}} \sigma'}{\mathsf{tycase}[c](\sigma; \boldsymbol{x}.\sigma_1; \sigma_2) \mapsto_{\mathcal{A}} \mathsf{tycase}[c](\sigma'; \boldsymbol{x}.\sigma_1; \sigma_2)}$$

(n-tycase-elim-1)
$$\frac{c\langle\sigma\rangle\ \mathtt{val}_{\mathcal{A}}}{\mathsf{tycase}[c](c\langle\sigma\rangle; \boldsymbol{x}.\sigma_1; \sigma_2) \mapsto_{\mathcal{A}} [\sigma/\boldsymbol{x}]\sigma_1}$$

(n-tycase-elim-2)
$$\frac{c'\langle\sigma\rangle\ \mathtt{val}_{\mathcal{A}} \qquad c \neq c'}{\mathsf{tycase}[c](c'\langle\sigma\rangle; \boldsymbol{x}.\sigma_1; \sigma_2) \mapsto_{\mathcal{A}} \sigma_2}$$

(n-tycase-err-prop)
$$\frac{\sigma\ \mathtt{err}_{\mathcal{A}}}{\mathsf{tycase}[c](\sigma; \boldsymbol{x}.\sigma_1; \sigma_2)\ \mathtt{err}_{\mathcal{A}}}$$

Figure 24: Static Dynamics - Types

$\boxed{\sigma \mapsto_{\mathcal{A}} \sigma'}$ $\boxed{\sigma\ \mathtt{val}_{\mathcal{A}}}$ $\boxed{\sigma\ \mathtt{err}_{\mathcal{A}}}$ (continued from Figure 24, continues into Figure 26)

(n-q-t-unquote-step)
$$\frac{\sigma \mapsto_{\mathcal{A}} \sigma'}{\blacktriangleright(\blacktriangleleft(\sigma)) \mapsto_{\mathcal{A}} \blacktriangleright(\blacktriangleleft(\sigma'))}$$

(n-q-t-unquote-elim)
$$\frac{\blacktriangleright(\hat{\tau})\ \mathtt{val}_{\mathcal{A}}}{\blacktriangleright(\blacktriangleleft(\blacktriangleright(\hat{\tau}))) \mapsto_{\mathcal{A}} \blacktriangleright(\hat{\tau})}$$

(n-q-t-unquote-err-prop)
$$\frac{\sigma\ \mathtt{err}_{\mathcal{A}}}{\blacktriangleright(\blacktriangleleft(\sigma))\ \mathtt{err}_{\mathcal{A}}}$$

(n-q-tytrans-step)
$$\frac{\sigma \mapsto_{\mathcal{A}} \sigma'}{\blacktriangleright(\mathsf{trans}(\sigma)) \mapsto_{\mathcal{A}} \blacktriangleright(\mathsf{trans}(\sigma'))}$$

(n-q-tytrans-val)
$$\frac{\sigma\ \mathtt{val}_{\mathcal{A}}}{\blacktriangleright(\mathsf{trans}(\sigma))\ \mathtt{val}_{\mathcal{A}}}$$

(n-q-tytrans-err-prop)
$$\frac{\sigma\ \mathtt{err}_{\mathcal{A}}}{\blacktriangleright(\mathsf{trans}(\sigma))\ \mathtt{err}_{\mathcal{A}}}$$

Figure 25: Static Dynamics - Quoted Translational Internal Types (Unquote and Type Translation References)

$\boxed{\sigma \mapsto_{\mathcal{A}} \sigma'}$ $\boxed{\sigma\ \texttt{val}_{\mathcal{A}}}$ $\boxed{\sigma\ \texttt{err}_{\mathcal{A}}}$ (continued from Figure 25, continues into 27)

(n-q-parr-step-1)
$$\frac{\blacktriangleright(\hat{\tau}_1) \mapsto_{\mathcal{A}} \blacktriangleright(\hat{\tau}_1')}{\blacktriangleright(\hat{\tau}_1 \rightharpoonup \hat{\tau}_2) \mapsto_{\mathcal{A}} \blacktriangleright(\hat{\tau}_1' \rightharpoonup \hat{\tau}_2)}$$

(n-q-parr-step-2)
$$\frac{\blacktriangleright(\hat{\tau}_1)\ \texttt{val}_{\mathcal{A}} \qquad \blacktriangleright(\hat{\tau}_2) \mapsto_{\mathcal{A}} \blacktriangleright(\hat{\tau}_2')}{\blacktriangleright(\hat{\tau}_1 \rightharpoonup \hat{\tau}_2) \mapsto_{\mathcal{A}} \blacktriangleright(\hat{\tau}_1 \rightharpoonup \hat{\tau}_2')}$$

(n-q-parr-val)
$$\frac{\blacktriangleright(\hat{\tau}_1)\ \texttt{val}_{\mathcal{A}} \qquad \blacktriangleright(\hat{\tau}_2)\ \texttt{val}_{\mathcal{A}}}{\blacktriangleright(\hat{\tau}_1 \rightharpoonup \hat{\tau}_2)\ \texttt{val}_{\mathcal{A}}}$$

(n-q-parr-err-prop-1)
$$\frac{\blacktriangleright(\hat{\tau}_1)\ \texttt{err}_{\mathcal{A}}}{\blacktriangleright(\hat{\tau}_1 \rightharpoonup \hat{\tau}_2)\ \texttt{err}_{\mathcal{A}}}$$

(n-q-parr-err-prop-2)
$$\frac{\blacktriangleright(\hat{\tau}_2)\ \texttt{err}_{\mathcal{A}}}{\blacktriangleright(\hat{\tau}_1 \rightharpoonup \hat{\tau}_2)\ \texttt{err}_{\mathcal{A}}}$$

(n-q-alpha-val)
$$\frac{}{\blacktriangleright(\alpha)\ \texttt{val}_{\mathcal{A}}}$$

(n-q-forall-step)
$$\frac{\blacktriangleright(\hat{\tau}) \mapsto_{\mathcal{A}} \blacktriangleright(\hat{\tau}')}{\blacktriangleright(\forall(\alpha.\hat{\tau})) \mapsto_{\mathcal{A}} \blacktriangleright(\forall(\alpha.\hat{\tau}))}$$

(n-q-forall-val)
$$\frac{\blacktriangleright(\hat{\tau})\ \texttt{val}_{\mathcal{A}}}{\blacktriangleright(\forall(\alpha.\hat{\tau}))\ \texttt{val}_{\mathcal{A}}}$$

(n-q-forall-err-prop)
$$\frac{\blacktriangleright(\hat{\tau})\ \texttt{err}_{\mathcal{A}}}{\blacktriangleright(\forall(\alpha.\hat{\tau}))\ \texttt{err}_{\mathcal{A}}}$$

(n-q-t-val)
$$\frac{}{\blacktriangleright(t)\ \texttt{val}_{\mathcal{A}}}$$

(n-q-mu-step)
$$\frac{\blacktriangleright(\hat{\tau}) \mapsto_{\mathcal{A}} \blacktriangleright(\hat{\tau}')}{\blacktriangleright(\mu(t.\hat{\tau})) \mapsto_{\mathcal{A}} \blacktriangleright(\mu(t.\hat{\tau}))}$$

(n-q-mu-val)
$$\frac{\blacktriangleright(\hat{\tau})\ \texttt{val}_{\mathcal{A}}}{\blacktriangleright(\mu(t.\hat{\tau}))\ \texttt{val}_{\mathcal{A}}}$$

(n-q-mu-err)
$$\frac{\blacktriangleright(\hat{\tau})\ \texttt{err}_{\mathcal{A}}}{\blacktriangleright(\mu(t.\hat{\tau}))\ \texttt{err}_{\mathcal{A}}}$$

(n-q-unit-val)
$$\frac{}{\blacktriangleright(())\ \texttt{val}_{\mathcal{A}}}$$

(n-q-prod-step-1)
$$\frac{\blacktriangleright(\hat{\tau}_1) \mapsto_{\mathcal{A}} \blacktriangleright(\hat{\tau}_1')}{\blacktriangleright(\hat{\tau}_1 \times \hat{\tau}_2) \mapsto_{\mathcal{A}} \blacktriangleright(\hat{\tau}_1' \times \hat{\tau}_2)}$$

(n-q-prod-step-2)
$$\frac{\blacktriangleright(\hat{\tau}_1)\ \texttt{val}_{\mathcal{A}} \qquad \blacktriangleright(\hat{\tau}_2) \mapsto_{\mathcal{A}} \blacktriangleright(\hat{\tau}_2')}{\blacktriangleright(\hat{\tau}_1 \times \hat{\tau}_2) \mapsto_{\mathcal{A}} \blacktriangleright(\hat{\tau}_1 \times \hat{\tau}_2')}$$

(n-q-prod-val)
$$\frac{\blacktriangleright(\hat{\tau}_1)\ \texttt{val}_{\mathcal{A}} \qquad \blacktriangleright(\hat{\tau}_2)\ \texttt{val}_{\mathcal{A}}}{\blacktriangleright(\hat{\tau}_1 \times \hat{\tau}_2)\ \texttt{val}_{\mathcal{A}}}$$

(n-q-prod-err-prop-1)
$$\frac{\blacktriangleright(\hat{\tau}_1)\ \texttt{err}_{\mathcal{A}}}{\blacktriangleright(\hat{\tau}_1 \times \hat{\tau}_2)\ \texttt{err}_{\mathcal{A}}}$$

(n-q-prod-err-prop-2)
$$\frac{\blacktriangleright(\hat{\tau}_2)\ \texttt{err}_{\mathcal{A}}}{\blacktriangleright((\hat{\tau}_1, \hat{\tau}_2))\ \texttt{err}_{\mathcal{A}}}$$

(n-q-sum-step-1)
$$\frac{\blacktriangleright(\hat{\tau}_1) \mapsto_{\mathcal{A}} \blacktriangleright(\hat{\tau}_1')}{\blacktriangleright(\hat{\tau}_1 + \hat{\tau}_2) \mapsto_{\mathcal{A}} \blacktriangleright(\hat{\tau}_1' + \hat{\tau}_2)}$$

(n-q-sum-step-2)
$$\frac{\blacktriangleright(\hat{\tau}_1)\ \texttt{val}_{\mathcal{A}} \qquad \blacktriangleright(\hat{\tau}_2) \mapsto_{\mathcal{A}} \blacktriangleright(\hat{\tau}_2')}{\blacktriangleright(\hat{\tau}_1 + \hat{\tau}_2) \mapsto_{\mathcal{A}} \blacktriangleright(\hat{\tau}_1 + \hat{\tau}_2')}$$

(n-q-sum-val)
$$\frac{\blacktriangleright(\hat{\tau}_1)\ \texttt{val}_{\mathcal{A}} \qquad \blacktriangleright(\hat{\tau}_2)\ \texttt{val}_{\mathcal{A}}}{\blacktriangleright(\hat{\tau}_1 + \hat{\tau}_2)\ \texttt{val}_{\mathcal{A}}}$$

(n-q-sum-err-prop-1)
$$\frac{\blacktriangleright(\hat{\tau}_1)\ \texttt{err}_{\mathcal{A}}}{\blacktriangleright(\hat{\tau}_1 + \hat{\tau}_2)\ \texttt{err}_{\mathcal{A}}}$$

(n-q-sum-err-prop-2)
$$\frac{\blacktriangleright(\hat{\tau}_2)\ \texttt{err}_{\mathcal{A}}}{\blacktriangleright(\hat{\tau}_1 + \hat{\tau}_2)\ \texttt{err}_{\mathcal{A}}}$$

Figure 26: Static Dynamics - Quoted Translational Internal Types (Shared Forms). All shared forms simply quote and then evaluate all sub-terms, proceeding left-to-right, and propagate errors.

$\boxed{\sigma \mapsto_{\mathcal{A}} \sigma'}$ $\boxed{\sigma\ \mathtt{val}_{\mathcal{A}}}$ $\boxed{\sigma\ \mathtt{err}_{\mathcal{A}}}$ (continued from Figure 26, continues into Figure 28)

(n-q-unquote-step)
$$\frac{\sigma \mapsto_{\mathcal{A}} \sigma'}{\rhd(\lhd(\sigma)) \mapsto_{\mathcal{A}} \rhd(\lhd(\sigma'))}$$

(n-q-unquote-elim)
$$\frac{\rhd(\hat{\tau})\ \mathtt{val}_{\mathcal{A}}}{\rhd(\lhd(\rhd(\hat{\iota}))) \mapsto_{\mathcal{A}} \rhd(\hat{\iota})}$$

(n-q-unquote-err-prop)
$$\frac{\sigma\ \mathtt{err}_{\mathcal{A}}}{\rhd(\lhd(\sigma))\ \mathtt{err}_{\mathcal{A}}}$$

(n-q-anatrans-step)
$$\frac{\sigma \mapsto_{\mathcal{A}} \sigma'}{\rhd(\mathsf{anatrans}[n](\sigma)) \mapsto_{\mathcal{A}} \rhd(\mathsf{anatrans}[n](\sigma'))}$$

(n-q-anatrans-val)
$$\frac{\sigma\ \mathtt{val}_{\overline{e};\Upsilon;\Phi} \qquad |\overline{e}| = n \qquad n' < n}{\rhd(\mathsf{anatrans}[n'](\sigma))\ \mathtt{val}_{\overline{e};\Upsilon;\Phi}}$$

(n-q-anatrans-err-prop)
$$\frac{\sigma\ \mathtt{err}_{\mathcal{A}}}{\rhd(\mathsf{anatrans}[n](\sigma))\ \mathtt{err}_{\mathcal{A}}}$$

(n-q-syntrans-val)
$$\frac{|\overline{e}| = n \qquad n' < n}{\rhd(\mathsf{syntrans}[n'])\ \mathtt{val}_{\mathcal{A}}}$$

Figure 27: Static Dynamics - Quoted Translational Internal Terms (Unquote and Argument Translation References)

$\boxed{\sigma \mapsto_{\mathcal{A}} \sigma'}$ $\boxed{\sigma\ \texttt{val}_{\mathcal{A}}}$ $\boxed{\sigma\ \texttt{err}_{\mathcal{A}}}$ (continued from Figure 27, continues into Figure 29)

(n-q-x-val)
$$\frac{}{\triangleright(x)\ \texttt{val}_{\mathcal{A}}}$$

(n-q-abs-step-1)
$$\frac{\blacktriangleright(\hat{\tau}) \mapsto_{\mathcal{A}} \blacktriangleright(\hat{\tau}')}{\triangleright(\lambda x{:}\hat{\tau}.\hat{\iota}) \mapsto_{\mathcal{A}} \triangleright(\lambda x{:}\hat{\tau}'.\hat{\iota})}$$

(n-q-abs-step-2)
$$\frac{\blacktriangleright(\hat{\tau})\ \texttt{val}_{\mathcal{A}} \qquad \triangleright(\hat{\iota}) \mapsto_{\mathcal{A}} \triangleright(\hat{\iota}')}{\triangleright(\lambda x{:}\hat{\tau}.\hat{\iota}) \mapsto_{\mathcal{A}} \triangleright(\lambda x{:}\hat{\tau}.\hat{\iota}')}$$

(n-q-abs-val)
$$\frac{\blacktriangleright(\hat{\tau})\ \texttt{val}_{\mathcal{A}} \qquad \triangleright(\hat{\iota})\ \texttt{val}_{\mathcal{A}}}{\triangleright(\lambda x{:}\hat{\tau}.\hat{\iota})\ \texttt{val}_{\mathcal{A}}}$$

(n-q-abs-err-prop-1)
$$\frac{\blacktriangleright(\hat{\tau})\ \texttt{err}_{\mathcal{A}}}{\triangleright(\lambda x{:}\hat{\tau}.\hat{\iota})\ \texttt{err}_{\mathcal{A}}}$$

(n-q-abs-err-prop-2)
$$\frac{\triangleright(\hat{\iota})\ \texttt{err}_{\mathcal{A}}}{\triangleright(\lambda x{:}\hat{\tau}.\hat{\iota})\ \texttt{err}_{\mathcal{A}}}$$

(n-q-ap-step-1)
$$\frac{\triangleright(\hat{\iota}_1) \mapsto_{\mathcal{A}} \triangleright(\hat{\iota}_1')}{\triangleright(\hat{\iota}_1(\hat{\iota}_2)) \mapsto_{\mathcal{A}} \triangleright(\hat{\iota}_1'(\hat{\iota}_2))}$$

(n-q-ap-step-2)
$$\frac{\triangleright(\hat{\iota}_1)\ \texttt{val}_{\mathcal{A}} \qquad \triangleright(\hat{\iota}_2) \mapsto_{\mathcal{A}} \triangleright(\hat{\iota}_2')}{\triangleright(\hat{\iota}_1(\hat{\iota}_2)) \mapsto_{\mathcal{A}} \triangleright(\hat{\iota}_1(\hat{\iota}_2'))}$$

(n-q-ap-val)
$$\frac{\triangleright(\hat{\iota}_1)\ \texttt{val}_{\mathcal{A}} \qquad \triangleright(\hat{\iota}_2)\ \texttt{val}_{\mathcal{A}}}{\triangleright(\hat{\iota}_1(\hat{\iota}_2))\ \texttt{val}_{\mathcal{A}}}$$

(n-q-ap-err-prop-1)
$$\frac{\triangleright(\hat{\iota}_1)\ \texttt{err}_{\mathcal{A}}}{\triangleright(\hat{\iota}_1(\hat{\iota}_2))\ \texttt{err}_{\mathcal{A}}}$$

(n-q-ap-err-prop-2)
$$\frac{\triangleright(\hat{\iota}_2)\ \texttt{err}_{\mathcal{A}}}{\triangleright(\hat{\iota}_1(\hat{\iota}_2))\ \texttt{err}_{\mathcal{A}}}$$

(n-q-fix-step-1)
$$\frac{\blacktriangleright(\hat{\tau}) \mapsto_{\mathcal{A}} \blacktriangleright(\hat{\tau}')}{\triangleright(\mathsf{fix}[\hat{\tau}](x.\hat{\iota})) \mapsto_{\mathcal{A}} \triangleright(\mathsf{fix}[\hat{\tau}'](x.\hat{\iota}))}$$

(n-q-fix-step-2)
$$\frac{\blacktriangleright(\hat{\tau})\ \texttt{val}_{\mathcal{A}} \qquad \triangleright(\hat{\iota}) \mapsto_{\mathcal{A}} \triangleright(\hat{\iota}')}{\triangleright(\mathsf{fix}[\hat{\tau}](x.\hat{\iota})) \mapsto_{\mathcal{A}} \triangleright(\lambda x{:}\hat{\tau}.\hat{\iota}')}$$

(n-q-fix-val)
$$\frac{\blacktriangleright(\hat{\tau})\ \texttt{val}_{\mathcal{A}} \qquad \triangleright(\hat{\iota})\ \texttt{val}_{\mathcal{A}}}{\triangleright(\mathsf{fix}[\hat{\tau}](x.\hat{\iota}))\ \texttt{val}_{\mathcal{A}}}$$

(n-q-fix-err-prop-1)
$$\frac{\blacktriangleright(\hat{\tau})\ \texttt{err}_{\mathcal{A}}}{\triangleright(\mathsf{fix}[\hat{\tau}](x.\hat{\iota}))\ \texttt{err}_{\mathcal{A}}}$$

(n-q-fix-err-prop-2)
$$\frac{\triangleright(\hat{\iota})\ \texttt{err}_{\mathcal{A}}}{\triangleright(\mathsf{fix}[\hat{\tau}](x.\hat{\iota}))\ \texttt{err}_{\mathcal{A}}}$$

(n-q-tabs-step)
$$\frac{\triangleright(\hat{\iota}) \mapsto_{\mathcal{A}} \triangleright(\hat{\iota}')}{\triangleright(\Lambda(\alpha.\hat{\iota})) \mapsto_{\mathcal{A}} \triangleright(\Lambda(\alpha.\hat{\iota}'))}$$

(n-q-tabs-val)
$$\frac{\triangleright(\hat{\iota})\ \texttt{val}_{\mathcal{A}}}{\triangleright(\Lambda(\alpha.\hat{\iota}))\ \texttt{val}_{\mathcal{A}}}$$

(n-q-tabs-err-prop)
$$\frac{\triangleright(\hat{\iota})\ \texttt{err}_{\mathcal{A}}}{\triangleright(\Lambda(\alpha.\hat{\iota}))\ \texttt{err}_{\mathcal{A}}}$$

(n-q-tap-step-1)
$$\frac{\triangleright(\hat{\iota}) \mapsto_{\mathcal{A}} \triangleright(\hat{\iota}')}{\triangleright(\hat{\iota}\texttt{[}\hat{\tau}\texttt{]}) \mapsto_{\mathcal{A}} \triangleright(\hat{\iota}'\texttt{[}\hat{\tau}\texttt{]})}$$

(n-q-tap-step-2)
$$\frac{\triangleright(\hat{\iota})\ \texttt{val}_{\mathcal{A}} \qquad \blacktriangleright(\hat{\tau}) \mapsto_{\mathcal{A}} \blacktriangleright(\hat{\tau}')}{\triangleright(\hat{\iota}\texttt{[}\hat{\tau}\texttt{]}) \mapsto_{\mathcal{A}} \triangleright(\hat{\iota}\texttt{[}\hat{\tau}'\texttt{]})}$$

(n-q-tap-val)
$$\frac{\triangleright(\hat{\iota})\ \texttt{val}_{\mathcal{A}} \qquad \blacktriangleright(\hat{\tau})\ \texttt{val}_{\mathcal{A}}}{\triangleright(\hat{\iota}\texttt{[}\hat{\tau}\texttt{]})\ \texttt{val}_{\mathcal{A}}}$$

(n-q-tap-err-prop-1)
$$\frac{\triangleright(\hat{\iota})\ \texttt{err}_{\mathcal{A}}}{\triangleright(\hat{\iota}\texttt{[}\hat{\tau}\texttt{]})\ \texttt{err}_{\mathcal{A}}}$$

(n-q-tap-err-prop-2)
$$\frac{\blacktriangleright(\hat{\tau})\ \texttt{err}_{\mathcal{A}}}{\triangleright(\hat{\iota}\texttt{[}\hat{\tau}\texttt{]})\ \texttt{err}_{\mathcal{A}}}$$

(n-q-fold-step-1)
$$\frac{\blacktriangleright(\hat{\tau}) \mapsto_{\mathcal{A}} \blacktriangleright(\hat{\tau}')}{\triangleright(\mathsf{fold}[t.\hat{\tau}](\hat{\iota})) \mapsto_{\mathcal{A}} \triangleright(\mathsf{fold}[t.\hat{\tau}'](\hat{\iota}))}$$

(n-q-fold-step-2)
$$\frac{\blacktriangleright(\hat{\tau})\ \texttt{val}_{\mathcal{A}} \qquad \triangleright(\hat{\iota}) \mapsto_{\mathcal{A}} \triangleright(\hat{\iota}')}{\triangleright(\mathsf{fold}[t.\hat{\tau}](\hat{\iota})) \mapsto_{\mathcal{A}} \triangleright(\mathsf{fold}[t.\hat{\tau}](\hat{\iota}'))}$$

(n-q-fold-val)
$$\frac{\blacktriangleright(\hat{\tau})\ \texttt{val}_{\mathcal{A}} \qquad \triangleright(\hat{\iota})\ \texttt{val}_{\mathcal{A}}}{\triangleright(\mathsf{fold}[t.\hat{\tau}](\hat{\iota}))\ \texttt{val}_{\mathcal{A}}}$$

(n-q-fold-err-prop-1)
$$\frac{\blacktriangleright(\hat{\tau})\ \texttt{err}_{\mathcal{A}}}{\triangleright(\mathsf{fold}[t.\hat{\tau}](\hat{\iota}))\ \texttt{err}_{\mathcal{A}}}$$

(n-q-fold-err-prop-2)
$$\frac{\triangleright(\hat{\iota})\ \texttt{err}_{\mathcal{A}}}{\triangleright(\mathsf{fold}[t.\hat{\tau}](\hat{\iota}))\ \texttt{err}_{\mathcal{A}}}$$

(n-q-unfold-step)
$$\frac{\triangleright(\hat{\iota}) \mapsto_{\mathcal{A}} \triangleright(\hat{\iota}')}{\triangleright(\mathsf{unfold}(\hat{\iota})) \mapsto_{\mathcal{A}} \triangleright(\mathsf{unfold}(\hat{\iota}'))}$$

(n-q-unfold-val)
$$\frac{\triangleright(\hat{\iota})\ \texttt{val}_{\mathcal{A}}}{\triangleright(\mathsf{unfold}(\hat{\iota}))\ \texttt{val}_{\mathcal{A}}}$$

(n-q-unfold-err-prop)
$$\frac{\triangleright(\hat{\iota})\ \texttt{err}_{\mathcal{A}}}{\triangleright(\mathsf{unfold}(\hat{\iota}))\ \texttt{err}_{\mathcal{A}}}$$

Figure 28: Static Dynamics - Quoted Translational Internal Terms (Shared Forms, 1 of 2)

$\boxed{\sigma \mapsto_{\mathcal{A}} \sigma'}$ $\boxed{\sigma\ \mathtt{val}_{\mathcal{A}}}$ $\boxed{\sigma\ \mathtt{err}_{\mathcal{A}}}$ (continued from Figure 28, continues into Figure 30)

(n-q-triv-val)

$$\frac{}{\triangleright(())\ \mathtt{val}_{\mathcal{A}}}$$

(n-q-pair-step-1)

$$\frac{\triangleright(\hat{\iota}_1) \mapsto_{\mathcal{A}} \triangleright(\hat{\iota}_1')}{\triangleright((\hat{\iota}_1, \hat{\iota}_2)) \mapsto_{\mathcal{A}} \triangleright(\hat{\iota}_1' \times \hat{\iota}_2)}$$

(n-q-pair-step-2)

$$\frac{\triangleright(\hat{\iota}_1)\ \mathtt{val}_{\mathcal{A}} \qquad \triangleright(\hat{\iota}_2) \mapsto_{\mathcal{A}} \triangleright(\hat{\iota}_2')}{\triangleright((\hat{\iota}_1, \hat{\iota}_2)) \mapsto_{\mathcal{A}} \triangleright((\hat{\iota}_1, \hat{\iota}_2'))}$$

(n-q-pair-val)

$$\frac{\triangleright(\hat{\iota}_1)\ \mathtt{val}_{\mathcal{A}} \qquad \triangleright(\hat{\iota}_2)\ \mathtt{val}_{\mathcal{A}}}{\triangleright((\hat{\iota}_1, \hat{\iota}_2))\ \mathtt{val}_{\mathcal{A}}}$$

(n-q-pair-err-prop-1)

$$\frac{\triangleright(\hat{\iota}_1)\ \mathtt{err}_{\mathcal{A}}}{\triangleright((\hat{\iota}_1, \hat{\iota}_2))\ \mathtt{err}_{\mathcal{A}}}$$

(n-q-pair-err-prop-2)

$$\frac{\triangleright(\hat{\iota}_2)\ \mathtt{err}_{\mathcal{A}}}{\triangleright((\hat{\iota}_1, \hat{\iota}_2))\ \mathtt{err}_{\mathcal{A}}}$$

(n-q-fst-step)

$$\frac{\triangleright(\hat{\iota}) \mapsto_{\mathcal{A}} \triangleright(\hat{\iota}')}{\triangleright(\mathsf{fst}(\hat{\iota})) \mapsto_{\mathcal{A}} \triangleright(\mathsf{fst}(\hat{\iota}'))}$$

(n-q-fst-val)

$$\frac{\triangleright(\hat{\iota})\ \mathtt{val}_{\mathcal{A}}}{\triangleright(\mathsf{fst}(\hat{\iota}))\ \mathtt{val}_{\mathcal{A}}}$$

(n-q-fst-err-prop)

$$\frac{\triangleright(\hat{\iota})\ \mathtt{err}_{\mathcal{A}}}{\triangleright(\mathsf{fst}(\hat{\iota}))\ \mathtt{err}_{\mathcal{A}}}$$

(n-q-snd-step)

$$\frac{\triangleright(\hat{\iota}) \mapsto_{\mathcal{A}} \triangleright(\hat{\iota}')}{\triangleright(\mathsf{snd}(\hat{\iota})) \mapsto_{\mathcal{A}} \triangleright(\mathsf{snd}(\hat{\iota}'))}$$

(n-q-snd-val)

$$\frac{\triangleright(\hat{\iota})\ \mathtt{val}_{\mathcal{A}}}{\triangleright(\mathsf{snd}(\hat{\iota}))\ \mathtt{val}_{\mathcal{A}}}$$

(n-q-snd-err-prop)

$$\frac{\triangleright(\hat{\iota})\ \mathtt{err}_{\mathcal{A}}}{\triangleright(\mathsf{snd}(\hat{\iota}))\ \mathtt{err}_{\mathcal{A}}}$$

(n-q-inl-step-1)

$$\frac{\blacktriangleright(\hat{\tau}) \mapsto_{\mathcal{A}} \blacktriangleright(\hat{\tau}')}{\triangleright(\mathsf{inl}[\hat{\tau}](\hat{\iota})) \mapsto_{\mathcal{A}} \triangleright(\mathsf{inl}[\hat{\tau}'](\hat{\iota}))}$$

(n-q-inl-step-2)

$$\frac{\blacktriangleright(\hat{\tau})\ \mathtt{val}_{\mathcal{A}} \qquad \triangleright(\hat{\iota}) \mapsto_{\mathcal{A}} \triangleright(\hat{\iota}')}{\triangleright(\mathsf{inl}[\hat{\tau}](\hat{\iota})) \mapsto_{\mathcal{A}} \triangleright(\mathsf{inl}[\hat{\tau}](\hat{\iota}'))}$$

(n-q-inl-val)

$$\frac{\blacktriangleright(\hat{\tau})\ \mathtt{val}_{\mathcal{A}} \qquad \triangleright(\hat{\iota})\ \mathtt{val}_{\mathcal{A}}}{\triangleright(\mathsf{inl}[\hat{\tau}](\hat{\iota}))\ \mathtt{val}_{\mathcal{A}}}$$

(n-q-inl-err-prop-1)

$$\frac{\blacktriangleright(\hat{\tau})\ \mathtt{err}_{\mathcal{A}}}{\triangleright(\mathsf{inl}[\hat{\tau}](\hat{\iota}))\ \mathtt{err}_{\mathcal{A}}}$$

(n-q-inl-err-prop-2)

$$\frac{\triangleright(\hat{\iota})\ \mathtt{err}_{\mathcal{A}}}{\triangleright(\mathsf{inl}[\hat{\tau}](\hat{\iota}))\ \mathtt{err}_{\mathcal{A}}}$$

(n-q-inr-step-1)

$$\frac{\blacktriangleright(\hat{\tau}) \mapsto_{\mathcal{A}} \blacktriangleright(\hat{\tau}')}{\triangleright(\mathsf{inr}[\hat{\tau}](\hat{\iota})) \mapsto_{\mathcal{A}} \triangleright(\mathsf{inr}[\hat{\tau}'](\hat{\iota}))}$$

(n-q-inr-step-2)

$$\frac{\blacktriangleright(\hat{\tau})\ \mathtt{val}_{\mathcal{A}} \qquad \triangleright(\hat{\iota}) \mapsto_{\mathcal{A}} \triangleright(\hat{\iota}')}{\triangleright(\mathsf{inr}[\hat{\tau}](\hat{\iota})) \mapsto_{\mathcal{A}} \triangleright(\mathsf{inr}[\hat{\tau}](\hat{\iota}'))}$$

(n-q-inr-val)

$$\frac{\blacktriangleright(\hat{\tau})\ \mathtt{val}_{\mathcal{A}} \qquad \triangleright(\hat{\iota})\ \mathtt{val}_{\mathcal{A}}}{\triangleright(\mathsf{inr}[\hat{\tau}](\hat{\iota}))\ \mathtt{val}_{\mathcal{A}}}$$

(n-q-inr-err-prop-1)

$$\frac{\blacktriangleright(\hat{\tau})\ \mathtt{err}_{\mathcal{A}}}{\triangleright(\mathsf{inr}[\hat{\tau}](\hat{\iota}))\ \mathtt{err}_{\mathcal{A}}}$$

(n-q-inr-err-prop-2)

$$\frac{\triangleright(\hat{\iota})\ \mathtt{err}_{\mathcal{A}}}{\triangleright(\mathsf{inr}[\hat{\tau}](\hat{\iota}))\ \mathtt{err}_{\mathcal{A}}}$$

(n-q-case-step-1)

$$\frac{\triangleright(\sigma) \mapsto_{\mathcal{A}} \triangleright(\sigma')}{\triangleright(\mathsf{case}(\sigma; x.\sigma_1; x.\sigma_2)) \mapsto_{\mathcal{A}} \triangleright(\mathsf{case}(\sigma'; x.\sigma_1; x.\sigma_2))}$$

(n-q-case-step-2)

$$\frac{\triangleright(\sigma)\ \mathtt{val}_{\mathcal{A}} \qquad \triangleright(\sigma_1) \mapsto_{\mathcal{A}} \triangleright(\sigma_1')}{\triangleright(\mathsf{case}(\sigma; x.\sigma_1; x.\sigma_2)) \mapsto_{\mathcal{A}} \triangleright(\mathsf{case}(\sigma; x.\sigma_1'; x.\sigma_2))}$$

(n-q-case-step-3)

$$\frac{\triangleright(\sigma)\ \mathtt{val}_{\mathcal{A}} \qquad \triangleright(\sigma_1)\ \mathtt{val}_{\mathcal{A}} \qquad \triangleright(\sigma_2) \mapsto_{\mathcal{A}} \triangleright(\sigma_2')}{\triangleright(\mathsf{case}(\sigma; x.\sigma_1; x.\sigma_2)) \mapsto_{\mathcal{A}} \triangleright(\mathsf{case}(\sigma; x.\sigma_1; x.\sigma_2'))}$$

(n-q-case-val)

$$\frac{\triangleright(\sigma)\ \mathtt{val}_{\mathcal{A}} \qquad \triangleright(\sigma_1)\ \mathtt{val}_{\mathcal{A}} \qquad \triangleright(\sigma_2)\ \mathtt{val}_{\mathcal{A}}}{\triangleright(\mathsf{case}(\sigma; x.\sigma_1; x.\sigma_2))\ \mathtt{val}_{\mathcal{A}}}$$

(n-q-case-err-prop-1)

$$\frac{\triangleright(\sigma)\ \mathtt{err}_{\mathcal{A}}}{\triangleright(\mathsf{case}(\sigma; x.\sigma_1; x.\sigma_2))\ \mathtt{err}_{\mathcal{A}}}$$

(n-q-case-err-prop-2)

$$\frac{\triangleright(\sigma_1)\ \mathtt{err}_{\mathcal{A}}}{\triangleright(\mathsf{case}(\sigma; x.\sigma_1; x.\sigma_2))\ \mathtt{err}_{\mathcal{A}}}$$

(n-q-case-err-prop-3)

$$\frac{\triangleright(\sigma_2)\ \mathtt{err}_{\mathcal{A}}}{\triangleright(\mathsf{case}(\sigma; x.\sigma_1; x.\sigma_2))\ \mathtt{err}_{\mathcal{A}}}$$

Figure 29: Static Dynamics - Quoted Translational Internal Terms (Shared Forms, 2 of 2)

$\boxed{\sigma \mapsto_{\mathcal{A}} \sigma'}$ $\boxed{\sigma\ \texttt{val}_{\mathcal{A}}}$ $\boxed{\sigma\ \texttt{err}_{\mathcal{A}}}$ (continued from Figure 29)

(n-ana-step)
$$\frac{\sigma \mapsto_{\mathcal{A}} \sigma'}{\mathsf{ana}[n](\sigma) \mapsto_{\mathcal{A}} \mathsf{ana}[n](\sigma)}$$

(n-ana-success)
$$\frac{\sigma\ \texttt{val}_{\overline{e};\Upsilon;\Phi} \qquad \mathsf{nth}[n](\overline{e}) = e \qquad \Upsilon \vdash_{\Phi} e \Leftarrow \sigma \rightsquigarrow \iota}{\mathsf{ana}[n](\sigma) \mapsto_{\overline{e};\Upsilon;\Phi} \rhd(\mathsf{anatrans}[n](\sigma))}$$

(n-ana-fail)
$$\frac{\sigma\ \texttt{val}_{\overline{e};\Upsilon;\Phi} \qquad \mathsf{nth}[n](\overline{e}) = e \qquad [\Upsilon \vdash_{\Phi} e \not\Leftarrow \sigma]}{\mathsf{ana}[n](\sigma)\ \texttt{err}_{\overline{e};\Upsilon;\Phi}}$$

(n-ana-err-prop)
$$\frac{\sigma\ \texttt{err}_{\mathcal{A}}}{\mathsf{ana}[n](\sigma)\ \texttt{err}_{\mathcal{A}}}$$

(n-syn-success)
$$\frac{\mathsf{nth}[n](\overline{e}) = e \qquad \Upsilon \vdash_{\Phi} e \Rightarrow \sigma \rightsquigarrow \iota}{\mathsf{syn}[n] \mapsto_{\overline{e};\Upsilon;\Phi} (\sigma, \rhd(\mathsf{syntrans}[n]))}$$

(n-syn-fail)
$$\frac{\mathsf{nth}[n](\overline{e}) = e \qquad [\Upsilon \vdash_{\Phi} e \not\Rightarrow]}{\mathsf{syn}[n]\ \texttt{err}_{\overline{e};\Upsilon;\Phi}}$$

Figure 30: Static Dynamics - SL-EL Interface. Bracketed terms can be omitted to form a dynamics that can non-deterministically raise an error, because we do not inductively define the negations of the external analysis and synthesis judgements here (though it would be straightforward to).

$\boxed{\sigma \mapsto^{*}_{\mathcal{A}} \sigma}$

(n-multi-refl)
$$\frac{}{\sigma \mapsto^{*}_{\mathcal{A}} \sigma}$$

(n-multi-step)
$$\frac{\sigma \mapsto^{*}_{\mathcal{A}} \sigma'' \qquad \sigma'' \mapsto_{\mathcal{A}} \sigma'}{\sigma \mapsto^{*}_{\mathcal{A}} \sigma'}$$

Figure 31: Static Multistep Dynamics

$\boxed{\Upsilon\ \texttt{ctx}_{\Phi}}$

(tctx-ok-emp)
$$\frac{}{\vdash_{\Phi} \emptyset}$$

(tctx-ok-ext)
$$\frac{\Upsilon\ \texttt{ctx}_{\Phi} \qquad \sigma\ \texttt{type}_{\Phi}}{\Upsilon, x : \sigma\ \texttt{ctx}_{\Phi}}$$

Figure 32: Typing Context Validity

$\boxed{\mathsf{args}(n) = \sigma}$

(mk-arg-interface)
$$\frac{\mathsf{args}(n, 0) = \sigma}{\mathsf{args}(n) = \sigma}$$

$\boxed{\mathsf{args}(n, m) = \sigma}$

(mk-arg-interface-nil)
$$\frac{}{\mathsf{args}(0, m) = \mathsf{nil}[\mathsf{Arg}]}$$

(mk-arg-interface-cons)
$$\frac{\mathsf{args}(n, m+1) = \sigma'_{\text{args}}}{\mathsf{args}(n+1, m) = \mathsf{cons}((\lambda \mathsf{Ty}{::}\boldsymbol{ty}.\mathsf{ana}[m](\boldsymbol{ty}), \lambda\_{::}1.\mathsf{syn}[m]); \sigma'_{\text{args}})}$$

Figure 33: Argument Interface List Generation. The auxiliary judgement counts up to n to ensure that the numbers are in order.

$$\boxed{\vdash_\Phi \sigma \; \texttt{type} \rightsquigarrow \tau}$$

(ty-trans)

$$\frac{\sigma \; \texttt{type}_\Phi \qquad \mathsf{trans}(\sigma) \parallel \emptyset \hookrightarrow^{c}_{\Phi} \tau \parallel \mathcal{D} \qquad \mathcal{D} \rightsquigarrow \delta : \Delta}{\vdash_\Phi \sigma \; \texttt{type} \rightsquigarrow [\delta]\tau}$$

Figure 34: Type Translation

$$\boxed{\vdash_\Phi \Upsilon \; \texttt{ctx} \rightsquigarrow \Gamma}$$

(ctx-trans-emp)

$$\frac{}{\vdash_\Phi \emptyset \; \texttt{ctx} \rightsquigarrow \emptyset}$$

(ctx-trans-ext)

$$\frac{\vdash_\Phi \Upsilon \; \texttt{ctx} \rightsquigarrow \Gamma \qquad \vdash_\Phi \sigma \; \texttt{type} \rightsquigarrow \tau}{\vdash_\Phi \Upsilon, x : \sigma \; \texttt{ctx} \rightsquigarrow \Gamma, x : \tau}$$

Figure 35: Typing Context Translation

$\boxed{\hat{\tau} \parallel \mathcal{D} \hookrightarrow^{c}_{\Phi} \tau \parallel \mathcal{D}}$

(abs-parr)

$$\frac{\mathsf{trans}(\sigma_1) \parallel \mathcal{D} \hookrightarrow^{c}_{\Phi} \tau_1 \parallel \mathcal{D}' \qquad \mathsf{trans}(\sigma_2) \parallel \mathcal{D}' \hookrightarrow^{c}_{\Phi} \tau_2 \parallel \mathcal{D}''}{\mathsf{trans}(\rightharpoonup\langle(\sigma_1, \sigma_2)\rangle) \parallel \mathcal{D} \hookrightarrow^{c}_{\Phi} \tau_1 \rightharpoonup \tau_2 \parallel \mathcal{D}''}$$

(abs-ext-delegated)

$$\frac{\begin{array}{c}\mathsf{tycon}\ \textsc{tc}\ \{\mathsf{trans} = \sigma_{\text{schema}}\ \mathsf{in}\ \omega\} \sim \psi \in \Phi \\ \sigma_{\text{schema}}(\sigma_{\text{tyidx}}) \Downarrow_{\cdot;\emptyset;\Phi} \blacktriangleright(\hat{\tau}) \qquad \hat{\tau} \parallel \mathcal{D} \hookrightarrow^{\textsc{tc}}_{\Phi} \tau \parallel \mathcal{D}' \qquad \mathcal{D}' \leadsto \delta : \Delta \qquad \Delta \vdash \tau\end{array}}{\mathsf{trans}(\textsc{tc}\langle\sigma_{\text{tyidx}}\rangle) \parallel \mathcal{D} \hookrightarrow^{\textsc{tc}}_{\Phi} \tau \parallel \mathcal{D}'}$$

(abs-ext-not-delegated-new)

$$\frac{\begin{array}{c}c \neq \textsc{tc} \qquad \textsc{tc}\langle\sigma_{\text{tyidx}}\rangle \notin \text{dom}(\mathcal{D}) \qquad \mathsf{tycon}\ \textsc{tc}\ \{\mathsf{trans} = \sigma_{\text{schema}}\ \mathsf{in}\ \omega\} \sim \psi \in \Phi \\ \sigma_{\text{schema}}(\sigma_{\text{tyidx}}) \Downarrow_{\cdot;\emptyset;\Phi} \blacktriangleright(\hat{\tau}) \qquad \hat{\tau} \parallel \mathcal{D} \hookrightarrow^{c}_{\Phi} \tau \parallel \mathcal{D}' \qquad \mathcal{D}' \leadsto \delta : \Delta \qquad \Delta \vdash \tau\end{array}}{\mathsf{trans}(\textsc{tc}\langle\sigma_{\text{tyidx}}\rangle) \parallel \mathcal{D} \hookrightarrow^{c}_{\Phi} \alpha \parallel \mathcal{D}', \textsc{tc}\langle\sigma_{\text{tyidx}}\rangle \leftrightarrow [\delta]\tau/\alpha}$$

(abs-other-delegated)

$$\frac{\hat{\tau} \parallel \mathcal{D} \hookrightarrow^{\mathsf{other}[m;\kappa]}_{\Phi} \tau \parallel \mathcal{D}' \qquad \mathcal{D}' \leadsto \delta : \Delta \qquad \Delta \vdash \tau}{\mathsf{trans}(\mathsf{other}[m;\kappa]\langle(\sigma_{\text{data}}, \blacktriangleright(\hat{\tau}))\rangle) \parallel \mathcal{D} \hookrightarrow^{\mathsf{other}[m;\kappa]}_{\Phi} \tau \parallel \mathcal{D}'}$$

(abs-other-not-delegated-new)

$$\frac{\begin{array}{c}c \neq \mathsf{other}[m;\kappa] \qquad \mathsf{other}[m;\kappa]\langle(\sigma_{\text{data}}, \blacktriangleright(\hat{\tau}))\rangle \notin \text{dom}(\mathcal{D}) \\ \hat{\tau} \parallel \mathcal{D} \hookrightarrow^{c}_{\Phi} \tau \parallel \mathcal{D}' \qquad \mathcal{D}' \leadsto \delta : \Delta \qquad \Delta \vdash \tau\end{array}}{\mathsf{trans}(\mathsf{other}[m;\kappa]\langle(\sigma_{\text{data}}, \blacktriangleright(\hat{\tau}))\rangle) \parallel \mathcal{D} \hookrightarrow^{c}_{\Phi} \alpha \parallel \mathcal{D}', \mathsf{other}[m;\kappa]\langle(\sigma_{\text{data}}, \blacktriangleright(\hat{\tau}))\rangle \leftrightarrow [\delta]\tau/\alpha}$$

(abs-ty-stored)

$$\frac{\sigma \leftrightarrow \tau/\alpha \in \mathcal{D}}{\mathsf{trans}(\sigma) \parallel \mathcal{D} \hookrightarrow^{c}_{\Phi} \alpha \parallel \mathcal{D}}$$

(abs-i-parr)

$$\frac{\hat{\tau}_1 \parallel \mathcal{D} \hookrightarrow^{c}_{\Phi} \tau_1 \parallel \mathcal{D}' \qquad \hat{\tau}_2 \parallel \mathcal{D}' \hookrightarrow^{c}_{\Phi} \tau_2 \parallel \mathcal{D}''}{\hat{\tau}_1 \rightharpoonup \hat{\tau}_2 \parallel \mathcal{D} \hookrightarrow^{c}_{\Phi} \tau_1 \rightharpoonup \tau_2 \parallel \mathcal{D}''}$$

(abs-i-alpha)

$$\frac{}{\alpha \parallel \mathcal{D} \hookrightarrow^{c}_{\Phi} \alpha \parallel \mathcal{D}}$$

(abs-i-forall)

$$\frac{\hat{\tau} \parallel \mathcal{D} \hookrightarrow^{c}_{\Phi} \tau \parallel \mathcal{D}'}{\forall(\alpha.\hat{\tau}) \parallel \mathcal{D} \hookrightarrow^{c}_{\Phi} \forall(\alpha.\tau) \parallel \mathcal{D}'}$$

(abs-i-t)

$$\frac{}{t \parallel \mathcal{D} \hookrightarrow^{c}_{\Phi} t \parallel \mathcal{D}}$$

(abs-i-mu)

$$\frac{\hat{\tau} \parallel \mathcal{D} \hookrightarrow^{c}_{\Phi} \tau \parallel \mathcal{D}'}{\mu(t.\hat{\tau}) \parallel \mathcal{D} \hookrightarrow^{c}_{\Phi} \mu(t.\tau) \parallel \mathcal{D}'}$$

(abs-i-unit)

$$\frac{}{1 \parallel \mathcal{D} \hookrightarrow^{c}_{\Phi} 1 \parallel \mathcal{D}}$$

(abs-i-prod)

$$\frac{\hat{\tau}_1 \parallel \mathcal{D} \hookrightarrow^{c}_{\Phi} \tau_1 \parallel \mathcal{D}' \qquad \hat{\tau}_2 \parallel \mathcal{D}' \hookrightarrow^{c}_{\Phi} \tau_2 \parallel \mathcal{D}''}{\hat{\tau}_1 \times \hat{\tau}_2 \parallel \mathcal{D} \hookrightarrow^{c}_{\Phi} \tau_1 \times \tau_2 \parallel \mathcal{D}''}$$

(abs-i-sum)

$$\frac{\hat{\tau}_1 \parallel \mathcal{D} \hookrightarrow^{c}_{\Phi} \tau_1 \parallel \mathcal{D}' \qquad \hat{\tau}_2 \parallel \mathcal{D}' \hookrightarrow^{c}_{\Phi} \tau_2 \parallel \mathcal{D}''}{\hat{\tau}_1 + \hat{\tau}_2 \parallel \mathcal{D} \hookrightarrow^{c}_{\Phi} \tau_1 + \tau_2 \parallel \mathcal{D}''}$$

Figure 36: Selectively Abstracted Type Translations

$$\boxed{\vdash^{c}_{\Phi} \mathcal{D}}$$

(tyts-ok-emp)

$$\frac{}{\vdash^{c}_{\Phi} \emptyset}$$

(tyts-ok-ext)

$$\frac{\vdash^{c}_{\Phi} \mathcal{D} \qquad c \neq c' \qquad \vdash_{\Phi} c'\langle \sigma_{\text{tyidx}} \rangle \; \mathtt{type} \leadsto \tau}{\vdash^{c}_{\Phi} \mathcal{D}, c'\langle \sigma_{\text{tyidx}} \rangle \leftrightarrow \tau/\alpha}$$

Figure 37: Type Translation Store Validity

$$\boxed{\mathcal{D} \leadsto \delta : \Delta}$$

(ty-store-int-emp)

$$\frac{}{\emptyset \leadsto \emptyset : \emptyset}$$

(ty-store-int-ext)

$$\frac{\mathcal{D} \leadsto \delta : \Delta}{(\mathcal{D}, \sigma \leftrightarrow \tau/\alpha) \leadsto (\delta, \tau/\alpha) : (\Delta, \alpha)}$$

Figure 38: Type Translation Store Internalization

$$\boxed{\vdash^{c}_{\mathcal{A}} \mathcal{D}\, \mathcal{G}}$$

(tmts-ok-emp)

$$\frac{\vdash^{c}_{\Phi} \mathcal{D}}{\vdash^{c}_{\overline{e};\Upsilon;\Phi} \mathcal{D}\, \emptyset}$$

(tmts-ok-ext)

$$\frac{\begin{array}{c} \vdash^{c}_{\Phi} \mathcal{D} \qquad \mathsf{trans}(\sigma) \parallel \mathcal{D} \looparrowright^{c}_{\Phi} \tau \parallel \mathcal{D} \\ \mathcal{D} \leadsto \delta : \Delta \qquad \vdash_{\Phi} \sigma \; \mathtt{type} \leadsto [\delta]\tau \qquad \mathsf{nth}[n](\overline{e}) = e \qquad \Upsilon \vdash_{\Phi} e \Leftarrow \sigma \leadsto \iota \end{array}}{\vdash^{c}_{\overline{e};\Upsilon;\Phi} \mathcal{D}\, \mathcal{G}, n : \sigma \leadsto \iota/x : \tau}$$

Figure 39: Term Translation Store Validity

$$\boxed{\mathcal{G} \leadsto \gamma : \Gamma}$$

(arg-store-int-emp)

$$\frac{}{\emptyset \leadsto \emptyset : \emptyset}$$

(arg-store-int-ext)

$$\frac{\mathcal{G} \leadsto \gamma : \Gamma}{(\mathcal{G}, n : \sigma \leadsto \iota/x : \tau) \leadsto (\gamma, \iota/x) : (\Gamma, x : \tau)}$$

Figure 40: Term Translation Store Internalization

$\boxed{\hat{\iota} \parallel \mathcal{D}\ \mathcal{G} \hookrightarrow^{c}_{\mathcal{A}} \iota \parallel \mathcal{D}\ \mathcal{G}}$ (continues in Figure 42)

(abs-anatrans-new)

$$\frac{n \notin \mathsf{dom}(\mathcal{G}) \qquad \mathsf{nth}[n](\overline{e}) = e \qquad \Upsilon \vdash_{\Phi} e \Leftarrow \sigma \rightsquigarrow \iota \qquad \mathsf{trans}(\sigma) \parallel \mathcal{D} \hookrightarrow^{c}_{\Phi} \tau \parallel \mathcal{D}'}{\mathsf{anatrans}[n](\sigma) \parallel \mathcal{D}\ \mathcal{G} \hookrightarrow^{c}_{\overline{e};\Upsilon;\Phi} x \parallel \mathcal{D}'\ \mathcal{G}, n : \sigma \rightsquigarrow \iota/x : \tau}$$

(abs-anatrans-stored)

$$\frac{n : \sigma \rightsquigarrow \iota/x : \tau \in \mathcal{G}}{\mathsf{anatrans}[n](\sigma) \parallel \mathcal{D}\ \mathcal{G} \hookrightarrow^{c}_{\mathcal{A}} x \parallel \mathcal{D}\ \mathcal{G}}$$

(abs-syntrans-new)

$$\frac{n \notin \mathsf{dom}(\mathcal{G}) \qquad \mathsf{nth}[n](\overline{e}) = e \qquad \Upsilon \vdash_{\Phi} e \Rightarrow \sigma \rightsquigarrow \iota \qquad \mathsf{trans}(\sigma) \parallel \mathcal{D} \hookrightarrow^{c}_{\Phi} \tau \parallel \mathcal{D}'}{\mathsf{syntrans}[n] \parallel \mathcal{D}\ \mathcal{G} \hookrightarrow^{c}_{\overline{e};\Upsilon;\Phi} \mathcal{D}' \parallel \mathcal{G}, n : \sigma \rightsquigarrow \iota/x : \tau}$$

(abs-syntrans-stored)

$$\frac{n : \sigma \rightsquigarrow \iota/x : \tau \in \mathcal{G}}{\mathsf{syntrans}[n] \parallel \mathcal{D}\ \mathcal{G} \hookrightarrow^{c}_{\mathcal{A}} x \parallel \mathcal{D}\ \mathcal{G}}$$

(abs-x)

$$\frac{}{x \parallel \mathcal{D}\ \mathcal{G} \hookrightarrow^{c}_{\mathcal{A}} x \parallel \mathcal{D}\ \mathcal{G}}$$

(abs-abs)

$$\frac{\hat{\tau} \parallel \mathcal{D} \hookrightarrow^{c}_{\Phi} \tau \parallel \mathcal{D}' \qquad \hat{\iota} \parallel \mathcal{D}'\ \mathcal{G} \hookrightarrow^{c}_{\overline{e};\Upsilon;\Phi} \tau \parallel \mathcal{D}''\ \mathcal{G}'}{\lambda x{:}\hat{\tau}.\hat{\iota} \parallel \mathcal{D}\ \mathcal{G} \hookrightarrow^{c}_{\overline{e};\Upsilon;\Phi} \lambda x{:}\tau.\iota \parallel \mathcal{D}''\ \mathcal{G}'}$$

(abs-ap)

$$\frac{\hat{\iota}_1 \parallel \mathcal{D}\ \mathcal{G} \hookrightarrow^{c}_{\mathcal{A}} \iota_1 \parallel \mathcal{D}'\ \mathcal{G}' \qquad \hat{\iota}_2 \parallel \mathcal{D}'\ \mathcal{G}' \hookrightarrow^{c}_{\mathcal{A}} \iota_2 \parallel \mathcal{D}''\ \mathcal{G}''}{\hat{\iota}_1(\hat{\iota}_2) \parallel \mathcal{D}\ \mathcal{G} \hookrightarrow^{c}_{\mathcal{A}} \iota_1(\iota_2) \parallel \mathcal{D}''\ \mathcal{G}''}$$

(abs-fix)

$$\frac{\hat{\tau} \parallel \mathcal{D} \hookrightarrow^{c}_{\Phi} \tau \parallel \mathcal{D}' \qquad \hat{\iota} \parallel \mathcal{D}'\ \mathcal{G} \hookrightarrow^{c}_{\overline{e};\Upsilon;\Phi} \tau \parallel \mathcal{D}''\ \mathcal{G}'}{\mathsf{fix}[\hat{\tau}](x.\hat{\iota}) \parallel \mathcal{D}\ \mathcal{G} \hookrightarrow^{c}_{\overline{e};\Upsilon;\Phi} \mathsf{fix}[\tau](x.\iota) \parallel \mathcal{D}''\ \mathcal{G}'}$$

(abs-tabs)

$$\frac{\hat{\iota} \parallel \mathcal{D}\ \mathcal{G} \hookrightarrow^{c}_{\mathcal{A}} \iota \parallel \mathcal{D}'\ \mathcal{G}'}{\Lambda(\alpha.\hat{\iota}) \parallel \mathcal{D}\ \mathcal{G} \hookrightarrow^{c}_{\mathcal{A}} \Lambda(\alpha.\iota) \parallel \mathcal{D}'\ \mathcal{G}'}$$

(abs-tap)

$$\frac{\hat{\iota} \parallel \mathcal{D}\ \mathcal{G} \hookrightarrow^{c}_{\overline{e};\Upsilon;\Phi} \iota \parallel \mathcal{D}'\ \mathcal{G}' \qquad \hat{\tau} \parallel \mathcal{D}' \hookrightarrow^{c}_{\Phi} \tau \parallel \mathcal{D}''}{\hat{\iota}\texttt{[}\hat{\tau}\texttt{]} \parallel \mathcal{D}\ \mathcal{G} \hookrightarrow^{c}_{\overline{e};\Upsilon;\Phi} \iota\texttt{[}\tau\texttt{]} \parallel \mathcal{D}''\ \mathcal{G}'}$$

(abs-fold)

$$\frac{\hat{\tau} \parallel \mathcal{D} \hookrightarrow^{c}_{\Phi} \tau \parallel \mathcal{D}' \qquad \hat{\iota} \parallel \mathcal{D}'\ \mathcal{G} \hookrightarrow^{c}_{\overline{e};\Upsilon;\Phi} \tau \parallel \mathcal{D}''\ \mathcal{G}'}{\mathsf{fold}[t.\hat{\tau}](\hat{\iota}) \parallel \mathcal{D}\ \mathcal{G} \hookrightarrow^{c}_{\overline{e};\Upsilon;\Phi} \mathsf{fold}[t.\tau](\iota) \parallel \mathcal{D}''\ \mathcal{G}'}$$

(abs-unfold)

$$\frac{\hat{\iota} \parallel \mathcal{D}\ \mathcal{G} \hookrightarrow^{c}_{\mathcal{A}} \iota \parallel \mathcal{D}'\ \mathcal{G}'}{\mathsf{unfold}(\hat{\iota}) \parallel \mathcal{D}\ \mathcal{G} \hookrightarrow^{c}_{\mathcal{A}} \mathsf{unfold}(\iota) \parallel \mathcal{D}'\ \mathcal{G}'}$$

Figure 41: Selectively Abstracted Term Translations (1 of 2)

$\boxed{\hat{\iota} \parallel \mathcal{D}\;\mathcal{G} \looparrowright^{c}_{\mathcal{A}} \iota \parallel \mathcal{D}\;\mathcal{G}}$ (continued from Figure 41)

(abs-triv)
$$\frac{}{() \parallel \mathcal{D}\;\mathcal{G} \looparrowright^{c}_{\mathcal{A}} () \parallel \mathcal{D}\;\mathcal{G}}$$

(abs-pair)
$$\frac{\hat{\iota}_1 \parallel \mathcal{D}\;\mathcal{G} \looparrowright^{c}_{\mathcal{A}} \iota_1 \parallel \mathcal{D}'\;\mathcal{G}' \qquad \hat{\iota}_2 \parallel \mathcal{D}'\;\mathcal{G}' \looparrowright^{c}_{\mathcal{A}} \iota_2 \parallel \mathcal{D}''\;\mathcal{G}''}{(\hat{\iota}_1, \hat{\iota}_2) \parallel \mathcal{D}\;\mathcal{G} \looparrowright^{c}_{\mathcal{A}} (\iota_1, \iota_2) \parallel \mathcal{D}''\;\mathcal{G}''}$$

(abs-fst)
$$\frac{\hat{\iota} \parallel \mathcal{D}\;\mathcal{G} \looparrowright^{c}_{\mathcal{A}} \iota \parallel \mathcal{D}'\;\mathcal{G}'}{\mathsf{fst}(\hat{\iota}) \parallel \mathcal{D}\;\mathcal{G} \looparrowright^{c}_{\mathcal{A}} \mathsf{fst}(\iota) \parallel \mathcal{D}'\;\mathcal{G}'}$$

(abs-snd)
$$\frac{\hat{\iota} \parallel \mathcal{D}\;\mathcal{G} \looparrowright^{c}_{\mathcal{A}} \iota \parallel \mathcal{D}'\;\mathcal{G}'}{\mathsf{snd}(\hat{\iota}) \parallel \mathcal{D}\;\mathcal{G} \looparrowright^{c}_{\mathcal{A}} \mathsf{snd}(\iota) \parallel \mathcal{D}'\;\mathcal{G}'}$$

(abs-inl)
$$\frac{\hat{\tau} \parallel \mathcal{D} \looparrowright^{c}_{\Phi} \tau \parallel \mathcal{D}' \qquad \hat{\iota} \parallel \mathcal{D}'\;\mathcal{G} \looparrowright^{c}_{\overline{e};\Upsilon;\Phi} \tau \parallel \mathcal{D}''\;\mathcal{G}'}{\mathsf{inl}[\hat{\tau}](\hat{\iota}) \parallel \mathcal{D}\;\mathcal{G} \looparrowright^{c}_{\overline{e};\Upsilon;\Phi} \mathsf{inl}[\tau](\iota) \parallel \mathcal{D}''\;\mathcal{G}'}$$

(abs-inr)
$$\frac{\hat{\tau} \parallel \mathcal{D} \looparrowright^{c}_{\Phi} \tau \parallel \mathcal{D}' \qquad \hat{\iota} \parallel \mathcal{D}'\;\mathcal{G} \looparrowright^{c}_{\overline{e};\Upsilon;\Phi} \tau \parallel \mathcal{D}''\;\mathcal{G}'}{\mathsf{inr}[\hat{\tau}](\hat{\iota}) \parallel \mathcal{D}\;\mathcal{G} \looparrowright^{c}_{\overline{e};\Upsilon;\Phi} \mathsf{inr}[\tau](\iota) \parallel \mathcal{D}''\;\mathcal{G}'}$$

(abs-case)
$$\frac{\hat{\iota} \parallel \mathcal{D}\;\mathcal{G} \looparrowright^{c}_{\mathcal{A}} \iota \parallel \mathcal{D}'\;\mathcal{G}' \qquad \hat{\iota}_1 \parallel \mathcal{D}'\;\mathcal{G}' \looparrowright^{c}_{\mathcal{A}} \iota_1 \parallel \mathcal{D}''\;\mathcal{G}'' \qquad \hat{\iota}_2 \parallel \mathcal{D}''\;\mathcal{G}'' \looparrowright^{c}_{\mathcal{A}} \iota_2 \parallel \mathcal{D}'''\;\mathcal{G}'''}{\mathsf{case}(\hat{\iota}; x.\hat{\iota}_1; x.\hat{\iota}_2) \parallel \mathcal{D}\;\mathcal{G} \looparrowright^{c}_{\mathcal{A}} \mathsf{case}(\iota; x.\iota_1; x.\iota_2) \parallel \mathcal{D}'''\;\mathcal{G}'''}$$

Figure 42: Selectively Abstracted Term Translations (2 of 2)

$\boxed{\vdash^{c}_{\mathcal{A}} \hat{\iota} : \sigma \rightsquigarrow \iota^{+}}$

(validate-tr)
$$\frac{\begin{array}{c}\mathsf{trans}(\sigma) \parallel \emptyset \looparrowright^{c}_{\Phi} \tau_{\text{abs}} \parallel \mathcal{D} \qquad \hat{\iota} \parallel \mathcal{D}\;\emptyset \looparrowright^{c}_{\overline{e};\Upsilon;\Phi} \iota_{\text{abs}} \parallel \mathcal{D}'\;\mathcal{G} \\ \mathcal{D}' \rightsquigarrow \delta : \Delta_{\text{abs}} \qquad \mathcal{G} \rightsquigarrow \gamma : \Gamma_{\text{abs}} \qquad \Delta_{\text{abs}}\;\Gamma_{\text{abs}} \vdash \iota_{\text{abs}} : \tau_{\text{abs}}\end{array}}{\vdash^{c}_{\overline{e};\Upsilon;\Phi} \hat{\iota} : \sigma \rightsquigarrow [\delta][\gamma]\iota_{\text{abs}}}$$

Figure 43: Translation Validation

$\boxed{\Upsilon \vdash_\Phi e \Leftarrow \sigma \leadsto \iota}$ $\boxed{\Upsilon \vdash_\Phi e \Rightarrow \sigma \leadsto \iota}$

(subsume)

$$\frac{\Upsilon \vdash_\Phi e \Rightarrow \sigma \leadsto \iota}{\Upsilon \vdash_\Phi e \Leftarrow \sigma \leadsto \iota}$$

(ascribe)

$$\frac{\begin{array}{c}\emptyset\,\emptyset \vdash^0_\Phi \sigma :: \mathsf{Ty} \qquad \sigma \Downarrow_{\cdot;\emptyset;\Phi} \sigma' \\ \Upsilon \vdash_\Phi e \Leftarrow \sigma' \leadsto \iota\end{array}}{\Upsilon \vdash_\Phi e : \sigma \Rightarrow \sigma' \leadsto \iota}$$

(syn-var)

$$\frac{x : \sigma \in \Upsilon}{\Upsilon \vdash_\Phi x \Rightarrow \sigma \leadsto x}$$

(ana-fix)

$$\frac{\begin{array}{c}\Upsilon, x : \sigma \vdash_\Phi e \Leftarrow \sigma \leadsto \iota \\ \vdash_\Phi \sigma \; \mathtt{type} \leadsto \tau\end{array}}{\Upsilon \vdash_\Phi \mathsf{fix}(x.e) \Leftarrow \sigma \leadsto \mathsf{fix}[\tau](x.\iota)}$$

(syn-lam)

$$\frac{\begin{array}{c}\emptyset\,\emptyset \vdash^0_\Phi \sigma_1 :: \mathsf{Ty} \qquad \sigma_1 \Downarrow_{\cdot;\emptyset;\Phi} \sigma_1' \\ \Upsilon, x : \sigma_1' \vdash_\Phi e \Rightarrow \sigma_2 \leadsto \iota \qquad \vdash_\Phi \sigma_1' \; \mathtt{type} \leadsto \tau_1\end{array}}{\Upsilon \vdash_\Phi \lambda x{:}\sigma_1.e \Rightarrow \rightharpoonup\langle(\sigma_1', \sigma_2)\rangle \leadsto \lambda x{:}\tau_1.\iota}$$

(ana-lam)

$$\frac{\begin{array}{c}\Upsilon, x : \sigma_1 \vdash_\Phi e \Leftarrow \sigma_2 \leadsto \iota \\ \vdash_\Phi \sigma_1 \; \mathtt{type} \leadsto \tau_1\end{array}}{\Upsilon \vdash_\Phi \lambda x.e \Leftarrow \rightharpoonup\langle(\sigma_1, \sigma_2)\rangle \leadsto \lambda x{:}\tau_1.\iota}$$

(syn-ap)

$$\frac{\Upsilon \vdash_\Phi e_1 \Rightarrow \rightharpoonup\langle(\sigma_1, \sigma_2)\rangle \leadsto \iota_1 \qquad \Upsilon \vdash_\Phi e_2 \Leftarrow \sigma_2 \leadsto \iota_2}{\Upsilon \vdash_\Phi e_1(e_2) \Rightarrow \sigma_2 \leadsto \iota_1(\iota_2)}$$

(ana-intro)

$$\frac{\begin{array}{c}\mathsf{tycon}\ \mathrm{TC}\ \{\mathsf{trans} = \sigma_{\text{schema}}\ \mathsf{in}\ \omega\} \sim \mathsf{tcsig}[\kappa_{\text{tyidx}}]\ \{\chi\} \in \Phi \\ \mathsf{intro}[\kappa_{\text{tmidx}}] \in \chi \qquad \emptyset\,\emptyset \vdash^0_\Phi \sigma_{\text{tmidx}} :: \kappa_{\text{tmidx}} \\ \mathsf{ana\ intro} = \sigma_{\text{def}} \in \omega \qquad |\overline{e}| = n \qquad \mathsf{args}(n) = \sigma_{\text{args}} \\ \sigma_{\text{def}}(\sigma_{\text{tyidx}})(\sigma_{\text{tmidx}})(\sigma_{\text{args}}) \Downarrow_{\overline{e};\Upsilon;\Phi} \rhd(\hat{\iota}) \\ \vdash^{\mathrm{TC}}_{\overline{e};\Upsilon;\Phi} \hat{\iota} : \mathrm{TC}\langle\sigma_{\text{tyidx}}\rangle \leadsto \iota\end{array}}{\Upsilon \vdash_\Phi \mathsf{intro}[\sigma_{\text{tmidx}}](\overline{e}) \Leftarrow \mathrm{TC}\langle\sigma_{\text{tyidx}}\rangle \leadsto \iota}$$

(syn-targ)

$$\frac{\begin{array}{c}\Upsilon \vdash_\Phi e_{\text{targ}} \Rightarrow \mathrm{TC}\langle\sigma_{\text{tyidx}}\rangle \leadsto \iota_{\text{targ}} \\ \mathsf{tycon}\ \mathrm{TC}\ \{\mathsf{trans} = \sigma_{\text{schema}}\ \mathsf{in}\ \omega\} \sim \mathsf{tcsig}[\kappa_{\text{tyidx}}]\ \{\chi\} \in \Phi \\ \mathbf{op}[\kappa_{\text{tmidx}}] \in \chi \qquad \emptyset\,\emptyset \vdash^0_\Phi \sigma_{\text{tmidx}} :: \kappa_{\text{tmidx}} \\ \mathsf{syn}\ \mathbf{op} = \sigma_{\text{def}} \in \omega \qquad |e_{\text{targ}}; \overline{e}| = n \qquad \mathsf{args}(n) = \sigma_{\text{args}} \\ \sigma_{\text{def}}(\sigma_{\text{tyidx}})(\sigma_{\text{tmidx}})(\sigma_{\text{args}}) \Downarrow_{(e_{\text{targ}};\overline{e});\Upsilon;\Phi} (\sigma, \rhd(\hat{\iota})) \\ \vdash^{\mathrm{TC}}_{(e_{\text{targ}};\overline{e});\Upsilon;\Phi} \hat{\iota} : \sigma \leadsto \iota\end{array}}{\Upsilon \vdash_\Phi \mathsf{targop}[\mathbf{op}; \sigma_{\text{tmidx}}](e_{\text{targ}}; \overline{e}) \Rightarrow \sigma \leadsto \iota}$$

(ana-intro-other)

$$\frac{|\overline{e}| = n \qquad \emptyset\,\emptyset \vdash^n_\Phi \rhd(\hat{\iota}) :: \mathsf{ITm} \qquad \rhd(\hat{\iota})\ \mathtt{val}_{\overline{e};\Upsilon;\Phi} \qquad \vdash^{\mathsf{other}[m;\kappa]}_{\overline{e};\Upsilon;\Phi} \hat{\iota} : \mathsf{other}[m;\kappa]\langle\sigma_{\text{tyidx}}\rangle \leadsto \iota}{\Upsilon \vdash_\Phi \mathsf{intro}[\rhd(\hat{\iota})](\overline{e}) \Leftarrow \mathsf{other}[m;\kappa]\langle\sigma_{\text{tyidx}}\rangle \leadsto \iota}$$

(syn-targ-other)

$$\frac{\begin{array}{c}\Upsilon \vdash_\Phi e_{\text{targ}} \Rightarrow \mathsf{other}[m;\kappa]\langle\sigma_{\text{tyidx}}\rangle \leadsto \iota_{\text{targ}} \\ |e_{\text{targ}}; \overline{e}| = n \qquad \emptyset\,\emptyset \vdash^n_\Phi \rhd(\hat{\iota}) :: \mathsf{ITm} \qquad \rhd(\hat{\iota})\ \mathtt{val}_{(e_{\text{targ}};\overline{e});\Upsilon;\Phi} \qquad \sigma\ \mathtt{type}_\Phi \qquad \vdash^{\mathsf{other}[m;\kappa]}_{(e_{\text{targ}};\overline{e});\Upsilon;\Phi} \hat{\iota} : \sigma \leadsto \iota\end{array}}{\Upsilon \vdash_\Phi \mathsf{targop}[\mathbf{op}; (\sigma, \rhd(\hat{\iota}))](e_{\text{targ}}; \overline{e}) \Rightarrow \sigma \leadsto \iota}$$

Figure 44: Typing

References

[1] R. Harper. *Practical Foundations for Programming Languages*. Cambridge University Press, 2012.

[2] R. Milner, M. Tofte, and R. Harper. *The Definition of Standard ML*. MIT Press, Aug. 1990.

[3] C. Omar, D. Kurilova, L. Nistor, B. Chung, A. Potanin, and J. Aldrich. Safely composable type-specific languages. In *ECOOP*, 2014.