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.

1 Internal Language

We assume the statics of the IL 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 $\iota\ \texttt{val}$. 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 $\iota'\ \texttt{val}$. 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.

We use $\mathcal{L}\{\rightharpoonup \forall\, \mu\, 1 \times +\}$, the syntax for which is shown in Figure 1, as representative of any intermediate language for a typed functional language.

internal types
$\tau \quad ::= \quad \tau \rightharpoonup \tau \mid \alpha \mid \forall(\alpha.\tau) \mid t \mid \mu(t.\tau) \mid 1 \mid \tau \times \tau \mid \tau + \tau$
internal terms
$\iota \quad ::= \quad x \mid \lambda[\tau](x.\iota) \mid \iota(\iota) \mid \mathsf{fix}[\tau](x.\iota) \mid \Lambda(\alpha.\iota) \mid \iota\texttt{[}\tau\texttt{]}$
$\qquad \mid \quad \mathsf{fold}[t.\tau](\iota) \mid \mathsf{unfold}(\iota) \mid () \mid (\iota, \iota) \mid \mathsf{fst}(\iota) \mid \mathsf{snd}(\iota)$
$\qquad \mid \quad \mathsf{inl}[\tau](\iota) \mid \mathsf{inr}[\tau](\iota) \mid \mathsf{case}(\iota; x.\iota; x.\iota)$
internal typing contexts $\Gamma ::= \emptyset \mid \Gamma, x : \tau$
internal type formation contexts $\Delta ::= \emptyset \mid \Delta, \alpha \mid \Delta, t$

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

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.

Lemma 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 of proving preservation and progress lemmas.

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

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

We assume substitution and simultaneous n-ary substitutions, δ and γ, have the standard semantics, and that typing and type formation contexts, Δ and Γ, obey standard properties of weakening, exchange and contraction. We assume that the names of variables and type variables are unimportant, so that α-equivalent terms, types and contexts can be identified implicitly throughout this work. In particular, we need the definitions of substitution validity shown in Figures 2 and 3 and the following lemmas.

$\boxed{\Delta \vdash \delta : \Delta}$ $\delta ::= \emptyset \mid \delta, \tau/\alpha$

$$\text{(tysub-emp)}\ \frac{}{\Delta \vdash \emptyset : \emptyset} \qquad \text{(tysub-ext)}\ \frac{\Delta \vdash \delta : \Delta' \qquad \Delta \vdash \tau}{\Delta \vdash \delta, \tau/\alpha : \Delta', \alpha}$$

Figure 2: Internal Type Substitution Validity

Lemma 4 (Internal Type Substitution on Types)**.** *If* $\Delta \vdash \delta : \Delta'$ *and* $\Delta\Delta' \vdash \tau$ *then* $\Delta \vdash [\delta]\tau$.

Lemma 5 (Internal Type Substitutions on Typing Contexts)**.** *If* $\Delta \vdash \delta : \Delta'$ *and* $\Delta\Delta' \vdash \Gamma$ *then* $\Delta \vdash [\delta]\Gamma$.

Lemma 6 (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* $\Delta\ [\delta]\Gamma \vdash [\delta]\iota : [\delta]\tau$.

$\boxed{\Delta\ \Gamma \vdash \gamma : \Gamma}$ $\gamma ::= \emptyset \mid \gamma, \iota/x$

$$\text{(tmsub-emp)}\ \frac{}{\Delta\ \Gamma \vdash \emptyset : \emptyset} \qquad \text{(tmsub-ext)}\ \frac{\Delta\ \Gamma \vdash \gamma : \Gamma' \qquad \Delta \vdash \tau \qquad \Delta\ \Gamma \vdash \iota : \tau}{\Delta\ \Gamma \vdash \gamma, \iota/x : \Gamma', x : \tau}$$

Figure 3: Internal Term Substitution Validity

Lemma 7 (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* $\Delta\ \Gamma \vdash [\gamma]\iota : \tau$.

2 Tycons

tycons	c	$::=$	$\rightharpoonup \mid \textsc{tc} \mid \mathsf{other}[m]$
tycon contexts	Φ	$::=$	$\cdot \mid \Phi, \mathsf{tycon}\ \textsc{tc}\ \{\theta\} \sim \psi$
tycon structures	θ	$::=$	$\mathsf{trans} = \sigma\ \mathsf{in}\ \omega$
opcon structures	ω	$::=$	$\mathsf{ana\ intro} = \sigma \mid \omega; \mathsf{syn}\ \mathbf{op} = \sigma$
tycon sigs	ψ	$::=$	$\mathsf{tcsig}[\kappa]\ \{\chi\}$
opcon sigs	χ	$::=$	$\mathsf{intro}[\kappa] \mid \chi; \mathbf{op}[\kappa]$

Figure 4: Syntax of tycons. Metavariables TC and **op** range over user-defined tycon and opcon names, respectively, and m ranges over natural numbers.

2.1 Tycon Contexts

Tycon contexts are ordered mappings from tycon names $\textsc{tc}$ to tycon definitions. The tycon context well-definedness judgement $\vdash \Phi$ is specified in Figure 5.

$\boxed{\vdash \Phi}$

$$\text{(tcc-emp)}\quad \frac{}{\vdash \cdot}$$

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

Figure 5: Tycon context well-definedness.

Opcon structures are checked against the tycon's signature.

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

$$\text{(ocstruct-intro)}\quad \frac{\begin{array}{c}\mathsf{intro}[\kappa_{\text{tmidx}}] \in \chi \qquad \emptyset \vdash \kappa_{\text{tmidx}} \\ \emptyset\,\emptyset \vdash^0_\Phi \sigma_{\text{def}} :: \kappa_{\text{tyidx}} \to \kappa_{\text{tmidx}} \to \mathsf{List}[\mathsf{Arg}] \to \mathsf{ITm}\end{array}}{\vdash_\Phi \mathsf{ana\ intro} = \sigma_{\text{def}} \sim \mathsf{tcsig}[\kappa_{\text{tyidx}}]\ \{\chi\}}$$

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

Figure 6: Checking opcon structures against tycon signatures.

For the purposes of the SL, only the type signatures are relevant. We thus define judgements $\vdash \psi$, $\vdash \chi$ and $\vdash \Phi$ `sigsok` in Figure 7 for checking this only.

The following lemmas characterize these judgements.

Lemma 8 (Inversion on Tycon Context Well-Definedness)**.** *If (1)*

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

then (a) $\vdash \Phi$ *and (b)* $\textsc{tc} \notin dom(\Phi)$ *and (c)* $\emptyset \vdash \kappa_{tyidx}\ \mathsf{eq}$ *and (d)* $\emptyset\,\emptyset \vdash^0_\Phi \sigma_{schema} :: \kappa_{tyidx} \to \mathsf{ITy}$ *and (e)* $\vdash_{\Phi,\mathsf{tycon}\ \textsc{tc}\ \{\mathsf{trans}=\sigma_{schema}\ \mathsf{in}\ \omega\}\sim\mathsf{tcsig}[\kappa_{tyidx}]\ \{\chi\}} \omega \sim \mathsf{tcsig}[\kappa_{tyidx}]\ \{\chi\}$.

Proof. Rule induction on assumption (1). Rule (tcc-emp) does not apply. Rule (tcc-ext) applies. The conclusions (a-e) are the five premises of (tcc-ext) and thus follow immediately. □

Lemma 9 (Tycon Context Composition)**.** *If (i)* $\vdash \Phi_1$ *and (ii)* $\vdash \Phi_2$ *and (iii)* $dom(\Phi_1) \cap dom(\Phi_2) = \emptyset$ *then (a)* $\vdash \Phi_1\Phi_2$.

$\boxed{\vdash \psi}$

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

$\boxed{\vdash \chi}$

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

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

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

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

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

Figure 7: Tycon and Opcon Signature Well-Formedness

Proof. By syntactic case analysis on Φ_1 and Φ_2.

If $\Phi_1 = \emptyset$ and $\Phi_2 = \emptyset$, then $\Phi_1\Phi_2 = \emptyset$ and (a) follows by (tcc-emp).

If $\Phi_2 = \emptyset$ then $\Phi_1\Phi_2 = \Phi_1$ and (a) follows by (1).

If $\Phi_1 = \emptyset$ then $\Phi_1\Phi_2 = \Phi_2$ and (a) follows by (2).

If $\Phi_1 = \Phi_1', \mathsf{tycon}\ \textsc{tc}_1\ \{\theta_1\} \sim \psi_1$ and $\Phi_2 = \Phi_2', \mathsf{tycon}\ \textsc{tc}_2\ \{\theta_2\} \sim \psi_2$ then $\Phi_1\Phi_2 = \Phi_1'\Phi_2', \mathsf{tycon}\ \textsc{tc}_1\ \{\theta_1\} \sim \psi_1, \mathsf{tycon}\ \textsc{tc}_2\ \{\theta_2\} \sim \psi_2$. Let (1-5) be the result of applying Lemma 8 on (i), and (6-10) be the result of applying Lemma 8 on (ii). By (iii) and the usual properties of domains of finite mappings, we have that (11) $\text{dom}(\Phi_1') \cap \text{dom}(\Phi_2') = \emptyset$ and further by (2) we have that (12) $\textsc{tc} \notin \text{dom}(\Phi_1'\Phi_2')$ and by (8) that (13) $\textsc{tc}' \notin \text{dom}(\Phi_1'\Phi_2')$. By the IH on (1), (6) and (11) we have that (14) $\vdash \Phi_1'\Phi_2'$. By (tcc-ext) on (14), (12), (3), (4) and on (5), we have that (15) $\vdash \Phi_1'\Phi_2', \mathsf{tycon}\ \textsc{tc}_1\ \{\theta_1\} \sim \psi_1$. By (tcc-ext) on (15), (13), (8), (9) and SAME on (10), we have (a). ☐

weakening of kinding and opcon well-definedness

Lemma 10 (Intro Opcon Existence and Well-Definedness)**.** *If (1)* $\vdash_\Phi \omega \sim \mathsf{tcsig}[\kappa_{tyidx}]\ \{\chi\}$ *then (a)* $\mathsf{intro}[\kappa_{tmidx}] \in \chi$ *and (b)* $\emptyset \vdash \kappa_{tmidx}$ *and (c)* $\mathsf{ana\ intro} = \sigma_{def} \in \omega$ *and (d)* $\emptyset\ \emptyset \vdash^0_\Phi \sigma_{def} :: \kappa_{tyidx} \to \kappa_{tmidx} \to \mathsf{List}[\mathsf{Arg}] \to \mathsf{ITm}$.

Proof. Rule induction on assumption (1). If rule (ocstruct-intro) applies then conclusion (c) follows syntactically and (a), (b) and (d) are the three premises and thus follow immediately. If rule (ocstruct-targ) applies, then we apply the IH to the first premise. ☐

Lemma 11 (Targeted Opcon Well-Definedness and Unicity)**.** *If (1)* $\vdash_\Phi \omega \sim \mathsf{tcsig}[\kappa_{tyidx}]\ \{\chi\}$ *and (2)* $\mathsf{syn}\ \boldsymbol{op} = \sigma_{def} \in \omega$ *then (a)* $\boldsymbol{op}[\kappa_{tmidx}] \in \chi$ *and (b)* $\emptyset \vdash \kappa_{tmidx}$ *and (c)* $\emptyset\ \emptyset \vdash^0_\Phi \sigma_{def} :: \kappa_{tyidx} \to \kappa_{tmidx} \to \mathsf{List}[\mathsf{Arg}] \to (\mathsf{Ty} \times \mathsf{ITm})$ *and (d) if* $\mathsf{syn}\ \boldsymbol{op} = \sigma'_{def} \in \omega$ *then* $\sigma'_{def} = \sigma_{def}$ *and (e) if* $\boldsymbol{op}[\kappa'_{tmidx}] \in \chi$ *then* $\kappa'_{tmidx} = \kappa_{tmidx}$.

Proof. Rule induction on assumption (1). Rule (ocstruct-intro) does not apply by assumption (2). Rule (ocstruct-targ) applies. Conclusion (a) follows syntactically. Conclusions (b) and (c) are premises 3 and 4. Conclusion (d) follows because either $\sigma'_{\text{def}} = \sigma_{\text{def}}$ (i.e. we considering the current definition) or we can apply the IH to premise 1 and the assumption of (d). Conclusion (e) follows because premise 2 checks the unicity condition directly. □

2.2 Full Examples

2.2.1 Regular Strings

2.2.2 Labeled Products

2.2.3 Records

3 Static Language

3.1 Kind Formation

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

$$\frac{\boldsymbol{\Delta} \vdash \kappa_1 \qquad \boldsymbol{\Delta} \vdash \kappa_2}{\boldsymbol{\Delta} \vdash \kappa_1 \to \kappa_2}\text{(kf-arrow)} \qquad \frac{\boldsymbol{\alpha} \in \boldsymbol{\Delta}}{\boldsymbol{\Delta} \vdash \boldsymbol{\alpha}}\text{(kf-alpha)} \qquad \frac{\boldsymbol{\Delta}, \boldsymbol{\alpha} \vdash \kappa}{\boldsymbol{\Delta} \vdash \forall(\boldsymbol{\alpha}.\kappa)}\text{(kf-forall)} \qquad \frac{\boldsymbol{k} \in \boldsymbol{\Delta}}{\boldsymbol{\Delta} \vdash \boldsymbol{k}}\text{(kf-k)} \qquad \frac{???}{\boldsymbol{\Delta} \vdash \mu_{\text{ind}}(\boldsymbol{k}.\kappa)}\text{(kf-ind)} \qquad \frac{}{\boldsymbol{\Delta} \vdash 1}\text{(kf-unit)}$$

$$\frac{\boldsymbol{\Delta} \vdash \kappa_1 \qquad \boldsymbol{\Delta} \vdash \kappa_2}{\boldsymbol{\Delta} \vdash \kappa_1 \times \kappa_2}\text{(kf-prod)} \qquad \frac{\boldsymbol{\Delta} \vdash \kappa_1 \qquad \boldsymbol{\Delta} \vdash \kappa_2}{\boldsymbol{\Delta} \vdash \kappa_1 + \kappa_2}\text{(kf-sum)} \qquad \frac{}{\boldsymbol{\Delta} \vdash \mathsf{Ty}}\text{(kf-ty)} \qquad \frac{}{\boldsymbol{\Delta} \vdash \mathsf{ITy}}\text{(kf-ity)} \qquad \frac{}{\boldsymbol{\Delta} \vdash \mathsf{ITm}}\text{(kf-itm)}$$

Figure 8: Kind Formation

Lemma 12 (Kind Variable Substitution - Kinds)**.**

1. *If* $\boldsymbol{\Delta}, \boldsymbol{\alpha} \vdash \kappa$ *and* $\boldsymbol{\Delta} \vdash \kappa'$ *then* $\boldsymbol{\Delta} \vdash [\kappa'/\boldsymbol{\alpha}]\kappa$.
2. *If* $\boldsymbol{\Delta}, \boldsymbol{k} \vdash \kappa$ *and* $\boldsymbol{\Delta} \vdash \kappa'$ *then* $\boldsymbol{\Delta} \vdash [\kappa'/\boldsymbol{k}]\kappa$.

???

Figure 9: Positive Kinds

3.2 Equality Kinds

Need an equational theory for SL to state equality kind property, but not important for other metatheory.

Lemma 13 (Equality Kind Well-Formedness)**.** *If* $\emptyset \vdash \kappa\ \mathsf{eq}$ *then* $\emptyset \vdash \kappa$.

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

$$\frac{}{\mathbf{\Delta} \vdash \emptyset}\ \text{(kctx-emp)} \qquad \frac{\mathbf{\Delta} \vdash \mathbf{\Gamma} \qquad \mathbf{\Delta} \vdash \kappa}{\mathbf{\Delta} \vdash \mathbf{\Gamma}, \boldsymbol{x} :: \kappa}\ \text{(kctx-ext)}$$

Figure 10: Kinding Context Formation

3.3 Kinding

3.4 Static Dynamics

Lemma 14 (Static Canonical Forms)**.** ...

Discussion of decidability and weak normalization.

3.5 Type Translations

3.6 Typing Context Translations

3.7 Arguments

Definition 1 (Argument Interface Kind)**.** *The kind abbreviated* Arg *is defined as*

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

3.8 Kind Safety

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)**.** *If* $\emptyset\ \emptyset \vdash_{\Phi}^{n} \sigma :: \kappa$ *and* $\vdash \Phi$ *and* $|\overline{e}| = n$ *and* $\vdash_{\Phi} \Upsilon \rightsquigarrow \Gamma$ *then* $\sigma\ \mathtt{val}_{\overline{e};\Upsilon;\Phi}$ *or* $\sigma\ \mathtt{err}_{\overline{e};\Upsilon;\Phi}$ *or* $\sigma \mapsto_{\overline{e};\Upsilon;\Phi} \sigma'$*.*

Theorem 2 (Static Preservation)**.** *If* $\emptyset\ \emptyset \vdash_{\Phi}^{n} \sigma :: \kappa$ *and* $\vdash \Phi$ *and* $|\overline{e}| = n$ *and* $\vdash_{\Phi} \Upsilon \rightsquigarrow \Gamma$ *and* $\sigma \mapsto_{\overline{e};\Upsilon;\Phi} \sigma'$ *then* $\emptyset\ \emptyset \vdash_{\Phi}^{n} \sigma' :: \kappa$*.*

The case in the proof of Theorem 2 for $\mathsf{syn}[n]$ requires that the following theorem be mutually defined. The mutual induction is well-founded because the total number of argument lists in the terms being considered decreases.

Theorem 3 (Type Synthesis)**.** *If* $\vdash \Phi$ *and* $\vdash_{\Phi} \Upsilon \rightsquigarrow \Gamma$ *and* $\Upsilon \vdash_{\Phi} e \Rightarrow \sigma \rightsquigarrow \iota$ *then* $\vdash_{\Phi} \sigma \rightsquigarrow \tau$ *(and thus* $\sigma\ \mathtt{type}_{\Phi}$*).*

4 External Language

4.1 Additional Desugarings

4.2 Typing

Unicity The rules are structured so that if a term is well-typed, both its type and translation are unique.

could move this whole thing to supplement if room needed

Theorem 4 (Unicity)**.** *If* $\vdash \Phi$ *and* $\vdash_\Phi \Upsilon \leadsto \Gamma$ *and* $\vdash_\Phi \sigma \leadsto \tau$ *and* $\vdash_\Phi \sigma' \leadsto \tau'$ *and* $\Upsilon \vdash_\Phi e \Leftarrow \sigma \leadsto \iota$ *and* $\Upsilon \vdash_\Phi e \Leftarrow \sigma' \leadsto \iota'$ *then* $\sigma = \sigma'$ *and* $\tau = \tau'$ *and* $\iota = \iota'$*.*

4.3 Proof of Regular String Soundness Tycon Invariant

References

[1] R. Harper. *Practical Foundations for Programming Languages*. Cambridge University Press, 2012.

A Appendix

(s-ty-step)
$$\frac{\sigma \mapsto_{\mathcal{A}} \sigma'}{c\langle\sigma\rangle \mapsto_{\mathcal{A}} c\langle\sigma'\rangle}$$

(s-ty-err)
$$\frac{\sigma\ \mathtt{err}_{\mathcal{A}}}{c\langle\sigma\rangle\ \mathtt{err}_{\mathcal{A}}}$$

(s-ty-v)
$$\frac{\sigma\ \mathtt{val}_{\mathcal{A}}}{c\langle\sigma\rangle\ \mathtt{val}_{\mathcal{A}}}$$

(s-otherty-v)
$$\frac{}{\mathsf{otherty}[m;\tau]\ \mathtt{val}_{\mathcal{A}}}$$

(s-tycase-step)
$$\frac{\sigma \mapsto_{\mathcal{A}} \sigma'}{\mathsf{tycase}[c](\sigma;\boldsymbol{x}.\sigma_1;\sigma_2) \mapsto_{\mathcal{A}} \mathsf{tycase}[c](\sigma';x.\sigma_1;\sigma_2)}$$

(s-tycase-err)
$$\frac{\sigma\ \mathtt{err}_{\mathcal{A}}}{\mathsf{tycase}[c](\sigma;\boldsymbol{x}.\sigma_1;\sigma_2)\ \mathtt{err}_{\mathcal{A}}}$$

(s-tycase-match)
$$\frac{c\langle\sigma\rangle\ \mathtt{val}_{\mathcal{A}}}{\mathsf{tycase}[c](c\langle\sigma\rangle;\boldsymbol{x}.\sigma_1;\sigma_2) \mapsto_{\mathcal{A}} [\sigma/x]\sigma_1}$$

(s-tycase-fail)
$$\frac{\sigma \neq c\langle\sigma'\rangle}{\mathsf{tycase}[c](\sigma;\boldsymbol{x}.\sigma_1;\sigma_2) \mapsto_{\mathcal{A}} \sigma_2}$$

(keq-k)
$$\frac{\boldsymbol{k} \in \boldsymbol{\Delta}}{\boldsymbol{\Delta} \vdash \boldsymbol{k}\ \mathsf{eq}}$$

(keq-ind)
$$\frac{\boldsymbol{\Delta},\boldsymbol{k} \vdash \kappa\ \mathsf{eq}}{\boldsymbol{\Delta} \vdash \mu_{\text{ind}}(\boldsymbol{k}.\kappa)\ \mathsf{eq}}$$

(keq-unit)
$$\frac{}{\boldsymbol{\Delta} \vdash 1\ \mathsf{eq}}$$

(keq-prod)
$$\frac{\boldsymbol{\Delta} \vdash \kappa_1\ \mathsf{eq} \quad \boldsymbol{\Delta} \vdash \kappa_2\ \mathsf{eq}}{\boldsymbol{\Delta} \vdash \kappa_1 \times \kappa_2\ \mathsf{eq}}$$

(keq-sum)
$$\frac{\boldsymbol{\Delta} \vdash \kappa_1\ \mathsf{eq} \quad \boldsymbol{\Delta} \vdash \kappa_2\ \mathsf{eq}}{\boldsymbol{\Delta} \vdash \kappa_1 + \kappa_2\ \mathsf{eq}}$$

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

(k-ity-alpha)
$$\frac{}{\boldsymbol{\Delta}\ \boldsymbol{\Gamma} \vdash^n_\Phi \blacktriangleright(\alpha) :: \mathsf{ITy}}$$

(s-ity-lam-step-1)
$$\frac{\blacktriangleright(\hat\tau_1) \mapsto_{\mathcal{A}} \blacktriangleright(\hat\tau_1')}{\blacktriangleright(\hat\tau_1 \times \hat\tau_2) \mapsto_{\mathcal{A}} \blacktriangleright(\hat\tau_1' \times \hat\tau_2)}$$

(s-ity-lam-step-2)
$$\frac{\blacktriangleright(\hat\tau_1)\ \mathtt{val}_{\mathcal{A}} \qquad \blacktriangleright(\hat\tau_2) \mapsto_{\mathcal{A}} \blacktriangleright(\hat\tau_2')}{\blacktriangleright(\hat\tau_1 \times \hat\tau_2) \mapsto_{\mathcal{A}} \blacktriangleright(\hat\tau_1 \times \hat\tau_2')}$$

(s-ity-lam-err-1)
$$\frac{\blacktriangleright(\hat\tau_1)\ \mathtt{err}_{\mathcal{A}}}{\blacktriangleright(\hat\tau_1 \times \hat\tau_2)\ \mathtt{err}_{\mathcal{A}}}$$

(s-ity-lam-err-2)
$$\frac{\blacktriangleright(\hat\tau_2)\ \mathtt{err}_{\mathcal{A}}}{\blacktriangleright(\hat\tau_1 \times \hat\tau_2)\ \mathtt{err}_{\mathcal{A}}}$$

(s-ity-lam-v)
$$\frac{\blacktriangleright(\hat\tau_1)\ \mathtt{val}_{\mathcal{A}} \qquad \blacktriangleright(\hat\tau_2)\ \mathtt{val}_{\mathcal{A}}}{\blacktriangleright(\hat\tau_1 \times \hat\tau_2)\ \mathtt{val}_{\mathcal{A}}}$$

(s-ity-alpha-v)
$$\frac{}{\blacktriangleright(\alpha)\ \mathtt{val}_{\mathcal{A}}}$$

(k-tycase-parr)
$$\frac{\boldsymbol{\Delta}\ \boldsymbol{\Gamma} \vdash^n_\Phi \sigma :: \mathsf{Ty} \qquad \boldsymbol{\Delta}\ \boldsymbol{\Gamma},\boldsymbol{x} :: \mathsf{Ty} \times \mathsf{Ty} \vdash^n_\Phi \sigma_1 :: \kappa \qquad \boldsymbol{\Delta}\ \boldsymbol{\Gamma} \vdash^n_\Phi \sigma_2 :: \kappa}{\boldsymbol{\Delta}\ \boldsymbol{\Gamma} \vdash^n_\Phi \mathsf{tycase}[\rightharpoonup](\sigma;\boldsymbol{x}.\sigma_1;\sigma_2) :: \kappa}$$

(s-ity-unquote-step)
$$\frac{\sigma \mapsto_{\mathcal{A}} \sigma'}{\blacktriangleright(\blacktriangleleft(\sigma)) \mapsto_{\mathcal{A}} \blacktriangleright(\blacktriangleleft(\sigma'))}$$

(s-ity-unquote-err)
$$\frac{\sigma\ \mathtt{err}_{\mathcal{A}}}{\blacktriangleright(\blacktriangleleft(\sigma))\ \mathtt{err}_{\mathcal{A}}}$$

(s-ity-trans-step)
$$\frac{\sigma \mapsto_{\mathcal{A}} \sigma'}{\blacktriangleright(\mathsf{trans}(\sigma)) \mapsto_{\mathcal{A}} \blacktriangleright(\mathsf{trans}(\sigma'))}$$

(s-ity-trans-err)
$$\frac{\sigma\ \mathtt{err}_{\mathcal{A}}}{\blacktriangleright(\mathsf{trans}(\sigma))\ \mathtt{err}_{\mathcal{A}}}$$

This judgement is defined by the following straightforward rules:

(tstore-emp)
$$\frac{}{\emptyset \rightsquigarrow \emptyset : \emptyset}$$

(tstore-ext)
$$\frac{\mathcal{D} \rightsquigarrow \delta : \Delta}{(\mathcal{D}, \sigma \leftrightarrow \tau/\alpha) \rightsquigarrow (\delta, \tau/\alpha) : (\Delta, \alpha)}$$

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

(s-itm-var-v)
$$\frac{}{\rhd(x)\ \mathtt{val}_{\mathcal{A}}}$$

(s-itm-lam-step-1)
$$\frac{\blacktriangleright(\hat{\tau}) \mapsto_{\mathcal{A}} \blacktriangleright(\hat{\tau}')}{\rhd(\lambda[\hat{\tau}](x.\hat{\iota})) \mapsto_{\mathcal{A}} \rhd(\lambda[\hat{\tau}'](x.\hat{\iota}))}$$

(s-itm-lam-step-2)
$$\frac{\blacktriangleright(\hat{\tau})\ \mathtt{val}_{\mathcal{A}} \qquad \rhd(\hat{\iota}) \mapsto_{\mathcal{A}} \rhd(\hat{\iota}')}{\rhd(\lambda[\hat{\tau}](x.\hat{\iota})) \mapsto_{\mathcal{A}} \rhd(\lambda[\hat{\tau}](x.\hat{\iota}'))}$$

(s-itm-lam-err-1)
$$\frac{\blacktriangleright(\hat{\tau})\ \mathtt{err}_{\mathcal{A}}}{\rhd(\lambda[\hat{\tau}](x.\hat{\iota}))\ \mathtt{err}_{\mathcal{A}}}$$

(s-itm-lam-err-2)
$$\frac{\rhd(\hat{\iota})\ \mathtt{err}_{\mathcal{A}}}{\rhd(\lambda[\hat{\tau}](x.\hat{\iota}))\ \mathtt{err}_{\mathcal{A}}}$$

(s-itm-lam-v)
$$\frac{\blacktriangleright(\hat{\tau})\ \mathtt{val}_{\mathcal{A}} \qquad \rhd(\hat{\iota})\ \mathtt{val}_{\mathcal{A}}}{\rhd(\lambda[\hat{\tau}](x.\hat{\iota}))\ \mathtt{val}_{\mathcal{A}}}$$

(k-itm-unquote)
$$\frac{\boldsymbol{\Delta}\ \boldsymbol{\Gamma} \vdash^n_\Phi \sigma :: \mathsf{ITm}}{\boldsymbol{\Delta}\ \boldsymbol{\Gamma} \vdash^n_\Phi \rhd(\lhd(\sigma)) :: \mathsf{ITm}}$$

(s-itm-unquote-step)
$$\frac{\sigma \mapsto_{\mathcal{A}} \sigma'}{\rhd(\lhd(\sigma)) \mapsto_{\mathcal{A}} \rhd(\lhd(\sigma'))}$$

(s-itm-unquote-err)
$$\frac{\sigma\ \mathtt{err}_{\mathcal{A}}}{\rhd(\lhd(\sigma))\ \mathtt{err}_{\mathcal{A}}}$$

(s-itm-unquote-elim)
$$\frac{\rhd(\hat{\iota})\ \mathtt{val}_{\mathcal{A}}}{\rhd(\lhd(\rhd(\hat{\iota}))) \mapsto_{\mathcal{A}} \rhd(\hat{\iota})}$$

(s-ana-step)
$$\frac{\sigma \mapsto_{\mathcal{A}} \sigma'}{\mathsf{ana}[n](\sigma) \mapsto_{\mathcal{A}} \mathsf{ana}[n](\sigma')}$$

(s-ana-err)
$$\frac{\sigma\ \mathtt{err}_{\mathcal{A}}}{\mathsf{ana}[n](\sigma)\ \mathtt{err}_{\mathcal{A}}}$$

(s-itm-anatrans-step)
$$\frac{\sigma \mapsto_{\mathcal{A}} \sigma'}{\rhd(\mathsf{anatrans}[n](\sigma)) \mapsto_{\mathcal{A}} \rhd(\mathsf{anatrans}[n](\sigma'))}$$

(s-itm-anatrans-err)
$$\frac{\sigma\ \mathtt{err}_{\mathcal{A}}}{\rhd(\mathsf{anatrans}[n](\sigma))\ \mathtt{err}_{\mathcal{A}}}$$

, as specified by the judgement $\mathcal{G} \rightsquigarrow \gamma : \Gamma$ defined by the following rules:

(ttrs-emp)
$$\frac{}{\emptyset \rightsquigarrow \emptyset : \emptyset}$$

(ttrs-ext)
$$\frac{\mathcal{G} \rightsquigarrow \gamma : \Gamma}{(\mathcal{G}, n : \sigma \rightsquigarrow \iota/x : \tau) \rightsquigarrow (\gamma, \iota/x) : (\Gamma, x : \tau)}$$

(k-itm-lam)
$$\frac{\mathbf{\Delta}\ \mathbf{\Gamma} \vdash_{\Phi}^{n} \blacktriangleright(\hat{\tau}) :: \mathsf{ITy} \qquad \mathbf{\Delta}\ \mathbf{\Gamma} \vdash_{\Phi}^{n} \triangleright(\hat{\iota}) :: \mathsf{ITm}}{\mathbf{\Delta}\ \mathbf{\Gamma} \vdash_{\Phi}^{n} \triangleright(\lambda[\hat{\tau}](x.\hat{\iota})) :: \mathsf{ITm}}$$

(k-raise)
$$\frac{\mathbf{\Delta} \vdash \kappa}{\mathbf{\Delta}\ \mathbf{\Gamma} \vdash_{\Phi}^{n} \mathsf{raise}[\kappa] :: \kappa}$$

(s-raise)
$$\frac{}{\mathsf{raise}[\kappa]\ \mathtt{err}_{\mathcal{A}}}$$

(s-syn-success)
$$\frac{\mathsf{nth}[n](\overline{e}) = e \qquad \Upsilon \vdash_{\Phi} e \Rightarrow \sigma \rightsquigarrow \iota}{\mathsf{syn}[n] \mapsto_{\overline{e};\Upsilon;\Phi} (\sigma, \triangleright(\mathsf{syntrans}[n]))}$$

(s-syn-fail)
$$\frac{\mathsf{nth}[n](\overline{e}) = e \qquad [\Upsilon \vdash_{\Phi} e \not\Rightarrow]}{\mathsf{syn}[n]\ \mathtt{err}_{\overline{e};\Upsilon;\Phi}}$$

(k-itm-syntrans)
$$\frac{n' < n}{\mathbf{\Delta}\ \mathbf{\Gamma} \vdash_{\Phi}^{n} \triangleright(\mathsf{syntrans}[n']) :: \mathsf{ITm}}$$

, e.g. for lambdas:

(abs-lam)
$$\frac{\hat{\tau} \parallel \mathcal{D} \hookrightarrow_{\Phi}^{\text{TC}} \tau \parallel \mathcal{D}' \qquad \hat{\iota} \parallel \mathcal{D}'\ \mathcal{G} \hookrightarrow_{\overline{e};\Upsilon;\Phi}^{\text{TC}} \iota \parallel \mathcal{D}''\ \mathcal{G}'}{\lambda[\hat{\tau}](x.\hat{\iota}) \parallel \mathcal{D}\ \mathcal{G} \hookrightarrow_{\overline{e};\Upsilon;\Phi}^{\text{TC}} \lambda[\tau](x.\iota) \parallel \mathcal{G}'\ \mathcal{D}''}$$

(abs-anatrans-stored)
$$\frac{n : \sigma \rightsquigarrow \iota / x : \tau \in \mathcal{G}}{\mathsf{anatrans}[n](\sigma) \parallel \mathcal{G}\ \mathcal{D} \hookrightarrow_{\mathcal{A}}^{\text{TC}} x \parallel \mathcal{G}\ \mathcal{D}}$$

(abs-syntrans-stored)
$$\frac{n : \sigma \rightsquigarrow \iota / x : \tau \in \mathcal{G}}{\mathsf{syntrans}[n] \parallel \mathcal{G}\ \mathcal{D} \hookrightarrow_{\mathcal{A}}^{\text{TC}} x \parallel \mathcal{G}\ \mathcal{D}}$$

(k-itm-anatrans)
$$\frac{n' < n \qquad \mathbf{\Delta}\ \mathbf{\Gamma} \vdash_{\Phi}^{n} \sigma :: \mathsf{Ty}}{\mathbf{\Delta}\ \mathbf{\Gamma} \vdash_{\Phi}^{n} \triangleright(\mathsf{anatrans}[n'](\sigma)) :: \mathsf{ITm}}$$

(abs-syntrans-new)
$$\frac{n \notin \text{dom}(\mathcal{G}) \qquad \mathsf{nth}[n](\overline{e}) = e \qquad \Upsilon \vdash_{\Phi} e \Rightarrow \sigma \rightsquigarrow \iota \qquad \mathsf{trans}(\sigma) \parallel \mathcal{D} \hookrightarrow_{\Phi}^{\text{TC}} \tau \parallel \mathcal{D}' \qquad (x \text{ fresh})}{\mathsf{syntrans}[n] \parallel \mathcal{G}\ \mathcal{D} \hookrightarrow_{\overline{e};\Upsilon;\Phi}^{\text{TC}} x \parallel \mathcal{G}, n : \sigma \rightsquigarrow \iota / x : \tau\ \mathcal{D}'}$$

(etctx-emp)
$$\frac{}{\vdash_{\Phi} \emptyset \rightsquigarrow \emptyset}$$

(etctx-ext)
$$\frac{\vdash_{\Phi} \Upsilon \rightsquigarrow \Gamma \qquad \sigma\ \mathtt{type}_{\Phi} \qquad \vdash_{\Phi} \sigma \rightsquigarrow \tau}{\vdash_{\Phi} \Upsilon, x \Rightarrow \sigma \rightsquigarrow \Gamma, x : \tau}$$

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

For example,

$$\text{(abs-prod)}\quad \frac{\hat{\tau}_1 \parallel \mathcal{D} \hookrightarrow^{\text{TC}}_{\Phi} \tau_1 \parallel \mathcal{D}' \qquad \hat{\tau}_2 \parallel \mathcal{D}' \hookrightarrow^{\text{TC}}_{\Phi} \tau_2 \parallel \mathcal{D}''}{\hat{\tau}_1 \times \hat{\tau}_2 \parallel \mathcal{D} \hookrightarrow^{\text{TC}}_{\Phi} \tau_1 \times \tau_2 \parallel \mathcal{D}''}$$

The argument interfaces that populate the list provided to opcon definitions is derived from the argument list by the judgement $\mathsf{args}(\overline{e}) =_n \sigma_{\text{args}}$, defined as follows:

$$\text{(args-z)}\quad \frac{}{\mathsf{args}(\cdot) =_0 \boldsymbol{nil}\,\texttt{[Arg]}} \qquad \text{(args-s)}\quad \frac{\mathsf{args}(\overline{e}) =_n \sigma}{\mathsf{args}(\overline{e}; e) =_{n+1} \boldsymbol{rcons}\,\texttt{[Arg]}\ \sigma\ (\lambda \boldsymbol{ty}{::}\mathsf{Ty}.\mathsf{ana}[n](\boldsymbol{ty}), \lambda\_{::}1.\mathsf{syn}[n])}$$

We assume that the definitions of the standard helper functions $\boldsymbol{nil} :: \forall(\boldsymbol{\alpha}.\mathsf{List}[\boldsymbol{\alpha}])$ and $\boldsymbol{rcons} :: \forall(\boldsymbol{\alpha}.\mathsf{List}[\boldsymbol{\alpha}] \to \boldsymbol{\alpha} \to \mathsf{List}[\boldsymbol{\alpha}])$, which adds an item to the end of a list, have been substituted into these rules. The result is that the nth element of the argument interface list simply wraps the static terms $\mathsf{ana}[n](\sigma)$ and $\mathsf{syn}[n]$.

Lemma 15. *If $\boldsymbol{\Delta}\ \boldsymbol{\Gamma} \vdash^{n'}_{\Phi} \sigma :: \kappa$ and $n > n'$ then $\boldsymbol{\Delta}\ \boldsymbol{\Gamma} \vdash^{n}_{\Phi} \sigma :: \kappa$.*