Functional Translation of a Calculus of Capabilities –Technical Appendix–

Arthur Charguéraud
INRIA
arthur.chargueraud@inria.fr

François Pottier
INRIA
francois.pottier@inria.fr

1. Source and target language

Figures 1 and 2 give the syntax and semantics of the imperative source language. Figures 3 and 4 give the syntax and semantics of the functional target language.

$$\begin{array}{lll} \textit{Values} & v & ::= x \mid () \mid \mathsf{inj}^i v \mid (v_1, v_2) \mid \mu f.\lambda x.t \mid p \mid l \\ \textit{Prim. ops.} & p & ::= \mathsf{case} \mid \mathsf{proj}^i \mid \mathsf{ref} \mid \mathsf{get} \mid \mathsf{set} \\ \textit{Terms} & t & ::= v \mid (v\,t) \end{array}$$

Figure 1. Source language syntax

$$\begin{array}{lll} ((\mu f.\lambda x.t)\,v) \,/\, s & \longrightarrow & ([f \to \mu f.\lambda x.t]\,[x \to v]\,t) \,/\, s \\ (\mathsf{case}\,((\mathsf{inj}^i\,v), v_1, v_2)) \,/\, s & \longrightarrow & (v_i\,v) \,/\, s \\ (\mathsf{proj}^i\,(v_1, v_2)) \,/\, s & \longrightarrow & v_i \,/\, s \\ (\mathsf{ref}\,v) \,/\, s & \longrightarrow & l \,/\, s \uplus [l \mapsto v] \\ (\mathsf{get}\,l) \,/\, s & \longrightarrow & s[l] \,/\, s \\ (\mathsf{set}\,(l, v)) \,/\, s & \longrightarrow & () \,/\, s[l \mapsto v] \\ (v\,t) \,/\, s & \longrightarrow & (v\,t') \,/\, s' \quad \text{if } (t \,/\, s \longrightarrow t \,/\, s') \end{array}$$

Figure 2. Source language semantics

$$\begin{array}{lll} \textit{Values} & w & ::= x \mid () \mid \mathsf{inj}^i\,w \mid (w_1, w_2) \mid \mu f.\lambda x.t \mid q \mid k \mid m \\ \textit{Prim. ops.} & q & ::= \mathsf{case} \mid \mathsf{proj}^i \mid \\ & & \quad \mathsf{map\_fresh} \mid \mathsf{map\_add} \mid \mathsf{map\_get} \mid \mathsf{map\_set} \\ \textit{Terms} & u & ::= w \mid (w\,u) \end{array}$$

Figure 3. Target language syntax

2. Syntax and kinds of types

Figure 5 gives the underlying syntax of typing entities, and Figure 6 define kinds on the top of this syntax. Figure 7 associates conventional metavariables to each kind. Figure 8 defines the translation of typing entities towards System F types and environments.

$$\begin{array}{lll} (\mu f.\lambda x.u)\,w & \longrightarrow & [f \to \mu f.\lambda x.u][x \to w]\,u \\ \mathsf{proj}^i\,(w_1, w_2) & \longrightarrow & w_i \\ \mathsf{case}\,(\mathsf{inj}^i\,w, w_1, w_2) & \longrightarrow & (w_i\,w) \\ \mathsf{map\_fresh}\,m & \longrightarrow & \min\{k \mid k \notin \mathrm{dom}(m)\} \\ \mathsf{map\_add}\,(m, k, w) & \longrightarrow & m \uplus [k \mapsto w] \\ \mathsf{map\_get}\,(m, k) & \longrightarrow & m[k] \\ \mathsf{map\_set}\,(m, k, w) & \longrightarrow & m[k \mapsto w] \\ (w\,u) & \longrightarrow & (w\,u') \quad \text{if } (u \longrightarrow u') \end{array}$$

Figure 4. Target language semantics

$$\begin{array}{lll} o & ::= & \alpha \mid \bot \mid \top \mid \mathsf{unit} \mid [o] \mid o + o \mid o \times o \mid o \to o \mid \mathsf{ref}\,o \mid \\ & & o * o \mid \forall\alpha.o \mid \exists\alpha.o \mid \mu\alpha.o \mid \emptyset \mid \{o : o\} \mid \{o : o \setminus o\} \mid \\ & & \varnothing \mid o,\, \alpha \mid o,\, x : o \\ \kappa & ::= & \textsc{val} \mid \textsc{mem} \mid \textsc{cmp} \mid \textsc{cap} \mid \textsc{sng} \mid \textsc{grp} \mid \textsc{dnv} \mid \textsc{lnv} \end{array}$$

Figure 5. Underlying syntax of types, capabilities and environments

$$\begin{array}{l} (\tau : \textsc{val})\,(\theta : \textsc{mem})\,(\chi : \textsc{cmp})\,(C : \textsc{cap}) \\ (\sigma : \textsc{sng})\,(\rho : \textsc{grp})\,(\Delta : \textsc{dnv})\,(\Gamma : \textsc{lnv}) \end{array}$$

Figure 7. Metavariables

3. Typing judgments

We first introduce structures that are used to state typing judgments.

- μ describes the shape of regions as follows:
  - μ maps each singleton region σ to a value v, which we write $v = \mu[\sigma]$.
  - μ maps each group region ρ to a map from keys to values, and we write $v = \mu[\rho][k]$.
- P stands for a set of locations and region names, and is used to describe the data posessed by a given capability.
- Σ stands for a set of subtyping hypotheses, so that each of its item is a proposition of the form $o_1 \le o_2 \rhd w$.

Figure 9 summarizes all the judgments involved. On gray background is the translation specific material. There are four subtyping relations, a typing judgment for source values and terms (mutually recursive), a typing judgment for memory values and capabilities

(mutually recursive), plus a typing judgment used to type-check an closed term when it is executed, and a typing judgment used to type-check closed values that programs output.

subtyping values	$\Sigma \vdash \tau_1 \leq \tau_2 \rhd w$
subtyping computations	$\Sigma \vdash \chi_1 \leq \chi_2 \rhd w$
subtyping memory values	$\Sigma \vdash \theta_1 \leq \theta_2 \rhd w$
subtyping capabilities	$\Sigma \vdash C_1 \leq C_2 \rhd w$
typing values	$\mu;\, \Delta \vdash v : \tau \rhd w$
typing terms	$\mu;\, \Gamma \Vdash t : \chi \rhd u$
typing memory values	$s;\, \mu \vdash v : \theta \measuredangle P \rhd w$
typing capabilities	$s;\, \mu \vdash C \measuredangle P \rhd w$
typing executed terms	$s;\, \mu;\, \bar{\alpha};\, P \Vdash\!\models t : \chi \rhd u$
typing output values	$s;\, \mu;\, \bar{\alpha};\, P \models v : \chi \rhd w$

Figure 9. Judgments

Light versions of subtyping judgments and of typing judgments are defined as follows.

$\tau_1 \leq \tau_2 \rhd w$ stands for $\forall \Sigma,\quad \Sigma \vdash \tau_1 \leq \tau_2 \rhd w$

$\Delta \vdash v : \tau \rhd w$ stands for $\forall \mu,\quad \mu;\, \Delta \vdash v : \tau \rhd w$

4. Typing source programs

Rules from figures 10 to 15 are used to type-check source programs.

In Figure 15, abbreviations are used to combine functions. They are defined below.

$$
\begin{array}{lcl}
\mathrm{id} & := & \lambda x.\, x \\
w_1 \circ w_2 & := & \lambda x.\, w_1\,(w_2\,x) \\
w_1 \times w_2 & := & \lambda(x_1, x_2).\, (w_1\,x_1, w_2\,x_2) \\
w_1 + w_2 & := & \lambda x.\, \mathrm{case}\,((\lambda x_1.\, \mathrm{inj}^1\,(w_1\,x_1)), \\
 & & \qquad (\lambda x_2.\, \mathrm{inj}^2\,(w_2\,x_2)), x)
\end{array}
$$

5. Typing runtime configurations

Figures 16 adds derivation rules for typing of values at runtime. Figures 17 and 18 define together the judgments for typing capabilities and values in memories. Figure 19 defines the judgments for typing executed terms, and Figure 20 the judgments for typing output values.

6. Derivable and admissible rules

Figure 21 gives rules that can be added safely to the system. They add no expressive power, but allow typing programs more conveniently (avoiding η and β expansions).

7. Examples

Examples are given through figures 22 to 25.

$$
\begin{array}{lcl}
\mathrm{mlist}\,\alpha & := & \mu\beta.\mathrm{ref}\,(\mathrm{unit} + \alpha \times \beta) \\
 & \rhd & \mu\beta.(\mathrm{unit} + \alpha \times \beta) \quad = \mathrm{list}\,\alpha \\
\\
\mathrm{nil} & : & \forall\alpha.\ \mathrm{unit} \to \exists\sigma.[\sigma] * \{\sigma : \mathrm{mlist}\,\alpha\} \\
 & \rhd & \forall\alpha.\ \mathrm{unit} \to \mathrm{unit} \times \mathrm{list}\,\alpha \\
 & := & \lambda().\mathrm{ref}\,(\mathrm{inj}^1\,()) \\
 & \rhd & \lambda().((), \mathrm{inj}^1\,()) \\
\\
\mathrm{cons} & : & \forall\alpha\,\sigma_1\,\sigma_2.\ [\sigma_1] \times [\sigma_2] * \{\sigma_1 : \alpha\} * \{\sigma_2 : \mathrm{mlist}\,\alpha\} \\
 & & \to \exists\sigma.[\sigma] * \{\sigma : \mathrm{mlist}\,\alpha\} \\
 & \rhd & \forall\alpha.\mathrm{unit} \times \mathrm{unit} \times \alpha \times \mathrm{list}\,\alpha \to \mathrm{unit} \times \mathrm{list}\,\alpha \\
 & := & \lambda(h,t).\mathrm{ref}\,(\mathrm{inj}^2\,(h,t)) \\
 & \rhd & \lambda((),(),h,t).((), (\mathrm{inj}^2\,(h,t))) \\
\\
\mathrm{reverse} & : & \forall\sigma\,\alpha.[\sigma] * \{\sigma : \mathrm{mlist}\,\alpha\} \to \exists\sigma'.[\sigma'] * \{\sigma' : \mathrm{mlist}\,\alpha\} \\
 & \rhd & \forall\alpha.\mathrm{unit} \times \mathrm{list}\,\alpha \to \mathrm{unit} \times \mathrm{list}\,\alpha \\
 & := & \mathrm{let\,aux} = \mu\,\mathrm{aux}.\lambda(l,p).\mathrm{match}\,(\mathrm{get}\,l)\ \mathrm{with} \\
 & & \quad |\,\mathrm{inj}^1\,()\ \Rightarrow\ p \\
 & & \quad |\,\mathrm{inj}^2\,(h,t)\ \Rightarrow\ \mathrm{set}\,(l, \mathrm{inj}^2\,(h,p))\,;\ \mathrm{aux}\,(t,l) \\
 & & \mathrm{in}\ \lambda l.\,(\mathrm{aux}\,(l, \mathrm{nil}\,())) \\
 & \rhd & \mathrm{let\,aux} = \mu\,\mathrm{aux}.\lambda((),(),l,p).\mathrm{match}\,l\ \mathrm{with} \\
 & & \quad |\,\mathrm{inj}^1\,()\ \Rightarrow\ ((),p) \\
 & & \quad |\,\mathrm{inj}^2\,(h,t)\ \Rightarrow \\
 & & \qquad \mathrm{let}\,l' = \mathrm{inj}^2\,(h,p)\ \mathrm{in} \\
 & & \qquad \mathrm{aux}\,((),(),t,l') \\
 & & \mathrm{in}\ \lambda((),l).\,(\mathrm{aux}\,((),(),l,\mathrm{nil}\,())) \\
\\
\mathrm{iter} & : & \forall\alpha\beta.\ (\forall\sigma.\ [\sigma] \to_{\{\sigma:\alpha\}*\beta} \mathrm{unit}) \\
 & & \quad \to (\forall\sigma.\ [\sigma] \to_{\{\sigma:\mathrm{mlist}\,\alpha\}*\beta} \mathrm{unit}) \\
 & \rhd & \forall\alpha\beta.\ (\mathrm{unit} \times \alpha \times \beta \to \mathrm{unit} \times \alpha \times \beta) \\
 & & \quad \to (\mathrm{unit} \times \mathrm{list}\,\alpha \times \beta \to \mathrm{unit} \times \mathrm{list}\,\alpha \times \beta) \\
 & := & \lambda f.\mu\,\mathrm{aux}.\lambda l.\mathrm{match}\,(\mathrm{get}\,l)\ \mathrm{with} \\
 & & \quad |\,\mathrm{inj}^1\,()\ \Rightarrow\ () \\
 & & \quad |\,\mathrm{inj}^2\,(h,t)\ \Rightarrow\ f\,h\,;\ \mathrm{aux}\,t \\
 & \rhd & \lambda f.\mu\,\mathrm{aux}.\lambda((),(e,l)).\mathrm{match}\,l\ \mathrm{with} \\
 & & \quad |\,\mathrm{inj}^1\,()\ \Rightarrow\ (e,l) \\
 & & \quad |\,\mathrm{inj}^2\,(h,t)\ \Rightarrow \\
 & & \qquad \mathrm{let}\,(e',h') = f\,((),(e,h))\ \mathrm{in} \\
 & & \qquad \mathrm{let}\,(e'',t') = \mathrm{aux}\,((),(e',t))\ \mathrm{in} \\
 & & \qquad (e'', \mathrm{inj}^2\,(h',t'))
\end{array}
$$

Figure 22. Example: mutable lists

$$\frac{o \,:\, \text{VAL}}{o \,:\, \text{CMP}} \qquad \frac{o \,:\, \text{VAL}}{o \,:\, \text{MEM}} \qquad \frac{}{\bot \,:\, \text{VAL}} \qquad \frac{}{\top \,:\, \text{VAL}} \qquad \frac{}{\text{unit} \,:\, \text{VAL}} \qquad \frac{o \,:\, \kappa}{[o] \,:\, \text{VAL}}\ \kappa\in\{\text{SNG,GRP}\}$$

$$\frac{o_1 \,:\, \kappa \qquad o_2 \,:\, \kappa}{(o_1 + o_2) \,:\, \kappa}\ \kappa\in\{\text{VAL,MEM}\} \qquad \frac{o_1 \,:\, \kappa \qquad o_2 \,:\, \kappa}{(o_1 \times o_2) \,:\, \kappa}\ \kappa\in\{\text{VAL,MEM}\} \qquad \frac{o_1 \,:\, \text{CMP} \qquad o_2 \,:\, \text{CMP}}{(o_1 \to o_2) \,:\, \text{VAL}} \qquad \frac{o \,:\, \text{MEM}}{(\text{ref}\, o) \,:\, \text{MEM}}$$

$$\frac{o_1 \,:\, \kappa \qquad o_2 \,:\, \text{CAP}}{(o_1 * o_2) \,:\, \kappa}\ \kappa\in\{\text{MEM,CAP,CMP}\} \qquad \frac{\alpha \text{ has kind } \kappa}{\alpha \,:\, \kappa} \qquad \frac{\alpha \,:\, \kappa_1 \qquad o \,:\, \kappa_2}{(\forall\alpha.o) \,:\, \kappa_2}\ \kappa_1\in\{\text{VAL,MEM,CAP,SNG,GRP}\}\ \kappa_2\in\{\text{VAL}\}$$

$$\frac{\alpha \,:\, \kappa_1 \qquad o \,:\, \kappa_2}{(\exists\alpha.o) \,:\, \kappa_2}\ \kappa_1\in\{\text{VAL,MEM,CAP,SNG,GRP}\}\ \kappa_2\in\{\text{VAL,MEM,CAP,CMP}\} \qquad \frac{\alpha \,:\, \kappa \qquad o \,:\, \kappa}{(\mu\alpha.o) \,:\, \kappa}\ \kappa\in\{\text{VAL,MEM,CAP}\}\ o \text{ not a variable or a } \mu \text{ form}$$

$$\frac{}{\emptyset \,:\, \text{CAP}} \qquad \frac{o_1 \,:\, \kappa \qquad o_2 \,:\, \text{MEM}}{\{o_1 : o_2\} \,:\, \text{CAP}}\ \kappa\in\{\text{SNG,GRP}\} \qquad \frac{o_1 \,:\, \text{GRP} \qquad o_2 \,:\, \text{MEM} \qquad o_3 \,:\, \text{SNG}}{\{o_1 : o_2 \setminus o_3\} \,:\, \text{CAP}}$$

$$\frac{}{\varnothing \,:\, \kappa}\ \kappa\in\{\text{DNV,LNV}\} \qquad \frac{o \,:\, \kappa_1 \qquad \alpha \,:\, \kappa_2}{(o,\, \alpha) \,:\, \kappa_1}\ \kappa_1\in\{\text{DNV,LNV}\}\ \kappa_2\in\{\text{VAL,MEM,CAP,SNG,GRP}\}\ \alpha\#o \qquad \frac{o_1 \,:\, \kappa_1 \qquad o_2 \,:\, \kappa_2}{(o_1,\, x : o_2) \,:\, \kappa_1}\ (\kappa_1\in\{\text{DNV}\} \wedge \kappa_2\in\{\text{VAL}\})\ \text{or}\ (\kappa_1\in\{\text{LNV}\} \wedge \kappa_2\in\{\text{VAL,CMP,CAP}\})\ x\#o_1 \wedge \text{fv}(o_2)\subseteq\text{dom}(o_1)$$

Figure 6. Kinds

$$\begin{array}{lcl} [\![\bot]\!] & = & \bot \\ [\![\top]\!] & = & \top \\ [\![\text{unit}]\!] & = & \text{unit} \\ [\![\,[\sigma]\,]\!] & = & \text{unit} \\ [\![\,[\rho]\,]\!] & = & \text{key} \end{array} \qquad \begin{array}{lcl} [\![o_1 + o_2]\!] & = & [\![o_1]\!] + [\![o_2]\!] \\ [\![o_1 \times o_2]\!] & = & [\![o_1]\!] \times [\![o_2]\!] \\ [\![o_1 \to o_2]\!] & = & [\![o_1]\!] \to [\![o_2]\!] \\ [\![\text{ref}\, o]\!] & = & [\![o]\!] \\ [\![o_1 * o_2]\!] & = & [\![o_1]\!] \times [\![o_2]\!] \end{array} \qquad \begin{array}{lcl} [\![\emptyset]\!] & = & \text{unit} \\ [\![\{\sigma : o\}]\!] & = & [\![o]\!] \\ [\![\{\rho : o\}]\!] & = & \text{map}\ [\![o]\!] \\ [\![\{\rho : o \setminus \sigma\}]\!] & = & \text{map}\ [\![o]\!] \times \text{key} \\ [\![\varnothing]\!] & = & \varnothing \\ [\![o_1,\, x : o_2]\!] & = & [\![o_1]\!],\, x : [\![o_2]\!] \end{array} \qquad \begin{array}{l} \text{If } (\alpha \,:\, \text{SNG}) \\ \text{or } (\alpha \,:\, \text{GRP}): \\ [\![\forall\alpha.o]\!] = [\![o]\!] \\ [\![\exists\alpha.o]\!] = [\![o]\!] \\ [\![\mu\alpha.o]\!] = [\![o]\!] \\ [\![o,\, \alpha]\!] = [\![o]\!] \end{array} \qquad \begin{array}{l} \text{Otherwise:} \\ [\![\alpha]\!] = \alpha \\ [\![\forall\alpha.o]\!] = \forall\alpha.[\![o]\!] \\ [\![\exists\alpha.o]\!] = \exists\alpha.[\![o]\!] \\ [\![\mu\alpha.o]\!] = \mu\alpha.[\![o]\!] \\ [\![o,\, \alpha]\!] = [\![o]\!],\, \alpha \end{array}$$

Figure 8. Translation of types, capabilities and environments

UNIT
$$\frac{}{\Delta \vdash () \,:\, \text{unit} \ \rhd\ ()}$$

INJ
$$\frac{\Delta \vdash v \,:\, \tau_i \ \rhd\ w}{\Delta \vdash (\text{inj}^i\, v) \,:\, (\tau_1 + \tau_2) \ \rhd\ (\text{inj}^i\, w)}$$

PAIR
$$\frac{\Delta \vdash v_1 \,:\, \tau_1 \ \rhd\ w_1 \qquad \Delta \vdash v_2 \,:\, \tau_2 \ \rhd\ w_2}{\Delta \vdash (v_1, v_2) \,:\, (\tau_1 \times \tau_2) \ \rhd\ (w_1, w_2)}$$

VAR
$$\frac{(x : \tau) \in \Delta}{\Delta \vdash x \,:\, \tau \ \rhd\ x}$$

FIX
$$\frac{\Delta,\, f : (\chi_1 \to \chi_2),\, x : \chi_1 \Vdash t \,:\, \chi_2 \ \rhd\ u}{\Delta \vdash (\mu f.\lambda x.t) \,:\, (\chi_1 \to \chi_2) \ \rhd\ (\mu f.\lambda x.u)}$$

Figure 10. Type-checking and type-directed translation: values

VAL
$$\frac{\Delta \vdash v \,:\, \tau \ \rhd\ w}{\Delta \Vdash v \,:\, \tau \ \rhd\ w}$$

APP
$$\frac{\Delta \Vdash v \,:\, (\chi_1 \to \chi_2) \ \rhd\ u_1 \qquad \Delta, \Gamma \Vdash t \,:\, \chi_1 \ \rhd\ u_2}{\Delta, \Gamma \Vdash (v\, t) \,:\, \chi_2 \ \rhd\ (u_1\, u_2)}$$

SUB
$$\frac{\Gamma \Vdash t \,:\, \chi_1 \ \rhd\ u \qquad \chi_1 \le \chi_2 \ \rhd\ w}{\Gamma \Vdash t \,:\, \chi_2 \ \rhd\ (w\, u)}$$

$*$-INTRO (FRAME)
$$\frac{\Gamma \Vdash t \,:\, \chi \ \rhd\ u}{\Gamma,\, (x : C) \Vdash t \,:\, (\chi * C) \ \rhd\ (u, x)}$$

$*$-ELIM
$$\frac{\Gamma,\, (x_1 : o),\, (x_2 : C) \Vdash t \,:\, \chi \ \rhd\ u}{\Gamma,\, x_1 : (o * C) \Vdash t \,:\, \chi \ \rhd\ \text{let}\,(x_1, x_2) = x_1\,\text{in}\,u}$$

Figure 11. Type-checking and type-directed translation: terms

$$\frac{\Delta,\ \alpha \vdash v : \tau \ \triangleright\ w}{\Delta \vdash v : (\forall\alpha.\tau) \ \triangleright\ w}\ \forall\text{-INTRO-VAL} \qquad \frac{\Delta \vdash v : (\forall\alpha.\tau) \ \triangleright\ w}{\Delta \vdash v : ([\alpha \to o]\,\tau) \ \triangleright\ w}\ \forall\text{-ELIM-VAL}$$

$$\frac{\Delta \vdash v : ([\alpha \to o]\,\tau) \ \triangleright\ w}{\Delta \vdash v : (\exists\alpha.\tau) \ \triangleright\ w}\ \exists\text{-INTRO-VAL} \qquad \frac{\Delta_1,\ \alpha,\ (x : \tau_1),\ \Delta_2 \vdash v : \tau \ \triangleright\ w}{\Delta_1,\ x : (\exists\alpha.\tau_1),\ \Delta_2 \vdash v : \tau \ \triangleright\ w}\ \exists\text{-ELIM-VAL}$$

$$\frac{\Gamma,\ \alpha \vdash t : \tau \ \triangleright\ u}{\Gamma \vdash t : (\forall\alpha.\tau) \ \triangleright\ u}\ \forall\text{-INTRO-TRM} \qquad \frac{\Gamma \Vdash t : (\forall\alpha.\tau) \ \triangleright\ u}{\Gamma \Vdash t : ([\alpha \to o]\,\tau) \ \triangleright\ u}\ \forall\text{-ELIM-TRM}$$

$$\frac{\Gamma \Vdash t : ([\alpha \to o]\,\chi) \ \triangleright\ u}{\Gamma \Vdash t : (\exists\alpha.\chi) \ \triangleright\ u}\ \exists\text{-INTRO-TRM} \qquad \frac{\Gamma_1,\ \alpha,\ (x : \chi_1),\ \Gamma_2 \Vdash t : \chi \ \triangleright\ u}{\Gamma_1,\ x : (\exists\alpha.\chi_1),\ \Gamma_2 \Vdash t : \chi \ \triangleright\ u}\ \exists\text{-ELIM-TRM}$$

Figure 12. Additional rules: introduction and elimination for quantifiers

ref	:	$\tau \to \exists\sigma.[\sigma] * \{\sigma : \mathrm{ref}\,\tau\}$	$\triangleright$	$\lambda x.\,((), x)$
get	:	$[\sigma] * \{\sigma : \mathrm{ref}\,\tau\} \to \tau * \{\sigma : \mathrm{ref}\,\tau\}$	$\triangleright$	$\lambda((), x).\,(x, x)$
set	:	$([\sigma] \times \tau_2) * \{\sigma : \mathrm{ref}\,\tau_1\} \to \mathrm{unit} * \{\sigma : \mathrm{ref}\,\tau_2\}$	$\triangleright$	$\lambda((), x_2, x_1).\,((), x_2)$
proj[1]	:	$[\sigma] * \{\sigma : \tau_1 \times \theta_2\} \to \tau_1 * \{\sigma : \tau_1 \times \theta_2\}$	$\triangleright$	$\lambda((), (x_1, x_2)).\,(x_1, (x_1, x_2))$
case	:	$\big(((\exists\sigma_1.([\sigma_1] * \{\sigma : [\sigma_1] + \bot\} * \{\sigma_1 : \theta_1\} * C)) \to \chi)$ $\times ((\exists\sigma_2.([\sigma_2] * \{\sigma : \bot + [\sigma_2]\} * \{\sigma_2 : \theta_2\} * C)) \to \chi)$ $\times [\sigma]\big) * \{\sigma : \theta_1 + \theta_2\} * C \to \chi$	$\triangleright$	$\lambda(f_1, f_2, (), x, c).\,\mathrm{case}\,($ $(\lambda x_1.\,(f_1\,((), \mathrm{inj}^1(), x_1, c))),$ $(\lambda x_2.\,(f_2\,((), \mathrm{inj}^2(), x_2, c))), x)$

Figure 13. Typing and translation of primitives

General

SNG-CREATE	:	$\tau \le \exists\sigma.[\sigma] * \{\sigma : \tau\}$	$\triangleright$	$x \rightsquigarrow ((), x)$
SNG-EXTRACT	:	$[\sigma] * \{\sigma : \tau\} \le \tau * \{\sigma : \tau\}$	$\triangleright$	$((), x) \rightsquigarrow (x, x)$
FREE	:	$C \le \emptyset$	$\triangleright$	$x \rightsquigarrow ()$

Focus-value

FOCUS-REF	:	$\{\sigma : \mathrm{ref}\,\theta_1\} \equiv \exists\sigma_1.\{\sigma : \mathrm{ref}\,[\sigma_1]\} * \{\sigma_1 : \theta_1\}$	$\triangleright$	$x \leftrightsquigarrow ((), x)$
FOCUS-PAIR[1]	:	$\{\sigma : \theta_1 \times \theta_2\} \equiv \exists\sigma_1.\{\sigma : [\sigma_1] \times \theta_2\} * \{\sigma_1 : \theta_1\}$	$\triangleright$	$(x_1, x_2) \leftrightsquigarrow (((), x_2), x_1)$
FOCUS-SUM[1]	:	$\{\sigma : \theta_1 + \bot\} \equiv \exists\sigma_1.\{\sigma : [\sigma_1] + \bot\} * \{\sigma_1 : \theta_1\}$	$\triangleright$	$(\mathrm{inj}^1\,x) \leftrightsquigarrow (\mathrm{inj}^1\,(), x)$

Regions

NEW-GRP	:	$\emptyset \le \exists\rho_1 \cdots \rho_n.\{\rho_1 : \theta_1\} * \ldots * \{\rho_n : \theta_n\}$	$\triangleright$	$() \rightsquigarrow (\mathrm{map\_empty}, \ldots, \mathrm{map\_empty})$
ADOPT-GRP	:	$[\sigma] * \{\sigma : \theta\} * \{\rho : \theta\} \le [\rho] * \{\rho : \theta\}$	$\triangleright$	$((), x, m) \rightsquigarrow \mathrm{let}\,k = \mathrm{map\_fresh}\,m\,\mathrm{in}$ $(k, \mathrm{map\_add}\,(m, k, x))$
FOCUS-GRP	:	$[\rho] * \{\rho : \theta\} \le \exists\sigma.[\sigma] * \{\sigma : \theta\} * \{\rho : \theta \setminus \sigma\}$	$\triangleright$	$(k, m) \rightsquigarrow ((), \mathrm{map\_get}\,(m, k), (m, k)))$
UNFOCUS-GRP	:	$\{\sigma : \theta\} * \{\rho : \theta \setminus \sigma\} \le \{\rho : \theta\}$	$\triangleright$	$(x, (m, k)) \rightsquigarrow \mathrm{map\_set}\,(m, k, x)$

Embedding

$\exists$.EMBED	:	$\{\sigma : (\exists\alpha.\theta)\} \equiv \exists\alpha.\{\sigma : \theta\}$	$\triangleright$	$x \leftrightsquigarrow x$
$*$.EMBED	:	$\{\sigma : (\theta * C)\} \equiv \{\sigma : \theta\} * C$	$\triangleright$	$x \leftrightsquigarrow x$

Administrative

$*$.COMM-CAP	:	$C_1 * C_2 \equiv C_2 * C_1$	$\triangleright$	$(x_1, x_2) \leftrightsquigarrow (x_2, x_1)$
$*$.ASSOC	:	$(o * C_1) * C_2 \equiv o * (C_1 * C_2)$	$\triangleright$	$((x_1, x_2), x_3) \leftrightsquigarrow (x_1, (x_2, x_3))$
$*$.NEUTRAL	:	$o * \emptyset \equiv o$	$\triangleright$	$(x, ()) \leftrightsquigarrow x$
$\exists$.COMM	:	$\exists\alpha_1.\exists\alpha_2.o \equiv \exists\alpha_2.\exists\alpha_1.o$	$\triangleright$	$x \leftrightsquigarrow x$
$\exists$.EXTRUDE-L	:	$o_1 * (\exists\alpha.o_2) \equiv \exists\alpha.(o_1 * o_2)$	$\triangleright$	$x \leftrightsquigarrow x$
$\exists$.EXTRUDE-R	:	$(\exists\alpha.o_1) * o_2 \equiv \exists\alpha.(o_1 * o_2)$	$\triangleright$	$x \leftrightsquigarrow x$

Figure 14. Subtyping rules and their translations

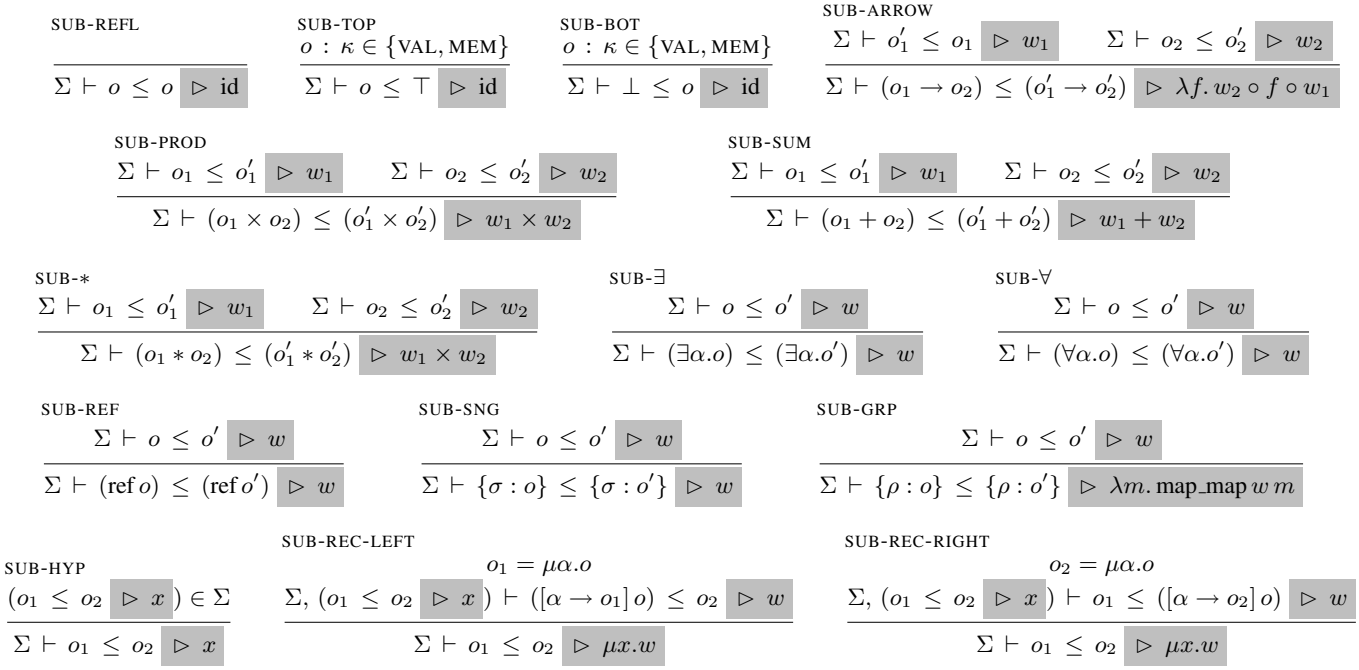

Figure 15. Subtyping under context

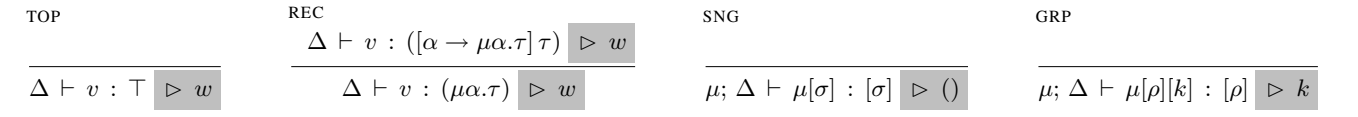

Figure 16. Additional rules for typing at runtime

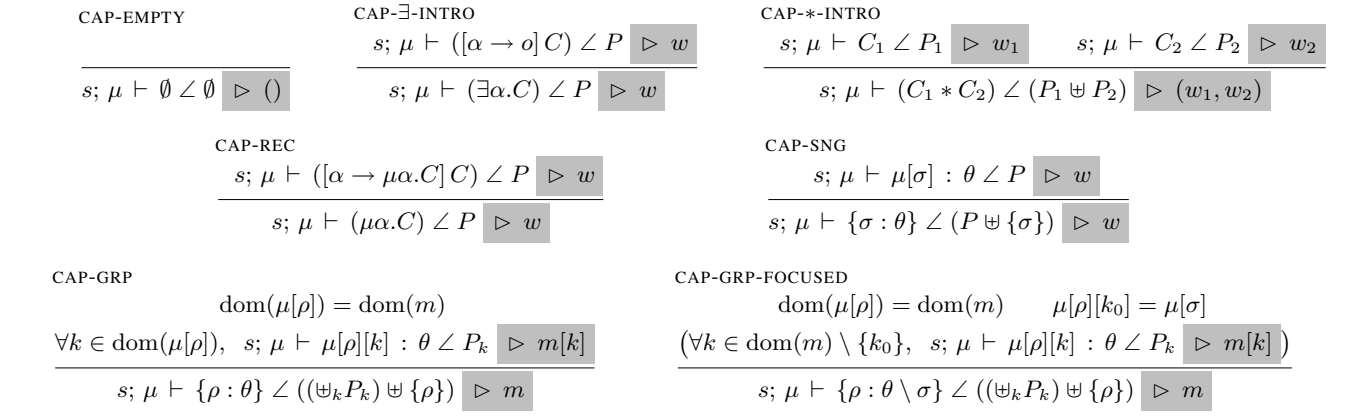

Figure 17. Typing, ownership and translation of capabilities

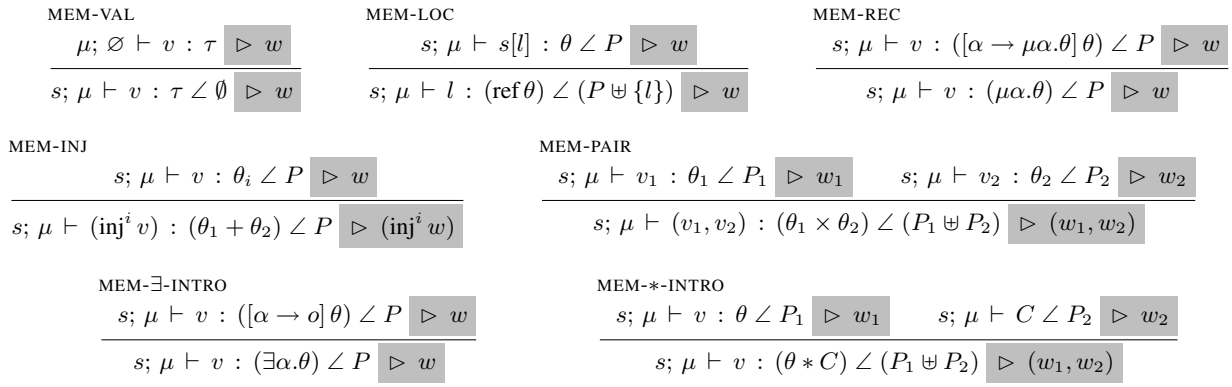

Figure 18. Typing, ownership and translation of memory values

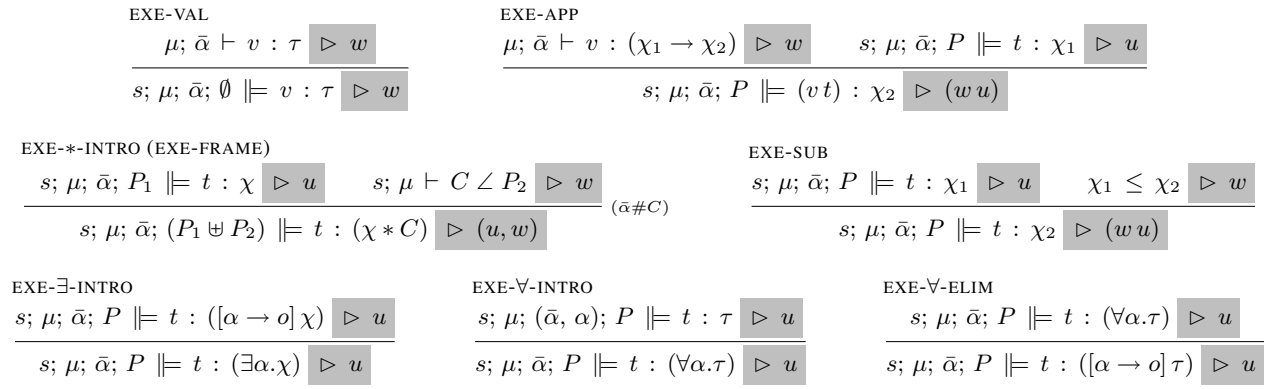

Figure 19. Typing executed terms

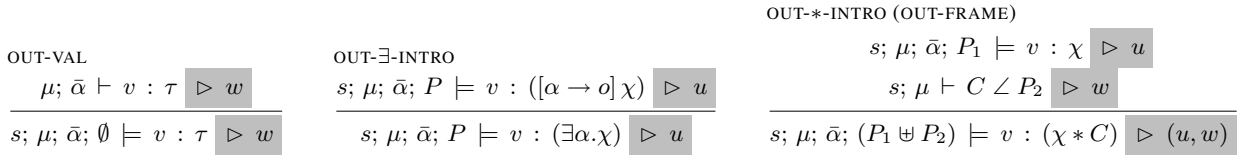

Figure 20. Typing output values

FUN

$$\frac{\Delta,\, x : \chi_1 \Vdash t : \chi_2 \;\triangleright\; u}{\Delta \vdash (\lambda x.\, t) : (\chi_1 \to \chi_2) \;\triangleright\; (\lambda x.\, u)}$$

LET

$$\frac{\Delta,\, \Gamma_1 \vdash t_1 : \chi_1 \;\triangleright\; u_1 \qquad \Delta,\, (x : \chi_1),\, \Gamma_2 \vdash t_2 : \chi_2 \;\triangleright\; u_2}{\Delta,\, \Gamma_1,\, \Gamma_2 \vdash (\mathsf{let}\, x = t_1\, \mathsf{in}\, t_2) : \chi_2 \;\triangleright\; (\mathsf{let}\, x = u_1\, \mathsf{in}\, u_2)}$$

SUB-CTX

$$\frac{\Gamma_1,\, (x : o_2),\, \Gamma_2 \Vdash t : \chi \;\triangleright\; u \qquad o_1 \le o_2 \;\triangleright\; w}{\Gamma,\, (x : o_1),\, \Gamma_2 \Vdash t : \chi \;\triangleright\; (\mathsf{let}\, x = w\, x\, \mathsf{in}\, u)}$$

SUB-REC (AMADIO AND CARDELLI)

$$\frac{\Sigma,\, (\alpha_1 \le \alpha_2) \vdash o_1 \le o_2}{\Sigma \vdash \mu\alpha_1.o_1 \le \mu\alpha_2.o_2}$$

FOLD-UNFOLD

$$\frac{}{\mu\alpha.o \equiv ([\alpha \to \mu\alpha.o]\, o) \;\triangleright\; \lambda x.\, x}$$

Quantifiers

∀.ELIM	:	$\forall\alpha.o_1 \le [\alpha \to o_2]\, o_1$		$\triangleright$	$x \rightsquigarrow x$
∃.INTRO	:	$[\alpha \to o_2]\, o_1 \le \exists\alpha.o_1$		$\triangleright$	$x \rightsquigarrow x$
∃.NO-OCCUR	:	$\exists\alpha.o \le o$ (when $\alpha \# o$)		$\triangleright$	$x \rightsquigarrow x$

Arrows

→.FRAME	:	$(\chi_1 \to \chi_2) \le (\chi_1 * C) \to (\chi_2 * C)$		$\triangleright$	$f \rightsquigarrow \lambda(x_1, x_2).\, (f\, x_1, x_2)$
→.DISTRIB-RIGHT	:	$\forall\alpha.(\chi_1 \to \chi_2) \equiv \chi_1 \to (\forall\alpha.\chi_2)$	(when $\alpha \# \chi_1$)	$\triangleright$	$x \leftrightsquigarrow x$
→.DISTRIB-LEFT	:	$\forall\alpha.(\chi_1 \to \chi_2) \equiv (\exists\alpha.\chi_1) \to \chi_2$	(when $\alpha \# \chi_2$)	$\triangleright$	$x \leftrightsquigarrow x$

Figure 21. Extra rules

8. Definitions for proofs

The following notation was introduced in the paper:

$$(s,\mu)_{\backslash P} \sqsubseteq (s',\mu')_{\backslash P'} \quad := \quad \begin{cases} \mu \sqsubseteq \mu' \\ s_{\backslash P} \subseteq s'_{\backslash P'} \\ \mu_{\backslash P} \subseteq \mu'_{\backslash P'} \end{cases}$$

Moreover, we define another notation to capture the idea that the store and structure of regions can only evolve on the portion P which is owned, as follows:

$$(s,\mu) \sqsubseteq_{|P} (s',\mu') \quad := \quad \begin{cases} \mu \sqsubseteq \mu' \\ \mu_{|P} \subseteq \mu'_{|P} \\ s_{|P} \subseteq s'_{|P} \end{cases}$$

where $s_{|P}$ describes the subset of bindings from s which bind a location contained in P, and similarly $\mu_{|P}$ describes the subset of bindings from μ which bind a region contained in P.

Properties Figure 26 gives the properties of the two above judgments that are used throughout the proof.

9. Main result of the proof

Each step of reduction in the source language is simulated by one or more steps of reduction in the target language. Moreover, if the source program converges to a value, then its translation also reaches a value after a finite number of steps. This is summarized in the following diagram.

$$\begin{array}{ccccccccc} t \,/\, s & \longrightarrow & t' \,/\, s' & \longrightarrow & \ldots & \longrightarrow & v'' \,/\, s'' & \not\longrightarrow \\ \vdots & & \vdots & & & & \vdots & \\ u & \longrightarrow^+ & u' & \longrightarrow^+ & \ldots & \longrightarrow^+ & u'' & \longrightarrow^* w \not\longrightarrow \end{array}$$

The corresponding formal statement is given below.

Theorem (EQUIVALENCE) *If the following property holds*

$$\bar{\alpha} \Vdash t : \chi \rhd u$$

then $\begin{cases} (1) & (t \,/\, \emptyset) \longrightarrow^\infty & \iff & u \longrightarrow^\infty \\ (2) & (t \,/\, \emptyset) \longrightarrow^* (v \,/\, s) & \iff & u \longrightarrow^* w \end{cases}$

Moreover, if the terms converge, there exists μ and P such that

$$s;\, \mu;\, \bar{\alpha};\, P \models v : \chi \rhd w$$

Proof. Lemma SIMULATION proves that a step of reduction in the source language is simulated by one or more steps of reduction in the target language. Lemma TERMINATION proves that the translated program converges when its source has converged. □

10. Proofs

Proofs follows, on single-columned pages.

$$\begin{array}{lcl}
\text{node}\,\rho & := & \text{ref}\,(\text{unit} + [\rho]) \\
& \rhd & \text{unit} + \text{key} \\
& & \{\rho : \text{node}\,\rho\} \\
\text{forest} & := & \text{map}\,(\text{unit} + \text{key}) \\
\\
\text{new} & : & \forall\rho.\text{unit} \to_{\{\rho:\text{node}\,\rho\}} [\rho] \\
& \rhd & \forall\alpha.\text{unit} \times \text{forest} \to \text{key} \times \text{forest} \\
& := & \lambda().\text{ref}\,(\text{inj}^1\,()) \\
& \rhd & \lambda((),r). \\
& & \quad \text{let}\,k = \text{map_fresh}\,r\,\text{in} \\
& & \quad \text{let}\,r' = \text{map_add}\,(r,k,(\text{inj}^1\,())\,)\,\text{in} \\
& & \quad (k,r') \\
\\
\text{find} & : & \forall\rho.[\rho] \to_{\{\rho:\text{node}\,\rho\}} [\rho] \\
& \rhd & \forall\alpha.\text{key} \times \text{forest} \to \text{key} \times \text{forest} \\
& := & \mu f.\lambda n.\text{match}\,(\text{get}\,n)\,\text{with} \\
& & \quad |\,\text{inj}^1\,() \;\Rightarrow\; n \\
& & \quad |\,\text{inj}^2\,m \;\Rightarrow \\
& & \quad\quad \text{let}\,p = f\,m\,\text{in} \\
& & \quad\quad \text{set}\,(n,\text{inj}^2\,p)\,; \\
& & \quad\quad p \\
& \rhd & \mu f.\lambda(n,r).\text{match}\,(\text{map_get}\,(r,n))\,\text{with} \\
& & \quad |\,\text{inj}^1\,() \;\Rightarrow\; (n,r) \\
& & \quad |\,\text{inj}^2\,m \;\Rightarrow \\
& & \quad\quad \text{let}\,(p,r') = f\,(m,r)\,\text{in} \\
& & \quad\quad \text{let}\,r'' = \text{map_set}\,(r',n,(\text{inj}^2\,p))\,\text{in} \\
& & \quad\quad (p,r'') \\
\\
\text{union} & : & \forall\rho.[\rho] \times [\rho] \to_{\{\rho:\text{node}\,\rho\}} \text{unit} \\
& \rhd & \forall\alpha.\text{key} \times \text{key} \times \text{forest} \to \text{unit} \times \text{forest} \\
& := & \lambda(n_1,n_2). \\
& & \quad \text{let}\,p_1 = \text{find}\,n_1\,\text{in} \\
& & \quad \text{let}\,p_2 = \text{find}\,n_2\,\text{in} \\
& & \quad \text{set}\,(p_1,\text{inj}^2\,p_2) \\
& \rhd & \lambda(n_1,n_2,r). \\
& & \quad \text{let}\,(p_1,r') = \text{find}\,(n_1,r)\,\text{in} \\
& & \quad \text{let}\,(p_2,r'') = \text{find}\,(n_2,r')\,\text{in} \\
& & \quad \text{let}\,r''' = \text{map_set}\,(r'',p_1,(\text{inj}^2\,p_2))\,\text{in} \\
& & \quad ((),r''')
\end{array}$$

Figure 23. Example: union-find

$$
\begin{array}{lll}
\text{reflexivity:} & & (s,\mu)_{\backslash P} \sqsubseteq (s,\mu)_{\backslash P} \\
\text{transitivity:} & \left\{ \begin{array}{l} (s,\mu)_{\backslash P} \sqsubseteq (s',\mu')_{\backslash P'} \\ (s',\mu')_{\backslash P'} \sqsubseteq (s'',\mu'')_{\backslash P''} \end{array} \right. & \Rightarrow \quad (s,\mu)_{\backslash P} \sqsubseteq (s'',\mu'')_{\backslash P''} \\
\text{composition:} & \left\{ (s,\mu)_{\backslash P} \sqsubseteq (s',\mu')_{\backslash P'} \right. & \Rightarrow \quad (s,\mu)_{\backslash (P \uplus P_1)} \sqsubseteq (s',\mu')_{\backslash (P' \uplus P_1)} \\
\text{complementary:} & \left\{ \begin{array}{l} (s,\mu)_{\backslash P} \sqsubseteq (s',\mu')_{\backslash P'} \\ P_1 \uplus P \end{array} \right. & \Rightarrow \quad (s,\mu) \sqsubseteq_{|P_1} (s',\mu') \\
\text{restriction:} & \left\{ \begin{array}{l} (s,\mu) \sqsubseteq_{|P} (s',\mu') \\ P_1 \subseteq P \end{array} \right. & \Rightarrow \quad (s,\mu) \sqsubseteq_{|P_1} (s',\mu')
\end{array}
$$

Figure 26. Properties about invariants

$$
\begin{array}{lcl}
\text{type of } f & := & \mu\alpha.(\text{unit} \rightarrow_{\{\sigma:\text{ref}\,\alpha\}} \text{unit}) \\
& \rhd & \mu\alpha.(\text{unit} \times \alpha \rightarrow \text{unit} \times \alpha) \\
\\
\text{loop} & := & \text{let}\, r = \text{ref}() \,\text{in} \\
& & \text{let}\, f = \lambda().\ \\
& & \quad \text{let}\, g = \text{get}\, r \,\text{in} \\
& & \quad g\,() \,\text{in} \\
& & \text{set}\,(r, f)\,; \\
& & f\,() \\
& \rhd & \text{let}\, r_1 = () \,\text{in} \\
& & \text{let}\, f = \lambda((), r).\ \\
& & \quad \text{let}\,(g, r') = (r, r) \,\text{in} \\
& & \quad g\,((), r') \,\text{in} \\
& & \text{let}\, r_2 = f \,\text{in} \\
& & f\,((), r_2) \\
& \rhd & \text{let}\, f = \lambda((), r).r\,((), r) \,\text{in} \\
& & f\,((), f) \\
& \rhd & \text{let}\, f = \lambda r.r\, r \,\text{in} \\
& & f\, f
\end{array}
$$

Figure 24. Example: Infinite loop without recursion

$$
\begin{array}{lcl}
\text{type of } F & := & \forall\epsilon.(\alpha \rightarrow_\epsilon \beta) \rightarrow (\alpha \rightarrow_\epsilon \beta) \\
& \rhd & \forall\gamma.([\![\alpha]\!] \times \gamma \rightarrow [\![\beta]\!] \times \gamma) \rightarrow ([\![\alpha]\!] \times \gamma \rightarrow [\![\beta]\!] \times \gamma) \\
\\
\text{fixpoint} & := & \lambda F.\text{let}\, r = \text{ref}() \,\text{in} \\
& & \quad \text{let}\, g = \lambda x.\ \\
& & \quad\quad \text{let}\, f = \text{get}\, r \,\text{in} \\
& & \quad\quad f\, x \,\text{in} \\
& & \quad \text{let}\, f = \lambda x.F\, g\, x \,\text{in} \\
& & \quad \text{set}\,(r, f)\,; \\
& & \quad f \\
& \rhd & \lambda F.\text{let}\, r_1 = () \,\text{in} \\
& & \quad \text{let}\, g = \lambda(x, r).\ \\
& & \quad\quad \text{let}\,(f, r') = (r, r) \,\text{in} \\
& & \quad\quad f\,(x, r') \,\text{in} \\
& & \quad \text{let}\, f = \lambda(x, r).F\, g\,(x, r) \,\text{in} \\
& & \quad \text{let}\, r_2 = f \,\text{in} \\
& & \quad (f, r_2) \\
& \rhd & \lambda F.\text{let}\, g = \lambda(x, r).r\,(x, r) \,\text{in} \\
& & \quad \text{let}\, f = F\, g \,\text{in} \\
& & \quad (f, f) \\
\\
\text{apply\_rec} & := & \lambda F.\lambda x.\ \\
& & \text{let}\, f = \text{fixpoint}F \ \text{in} \\
& & f\, x \\
& \rhd & \lambda F.\lambda x.\ \\
& & \text{let}\,(f, e) = \text{fixpoint}F \ \text{in} \\
& & f\,(x, e) \\
& \rhd & \lambda F.\lambda x.\ \\
& & \text{let}\, g = \lambda(y, f).f\,(y, f) \,\text{in} \\
& & F\, g\,(x, (F\, g))
\end{array}
$$

Figure 25. Example: Fixpoint without recursion (Landin's knot)

1 Substitution

Typing and translation are preserved when extending the context with a set of non-linear bindings Δ_2.

Lemma WEAKEN-VAL

$$\mu;\, (\Delta_1,\, \Delta_3) \vdash v : \tau \rhd w \quad \Rightarrow \quad \mu;\, (\Delta_1,\, \Delta_2,\, \Delta_3) \vdash v : \tau \rhd w$$

Lemma WEAKEN-TRM

$$\mu;\, (\Delta_1,\, \Gamma_3) \Vdash t : \chi \rhd u \quad \Rightarrow \quad \mu;\, (\Delta_1,\, \Delta_2,\, \Gamma_3) \Vdash t : \chi \rhd u$$

Proof. Trivial by mutual induction. □

Typing is preserved through substitution of a value x of non-linear value type τ_1 by a value v_1 of corresponding type. Futhermore, substitution in the source language is reflected by a substitution in the translated program.

Lemma SUBST-VAL

$$\begin{array}{l} \left\{ \begin{array}{l} \mu;\, \Delta_1 \vdash v_1 : \tau_1 \rhd w_1 \\ \mu;\, (\Delta_1,\, x : \tau_1,\, \Delta_2) \vdash v_2 : \tau_2 \rhd w_2 \end{array} \right. \\ \Rightarrow \\ \left\{ \mu;\, (\Delta_1,\, \Delta_2) \vdash ([x \to v_1]\, v_2) : \tau_2 \rhd ([x \to w_1]\, w_2) \right. \end{array}$$

Lemma SUBST-TRM

$$\begin{array}{l} \left\{ \begin{array}{l} \mu;\, \Delta_1 \vdash v_1 : \tau_1 \rhd w_1 \\ \mu;\, (\Delta_1,\, x : \tau_1,\, \Gamma_2) \Vdash t : \chi \rhd u \end{array} \right. \\ \Rightarrow \\ \left\{ \mu;\, (\Delta_1,\, \Gamma_2) \Vdash ([x \to v_1]\, t) : \chi \rhd ([x \to w_1]\, u) \right. \end{array}$$

Proof. By mutual induction. Non-trivial cases:

– Case VAR: use WEAKEN-VAL if the variable involved happens to be the variable substitued.

– Case SUB: the coercion is a closed term and thus is not affected by substitution.

– Case $*$-INTRO: the variable which is being substituted must be distinct from which is framed, since they are bound to objects of different kinds. Thus, it suffices to apply the IH.

– Case $*$-ELIM: the binding deconstructed is a conjunction, thereafter it is not a value type and cannot be the binding being substituted. Thus, it suffices to apply the IH. □

Beta-reduction of an abstraction $(\lambda x.\, t)$ of type $(\chi \to \chi')$ onto a value v of type χ is sound.

Lemma SUBST-OUT

$$\begin{array}{l} \left\{ \begin{array}{l} \mu;\, \bar{\alpha},\, (x : \chi) \Vdash t : \chi' \rhd u \\ s;\, \mu;\, \bar{\alpha};\, P \models v : \chi \rhd w \end{array} \right. \\ \Rightarrow \\ \quad \exists\, \mu'\, P'\, u', \\ \quad \left\{ \begin{array}{l} s;\, \mu';\, \bar{\alpha};\, P' \Vdash ([x \to v]\, t) : \chi' \rhd u' \\ ([x \to w]\, u) \longrightarrow^* u' \\ (s, \mu)_{\setminus P} \sqsubseteq (s, \mu')_{\setminus P'} \end{array} \right. \end{array}$$

Proof. The statement is strengthened for the induction in the next lemma.□

Lemma SUBST-OUT-INDUCTION

$$\begin{array}{l} \left\{ \begin{array}{l} \mu;\, \bar{\alpha},\, (x : \chi),\, (x_n : C_n),\, \ldots,\, (x_1 : C_1) \Vdash t : \chi' \rhd u \\ s;\, \mu;\, \bar{\alpha};\, P \models v : \chi \rhd w \\ s;\, \mu \vdash C_i \angle P_i \rhd w_i \quad (i \in 1..n) \\ \uplus_i P_i \end{array} \right. \\ \Rightarrow \\ \quad \exists\, \mu'\, P'\, u', \\ \quad \left\{ \begin{array}{l} s;\, \mu';\, \bar{\alpha};\, P' \Vdash ([x \to v]\, t) : \chi' \rhd u' \\ [x_1 \to w_1] \ldots [x_n \to w_n]\, [x \to w]\, u \longrightarrow^* u' \\ (s, \mu)_{\setminus (\uplus_{i \in 1..n} P_i \uplus P)} \sqsubseteq (s, \mu')_{\setminus P'} \end{array} \right. \end{array}$$

Proof. By induction on the typing derivation of t.

▶ *Case:* VAL

0) For this rule to be applicable, n must be 0, χ must be a value type (call it τ_χ), χ' must also be a value type (call it $\tau_{\chi'}$), t must be a value (call it v_t) and its translation u as well (call it w_u). The typing derivation is

$$\mu;\, \bar{\alpha},\, (x : \tau_\chi) \vdash v_t \,:\, \tau_{\chi'} \rhd w_u$$

1) By inversion on the typing of v (rule OUT-VAL), $P = \emptyset$ and

$$\mu;\, \bar{\alpha} \vdash v \,:\, \tau_\chi \rhd w$$

2) By lemma SUBST-VAL applied to the result of (0) and (1),

$$\mu;\, \bar{\alpha} \vdash ([x \to v]\, v_t) \,:\, \tau_{\chi'} \rhd ([x \to w]\, w_u)$$

3) By rule EXE-VAL, it follows:

$$s;\, \mu;\, \bar{\alpha};\, \emptyset \Vdash ([x \to v]\, v_t) \,:\, \tau_{\chi'} \rhd ([x \to w]\, w_u)$$

4) Conclude with $\mu' = \mu$ and $P' = \emptyset$ and $u' = ([x \to w]\, w_u)$.

▶ *Case:* APP

0) The typing derivation ends with:

$$\frac{\mu;\, \bar{\alpha},\, \Delta \Vdash v_1 \,:\, (\chi_1' \to \chi') \rhd u_1 \qquad \mu;\, \bar{\alpha},\, (x : \chi) \Vdash t_2 \,:\, \chi_1' \rhd u_2}{\mu;\, \bar{\alpha},\, (x : \chi) \Vdash (v_1\, t_2) \,:\, \chi' \rhd (u_1\, u_2)}\ \text{APP}$$

where Δ stands for the binding $(x : \tau)$ if χ is of the form τ, or is the empty environment binding otherwise.

1) If Δ is not empty, by lemma SUBST-VAL it follows:

$$\mu;\, \bar{\alpha} \Vdash ([x \to v]\, v_1) \,:\, (\chi_1' \to \chi') \rhd ([x \to w]\, u_1)$$

Otherwise Δ is empty and this same result is immediate because x is fresh from v_1 and u_1.

By TERMINATION applied to the first premise,

2) By induction on the above, there exists μ'' and w_1' such that

$$\begin{cases} ([x \to w]\, u_1) \longrightarrow^* w_1' \\ \mu'';\, \bar{\alpha} \vdash ([x \to v]\, v_1) \,:\, (\chi_1' \to \chi') \rhd w_1' \\ \mu \sqsubseteq \mu'' \end{cases}$$

3) By induction hypothesis applied to the typing of t_2, there exists μ', P' and u' such that

$$\begin{cases} s;\, \mu';\, \bar{\alpha};\, P' \;\Vdash\; ([x \to v]\, t_2) \;:\; \chi' \;\rhd\; u'_2 \\ [x_1 \to w_1] \ldots [x_n \to w_n]\, [x \to w]\, u_2 \longrightarrow^* u'_2 \\ (s, \mu'')_{\setminus (\uplus_{i \in 1..n} P_i \uplus P'')} \;\sqsubseteq\; (s, \mu')_{\setminus P'} \end{cases}$$

4) Let $u' = (w_1\, u'_2)$. Rule EXE-APP is used to build the following conclusion:

$$s;\, \mu';\, \bar{\alpha};\, P' \;\Vdash\; ([x \to v]\, (v_1\, t_2)) \;:\; \chi' \;\rhd\; u'$$

Its first premise is the conclusion of:

$$\frac{\mu'' \sqsubseteq \mu' \qquad \mu'';\, \bar{\alpha} \;\vdash\; ([x \to v]\, v_1) \;:\; (\chi'_1 \to \chi') \;\rhd\; w'_1}{\mu';\, \bar{\alpha} \;\vdash\; ([x \to v]\, v_1) \;:\; (\chi'_1 \to \chi') \;\rhd\; w'_1}\ \text{STABLE-VAL}$$

and its second premise is the derivation for $([x \to v]\, t_2)$ obtained in (3).

5) The evolution of state is obtained by transitivity.

6) The reduction sequence in the target language is:

$$([x \to w]\, (u_1\, u_2)) = (([x \to w]\, u_1)\, ([x \to w]\, u_2)) \longrightarrow^* (w'_1\, u'_2) = u'$$

▶ *Case:* SUB

Apply induction hypothesis, use the fact that a coercion is a closed value thus is not affected by substitution, and conclude using EXE-SUB.

▶ *Case:* $*$-INTRO

0) Without loss of generality, assume C_1 is being framed out. Then u must be of the form (u_1, x_1) and χ' must be of the form $(\chi'_1 * C'_1)$. The typing derivation ends with

$$\frac{\mu;\, \bar{\alpha},\, (x : \chi),\, (x_n : C_n),\, \ldots,\, (x_2 : C_2) \;\Vdash\; t \;:\; \chi'_1 \;\rhd\; u_1}{\mu;\, \bar{\alpha},\, (x : \chi),\, (x_n : C_n),\, \ldots,\, (x_1 : C_1) \;\Vdash\; t \;:\; (\chi'_1 * C'_1) \;\rhd\; (u_1, x_1)}\ \text{FRAME}$$

1) By induction hypothesis, there exists μ', P'' and u'_1 such that:

$$\begin{cases} s;\, \mu';\, \bar{\alpha};\, P'' \;\Vdash\; ([x \to v]\, t) \;:\; \chi' \;\rhd\; u'' \\ [x_2 \to w_2] \ldots [x_n \to w_n]\, [x \to w]\, u_1 \longrightarrow^* u'_1 \\ (s, \mu)_{\setminus (\uplus_{i \in 2..n} P_i \uplus P)} \;\sqsubseteq\; (s, \mu')_{\setminus P''} \end{cases}$$

2) Now let $P' = (P'' \uplus P_1)$ and $u' = (u'_1, w_1)$.

3) The typing derivation of the conclusion is built using EXE-FRAME:

$$\dfrac{s;\, \mu';\, \bar{\alpha};\, P'' \;\|\!\models\; t \;:\; \chi'_1 \rhd u'_1 \qquad \dfrac{\mu \sqsubseteq \mu' \qquad s;\, \mu \vdash C_1 \angle P_1 \rhd w_1}{s;\, \mu' \vdash C_1 \angle P_1 \rhd w_1}\ \textsc{stable-cap}}{s;\, \mu';\, \bar{\alpha};\, (P'' \uplus P_1) \;\|\!\models\; t \;:\; (\chi'_1 * C'_1) \rhd (u'_1, w_1)}\ \textsc{exe-frame}$$

4) The reduction sequence of the conclusion is obtained as follows:

$$\begin{array}{lll} & [x_1 \to w_1] \ldots [x_n \to w_n]\, [x \to w]\, u & \\ = & [x_1 \to w_1] \ldots [x_n \to w_n]\, [x \to w]\, (u_1, x_1) & \\ \longrightarrow & ([x_2 \to w_2] \ldots [x_n \to w_n]\, [x \to w]\, u_1), w_1) & \text{since } x_1 \# u_1 \\ \longrightarrow^* & (u'_1, w_1) & \text{by IH} \\ = & u' & \end{array}$$

5) For evolution of the state, it suffices to verify that:

$$\begin{array}{ll} & (s, \mu)_{\setminus (\uplus_{i \in 2..n} P_i \uplus P)} \;\sqsubseteq\; (s, \mu')_{\setminus P''} \\ \Rightarrow & (s, \mu)_{\setminus (\uplus_{i \in 2..n} P_i \uplus P \uplus P_1)} \;\sqsubseteq\; (s, \mu')_{\setminus (P'' \uplus P_1)} \end{array}$$

The conclusion then follows from the equalities:

$$(\uplus_{i \in 1..n} P_i \uplus P) = (\uplus_{i \in 2..n} P_i \uplus P \uplus P_1) \quad \text{and} \quad P' = (P'' \uplus P_1)$$

▶ *Case:* $*$-ELIM

⋆ *Sub-case: Elimination of a conjunction of a computation type and a capability*

0) In this case, χ must be of the form $\chi_1 * C_{n+1}$ and u must be of the form $(\mathrm{let}\,(x, x_{n+1}) = x\,\mathrm{in}\,u_1)$. The typing derivation ends with:

$$\mu;\, \bar{\alpha},\, (x : \chi_1),\, (x_{n+1} : C_{n+1}),\, (x_n : C_n),\, \ldots,\, (x_1 : C_1) \Vdash\; t \;:\; \chi' \rhd u_1$$

1) By inversion on the typing of v, since only rule OUT-FRAME applies, it follows the decompositions of P as $P_1 \uplus P_{n+1}$ and of w as (w_1, w_{n+1}) and

$$\begin{cases} s;\, \mu;\, \bar{\alpha};\, P_1 \;\models\; v \;:\; \chi_1 \rhd w_1 \\ s;\, \mu \vdash C_{n+1} \angle P_{n+1} \rhd w_{n+1} \end{cases}$$

2) By induction hypothesis, there exists μ', P' and u' such that

$$\begin{cases} s;\, \mu';\, \bar{\alpha};\, P' \Vdash ([x \to v]\, t) \,:\, \chi' \rhd u' \\ [x_1 \to w_1] \ldots [x_{n+1} \to w_{n+1}]\, [x \to w]\, u_1 \longrightarrow^* u' \\ (s, \mu)_{\setminus (\uplus_{i \in 1..n+1} P_i \uplus P_1)} \sqsubseteq (s, \mu')_{\setminus P'} \end{cases}$$

3) The derivation for u' is:

$$\begin{array}{lll} & [x_1 \to w_1] \ldots [x_n \to w_n]\, [x \to w]\, u & \\ = & [x_1 \to w_1] \ldots [x_n \to w_n]\, [x \to (w_1, w_{n+1})]\, (\mathrm{let}\,(x, x_{n+1}) = x \,\mathrm{in}\, u_1) & \\ \longrightarrow & [x_1 \to w_1] \ldots [x_n \to w_n]\, [x_{n+1} \to w_{n+1}]\, [x \to w_1]\, u_1 & \\ \longrightarrow^* & u' & \text{by IH} \end{array}$$

4) And the evolution of the state is already proved, since:

$$(\uplus_{i \in 1..n+1} P_i) \uplus P_1 \;=\; (\uplus_{i \in 1..n} P_i) \uplus (P_{n+1} \uplus P_1) \;=\; (\uplus_{i \in 1..n} P_i) \uplus P$$

⋆ *Sub-case: Elimination of a conjunction of a two capabilities*

0) Without loss of generality, we can assume C_n is the conjunction eliminated. Thus C_n must be of the form $C_{n1} * C_{n2}$ and u must be of the form $(\mathrm{let}\,(x_{n1}, x_{n2}) = x_n \,\mathrm{in}\, u_1)$ and we have

$$\mu;\, \bar{\alpha},\, (x : \chi_1),\, (x_{n2} : C_{n2})(x_{n1} : C_{n1}),\, (x_{n-1} : C_{n-1}),\, \ldots,\, (x_1 : C_1) \Vdash t \,:\, \chi' \rhd u_1$$

1) By inversion on the typing of C_n (rule CAP-∗-INTRO), P_n decomposes as $(P_{n1} \uplus P_{n2})$ and w_n decomposes as (w_{n1}, w_{n2}) and

$$\begin{cases} s;\, \mu \vdash C_{n1} \angle P_{n1} \rhd w_{n1} \\ s;\, \mu \vdash C_{n2} \angle P_{n2} \rhd w_{n2} \end{cases}$$

2) The conclusion follow from the induction hypothesis with similar arguments as in the previous subcase.

▶ *Case:* ∀-INTRO-TRM

0) In this case, χ' is of the form $\forall \alpha.\tau'$ and the following proposition holds:

$$\mu;\, \bar{\alpha},\, (x : \chi),\, (x_n : C_n),\, \ldots,\, (x_1 : C_1),\, \alpha \Vdash t \,:\, \tau' \rhd u$$

1) By commutativity of bindings, the judgment can be rewritten into:

$$\mu;\, (\bar{\alpha},\, \alpha),\, (x : \chi),\, (x'_n : C'_n),\, \ldots,\, (x'_1 : C'_1) \Vdash t \,:\, \tau' \rhd u$$

2) By the induction hypothesis, we obtain the reduction sequence on u' and the assumption on the evolution of the state, as well as a typing derivation for $([x \to v]\, t)$ which we can use to build the typing conclusion:

$$\frac{s;\, \mu';\, (\bar{\alpha},\, \alpha);\, P' \;\|\models\; ([x \to v]\, t) \;:\; \tau' \;\rhd\; u}{s;\, \mu';\, \bar{\alpha};\, P' \;\|\models\; ([x \to v]\, t) \;:\; (\forall \alpha.\tau') \;\rhd\; u}\ \textsc{exe-}\forall\textsc{-intro}$$

▶ *Case:* ∀-ELIM-TRM

Apply the induction hypothesis, conclude using EXE-∀-ELIM.

▶ *Case:* ∃-INTRO-TRM

Apply the induction hypothesis, conclude using EXE-∃-INTRO.

▶ *Case:* ∃-ELIM-TRM

0) In this case χ must be of the form $\exists \alpha.\chi_1$ and the following holds:

$$\mu;\, (\bar{\alpha},\, \alpha),\, (x : \chi_1),\, (x_n : C_n),\, \ldots,\, (x_1 : C_1) \;\Vdash\; t \;:\; \chi' \;\rhd\; u$$

1) By inversion on the typing of v, since only rule OUT-∃–INTRO applies, there exists o such that:

$$s;\, \mu;\, \bar{\alpha};\, P \;\models\; v \;:\; ([\alpha \to o]\, \chi_1) \;\rhd\; w$$

2) By type substitution from α to o in judgments (TYP-SUBST), if follows:

$$\mu;\, \bar{\alpha},\, (x : [\alpha \to o]\, \chi_1),\, (x_n : C_n),\, \ldots,\, (x_1 : C_1) \;\Vdash\; t \;:\; \chi' \;\rhd\; u$$

3) The rest follows from the induction hypothesis.

Substitution of a type variable by a type entity of the corresponding kind preserves all judgments.

Lemma TYP-SUBST *For any judgment J of n entities $e_1 \ldots e_n$,*

$$\begin{cases} \alpha \;:\; \kappa \\ e \;:\; \kappa \\ J(e_1, \ldots, e_n) \end{cases} \quad \Rightarrow \quad J([\alpha \to o]\, e_1, \ldots, [\alpha \to o]\, e_n)$$

Proof. By mutual induction on all judgments. □

2 Stability

Typing and translation of programs are preserved when regions grow from μ to μ' (written $\mu \sqsubseteq \mu'$).

Lemma STABLE-VAL

$$\left\{ \begin{array}{l} \mu;\, \Delta \vdash v : \tau \rhd w \\ \mu \sqsubseteq \mu' \end{array} \right. \quad \Rightarrow \quad \mu';\, \Delta \vdash v : \tau \rhd w$$

Lemma STABLE-TRM

$$\left\{ \begin{array}{l} \mu;\, \Gamma \Vdash t : \chi \rhd u \\ \mu \sqsubseteq \mu' \end{array} \right. \quad \Rightarrow \quad \mu';\, \Gamma \Vdash t : \chi \rhd u$$

Proof. By mutual induction. Non-trivial cases:

- Case SNG: $\mu'[\sigma] = \mu[\sigma]$ follows from $\mu \sqsubseteq \mu'$.
- Case RGN: $\mu'[\rho][k] = \mu[\rho][k]$ follows from $\mu \sqsubseteq \mu'$. □

Typing and translation of a capability are preserved when the piece of state controlled by this capability is not modified, and under the condition that all regions can only grow.

Lemma STABLE-CAP

$$\left\{ \begin{array}{l} s;\, \mu \vdash C \angle P \rhd w \\ (s, \mu) \sqsubseteq_{|P} (s', \mu') \end{array} \right. \quad \Rightarrow \quad s';\, \mu' \vdash C \angle P \rhd w$$

Lemma STABLE-MEM

$$\left\{ \begin{array}{l} s;\, \mu \vdash v : \theta \angle P \rhd w \\ (s, \mu) \sqsubseteq_{|P} (s', \mu') \end{array} \right. \quad \Rightarrow \quad s';\, \mu' \vdash v : \theta \angle P \rhd w$$

Proof. By mutual induction. Non-trivial cases:

- Case MEM-LOC: $s'[l] = s[l]$ follows from $s_{|P} \subseteq s'_{|P}$ and $l \in P$.
- Case CAP-SNG: $\mu'[\sigma] = \mu[\sigma]$ follows from $\mu \sqsubseteq \mu'$ and $\sigma \in P$.
- Case CAP-GRP: $\mu'[\rho] = \mu[\rho]$ follows from $\mu_{|P} \subseteq \mu'_{|P}$ and $\rho \in P$.
- Case CAP-GRP-FOCUSED: use $\mu'[\sigma] = \mu[\sigma]$ and $\mu'[\rho] = \mu[\rho]$, as above.

□

3 Subtyping

There is a correspondance between the typing judgment for values and the typing judgment for values covered by capabilities when restricted to value types.

Lemma MEM-TO-VAL

$$s;\, \mu \vdash v : \tau \angle \emptyset \rhd w \quad \Rightarrow \quad \mu;\, \varnothing \vdash v : \tau \rhd w$$

Proof. By induction.

- Case MEM-VAL: the premise of the rule gives the conclusion.
- Case MEM-LOC and MEM-$*$-INTRO: impossible (τ is non-linear).
- Other cases: apply the induction hypotheses and conclude using the corresponding typing rule for values (INJ, PAIR, REC or $\exists$-INTRO-VAL). □

Subtyping is admissible on closed values. Thus, subtyping operations can be eliminated from derivations. Moreover, coercions that translate subtyping operations transform target-language values accordingly.

Lemma REDUCE-SUB-VAL

$$\left\{ \begin{array}{l} \tau_1 \leq \tau_2 \rhd w \\ \mu;\, \bar{\alpha} \vdash v : \tau_1 \rhd w_1 \end{array} \right. \quad \Rightarrow \quad \exists\, w_2',\, \left\{ \begin{array}{l} (w\, w_1) \longrightarrow^+ w_2' \\ \mu;\, \bar{\alpha} \vdash v : \tau_2 \rhd w_2' \end{array} \right.$$

Proof. This statement is strengthened for the induction, as stated in the next lemma. □

Definition *We write* REDUCE-SUB-VAL(τ_1, τ_2, w, v) *if the previous lemma holds for these parameters.* ◇

Definition *The relation $v' \preceq v$ holds whenever v' is a value smaller from a structural point of view than v.* ◇

Lemma REDUCE-SUB-VAL-INDUCTION

$$\left\{\begin{array}{l}\Sigma \vdash \tau_1 \leq \tau_2 \rhd w \\ \forall\, v' \preceq v,\ \forall\, (\tau_1' \leq \tau_2' \rhd w') \in \Sigma,\ \textsc{reduce-sub-val}(\tau_1', \tau_2', w', v') \\ \mu;\, \bar{\alpha} \vdash v : \tau_1 \rhd w_1\end{array}\right.$$

$$\Rightarrow \quad \exists\, w_2',$$

$$\left\{\begin{array}{l}(w\, w_1) \longrightarrow^+ w_2' \\ \mu;\, \bar{\alpha} \vdash v : \tau_2 \rhd w_2'\end{array}\right.$$

Proof. By induction on the typing derivation of v. We consider the possible combination of (1) a typing rule that assigns type τ_1 to value v and (2) a subtyping rule of the form $\tau_1 \leq \tau_2$.

- Case Any rule, SUB-REFL: by assumption.
- Case Any rule, SUB-TOP: apply rule TOP.
- Case Any rule, SUB-BOT: impossible, because there is no rule that can be used to assign type $\bot$ to a value.
- Case FIX, SUB-ARROW: v is of the form $(\mu f.\lambda x.t)$, and χ is an arrow of the form $(\chi_1 \to \chi_2)$. The typing derivation is:

$$\frac{\Delta,\, f : (\chi_1 \to \chi_2),\, x : \chi_1 \Vdash t : \chi_2 \rhd u}{\Delta \vdash (\mu f.\lambda x.t) : (\chi_1 \to \chi_2) \rhd (\mu f.\lambda x.u)}\ \textsc{fix}$$

  We construct a typing derivation to give value v the type $(\chi_1' \to \chi_2')$ as follows (the translation component is left implicit):

$$\dfrac{\dfrac{\dfrac{\dfrac{\bar{\alpha},\, f : (\chi_1 \to \chi_2),\, x : \chi_1 \Vdash t : \chi_2}{\bar{\alpha},\, f : (\chi_1 \to \chi_2),\, x : \chi_1' \Vdash t : \chi_2}\ \textsc{sub-left}}{\bar{\alpha},\, f : (\chi_1 \to \chi_2),\, x : \chi_1' \Vdash t : \chi_2}\ \textsc{sub}}{\bar{\alpha},\, f : (\chi_1' \to \chi_2'),\, x : \chi_1' \Vdash t : \chi_2'}\ \textsc{sub-left}}{\bar{\alpha} \vdash (\mu f.\lambda x.t) : (\chi_1' \to \chi_2')}\ \textsc{fix}$$

- Case PAIR,SUB-PROD: Apply the induction hypothesis for each component, and conclude using rule PAIR.

– Case SUM, SUB-SUM: Apply the induction hypothesis, and conclude using rule SUM.

– Case ∃-INTRO-VAL, SUB-∃: Apply the induction hypothesis and conclude using ∃-INTRO-VAL.

– Case ∀-INTRO-VAL, SUB-∀: Apply the induction hypothesis and conclude using ∀-INTRO-VAL.

– Case Any rule, SUB-HYP: $(\tau_1 \leq \tau_2 \rhd w) \in \Sigma$, so by hypothesis SUB-REDUCE-VAL holds and this suffices to conclude.

– Case REC, SUB-REC-LEFT: The conclusion comes directly from the induction hypothesis. Applying the induction hypothesis is correct because the reasoning goes by induction on v and the conclusion will only be reused on strict subterms of v. (This is true because the recursive type is constructive.)

– Case Any rule, SUB-REC-RIGHT: Apply the induction hypothesis, valid for the same reason as above. Then conclude using rule REC. □

Lemma REDUCE-SUB-MEM

$$\left\{ \begin{array}{l} \theta_1 \leq \theta_2 \rhd w \\ s;\, \mu \vdash v : \theta_1 \angle P \rhd w_1 \end{array} \right. \Rightarrow \exists\, \mu'\, P'\, w_2', \left\{ \begin{array}{l} (w\, w_1) \longrightarrow^+ w_2' \\ s;\, \mu \vdash v : \theta_2 \angle P' \rhd w_2' \\ (s, \mu)_{\setminus P} \sqsubseteq (s, \mu')_{\setminus P'} \end{array} \right.$$

Lemma REDUCE-SUB-CAP

$$\left\{ \begin{array}{l} C_1 \leq C_2 \rhd w \\ s;\, \mu \vdash C_1 \angle P \rhd w_1 \end{array} \right. \Rightarrow \exists\, \mu'\, P'\, w_2', \left\{ \begin{array}{l} (w\, w_1) \longrightarrow^+ w_2' \\ s;\, \mu' \vdash C_2 \angle P' \rhd w_2' \\ (s, \mu)_{\setminus P} \sqsubseteq (s', \mu')_{\setminus P'} \end{array} \right.$$

Proof. By mutual induction. Formally, the proposition should be first strengthened as for the proof of REDUCE-SUB-VAL in order to deal with recursive types. Yet since dealing with the subtyping context brings no further difficulty, the details are omitted here to avoid being distracted from the main matter.

– Case Any rule, SUB-REFL: by assumption.

– Case Any rule, SUB-TOP: apply rule TOP.

– Case MEM-VAL, Any rule: take $\mu' = \mu$ and $P' = P$ then apply lemma REDUCE-SUB-VAL and conclude with rule MEM-VAL.

– Case MEM-LOC, SUB-REF: Apply the induction hypothesis, and conclude using rule MEM-LOC.

– Case MEM-INJ, SUB-SUM: Apply the induction hypothesis, and conclude using rule MEM-INJ.

– Case MEM-PAIR, SUB-PROD: Apply the induction hypothesis, and conclude using rule MEM-PAIR.

– Case MEM-$\exists$-INTRO, SUB-$\exists$: Apply the induction hypothesis and conclude using MEM-$\exists$-INTRO.

– Case MEM-$*$-INTRO, SUB-$*$: Apply the induction hypothesis on the first branch, then on the second branch, and conclude by transitivity of evolution and using rule MEM-$*$-INTRO.

– Case CAP-$\exists$-INTRO, SUB-$\exists$: Apply the induction hypothesis and conclude using CAP-$\exists$-INTRO.

– Case CAP-$*$-INTRO, SUB-$*$: Apply the induction hypothesis on the first branch, then on the second branch, and conclude by transitivity of evolution and using rule CAP-$*$-INTRO.

– Case CAP-SNG, SUB-SNG: Apply the induction hypothesis and conclude using CAP-SNG.

– Case CAP-GRP, SUB-GRP: Apply the induction hypothesis and conclude using CAP-GRP.

Moreover, we prove the subtyping axioms admissible.

– Case FREE: take $\mu' = \mu$ and $P' = \emptyset$ and conclude using CAP-EMPTY.

– Case FOCUS-REF: by inversion (rules CAP-SNG and MEM-REF), P corresponds to the ownership of singleton region σ, of the location l and of the value $v = s[l]$ at type θ. We let $\mu' = \mu \uplus [\sigma_1 \mapsto v]$ for a fresh region σ_1 and $P' = P$. The conclusion is built using rules CAP-$\exists$-INTRO, CAP-$*$-INTRO, CAP-SNG and MEM-REF.

– Case FOCUS-PAIR: similar.

– Case FOCUS-SUM: similar.

– Case NEW-GRP: μ' extends μ by binding fresh names ρ_i to empty maps. P' extends P with that set of region names.

In the remaining rules, capabilities are only rearranged, so we can take $\mu' = \mu$ and $P' = P$.

– Case UNFOCUS-RGN: Use facts obtained by inversion (rules CAP-SNG and CAP-GRP-FOCUSED) to build the conclusion using CAP-GRP.

– Case $\exists$-EMBED: by inversion (rules CAP-SNG then MEM-$\exists$-INTRO) and conclude with CAP-SNG then CAP-$\exists$-INTRO.

– Case $\star$-EMBED: by inversion (rules CAP-SNG then MEM-$*$-INTRO) and conclude with CAP-SNG then CAP-$*$-INTRO.

– Administrative rules: trivial. □

Lemma REDUCE-SUB-OUT

$$\begin{array}{l} \left\{ \begin{array}{l} \chi_1 \leq \chi_2 \rhd w \\ s;\, \mu;\, \bar{\alpha};\, P \models v : \chi_1 \rhd w_1 \end{array} \right. \\ \Rightarrow \\ \exists\, \mu'\, P'\, w_2', \\ \left\{ \begin{array}{l} (w\, w_1) \longrightarrow^+ w_2' \\ s;\, \mu';\, \bar{\alpha};\, P' \models v : \chi_2 \rhd w_2' \\ (s, \mu)_{\setminus P} \sqsubseteq (s, \mu')_{\setminus P'} \end{array} \right. \end{array}$$

Proof. By induction on the subtyping derivation.

– If χ_1 and χ_2 are value types, use lemma REDUCE-SUB-VAL (and by inversion obtain a typing judgment for the value v viewed as a value).

– Case SUB-$\exists$-INTRO: by inversion the typing of v must end with rule OUT-$\exists$-INTRO. Then apply the induction hypothesis, and conclude with OUT-$\exists$-INTRO.

– Case SUB-∗-INTRO: by inversion the typing of v must end with rule OUT-∗-INTRO. For the left branch use the induction hypothesis, then for the right branch use REDUCE-SUB-CAP. Conclude by transitivity, and using rule OUT-∗-INTRO.

– Case SUB-REFL: by assumption.

– Case SNG-CREATE: by assumption, v has type τ. Let $\mu' = \mu \uplus [\sigma \mapsto v]$ for a fresh singleton region σ. Let $P' = (P \uplus \sigma)$. Conclude using rules SNG, OUT-∗-INTRO and OUT-∃-INTRO.

– Case SNG-EXTRACT: by inversion, v has type $[\sigma]$, and region σ contains a single value of type τ. Thus v admits type τ, by lemma MEM-TO-VAL. Conclude taking $\mu' = \mu$ and $P' = P$.

– Case ADOPT-GRP: by inversion, v has type σ, and region σ contains a single value of type θ. Let $m = \mu[\rho]$, let k be the smallest unused key in m, let $m' = m \uplus [k \mapsto v]$ and then $\mu' = \mu[\rho \mapsto \theta']$. Take $P' = P$ and type-check the conclusion using rule CAP-GRP.

– Case FOCUS-GRP: by inversion, v has type ρ and so by rule GRP there exists k such that $v = \mu[\rho][k]$. Let $\mu' = \mu \uplus [\sigma \mapsto v]$ for a fresh singleton region σ. Let $P' = P$. Result is type-checked using rules SNG and FOCUSED.

– Administrative rules: trivial. □

4 Reduction

A well-typed application of a value v_1 to another value v_2 reduces to a term t'. Furthermore, typing is preserved, and the reduction step is simulated by a reduction step in the translated program.

Lemma REDUCTION

$$\begin{array}{l} \left\{\begin{array}{l} (v_1\, v_2) \,/\, s \longrightarrow t' \,/\, s' \\ \mu;\, (\bar{\alpha}_1,\, \bar{\alpha}_2) \vdash v_1 \,:\, (\forall \bar{\alpha}_3 . \chi_1 \to \chi_2) \rhd w_1 \\ s;\, \mu;\, \bar{\alpha}_1;\, P \models v_2 \,:\, [\bar{\alpha}_2 \to \bar{o}_2][\bar{\alpha}_3 \to \bar{o}_3]\, \chi_1 \rhd w_2 \end{array}\right. \\ \Rightarrow \\ \exists\, \mu'\, P'\, u', \\ \left\{\begin{array}{l} (w_1\, w_2) \longrightarrow u' \\ s';\, \mu';\, \bar{\alpha}_1;\, P' \Vdash t' \,:\, [\bar{\alpha}_2 \to \bar{o}_2][\bar{\alpha}_3 \to \bar{o}_3]\, \chi_2 \rhd u' \\ (s, \mu)_{\backslash P} \sqsubseteq (s', \mu')_{\backslash P'} \end{array}\right. \end{array}$$

Proof. By induction on the typing derivation of v_1.

▶ *Case:* BETA

0) In this case, $\bar{\alpha}_3$ is empty, v_1 is of the form $(\mu f . \lambda x . t_1)$ and t' of the form $[f \to v_1][x \to v_2]\, t_1$ and w_1 is of the form $(\mu f . \lambda x . u_1)$ and $s' = s$. The body of the function is typed as follows:

$$\mu;\, (\bar{\alpha}_1,\, \bar{\alpha}_2),\, f : (\chi_1 \to \chi_2),\, x : \chi_1 \Vdash t_1 \,:\, \chi_2 \rhd u_1$$

1) By type substitution (TYP-SUBST) in the above judgment:

$$\mu;\, \bar{\alpha}_1,\, f : [\bar{\alpha}_2 \to \bar{o}_2]\, (\chi_1 \to \chi_2),\, x : [\bar{\alpha}_2 \to \bar{o}_2]\, \chi_1 \Vdash t_1 \,:\, [\bar{\alpha}_2 \to \bar{o}_2]\, \chi_2 \rhd u_1$$

2) By TYP-SUBST applied to the typing assumption on v_1:

$$\mu;\, \bar{\alpha}_1 \vdash v_1 \,:\, ([\bar{\alpha}_2 \to \bar{o}_2]\, \chi_1 \to [\bar{\alpha}_2 \to \bar{o}_2]\, \chi_2) \rhd w_1$$

3) By SUBST-TRM applied to (1) and (2):

$$\mu;\, \bar{\alpha}_1,\, x : [\bar{\alpha}_2 \to \bar{o}_2]\, \chi_1 \Vdash [f \to v_1]\, t_1 \,:\, [\bar{\alpha}_2 \to \bar{o}_2]\, \chi_2 \rhd [f \to w_1]\, u_1$$

4) Now by the typing assumption on v_2:

$$s;\, \mu;\, \bar{\alpha}_1;\, P \models v_2 \,:\, [\bar{\alpha}_2 \to \bar{o}_2]\, \chi_1 \rhd w_2$$

5) By SUBST-OUT on (3) and (4), there exists μ', P' and u' such that:

$$\left\{\begin{array}{l} s;\, \mu';\, \bar{\alpha};\, P' \Vdash [x \to v_2][f \to v_1]\, t \,:\, [\bar{\alpha}_2 \to \bar{o}_2]\, \chi_2 \rhd u' \\ [x \to w_2][f \to w_1]\, u_1 \longrightarrow^* u' \\ (s, \mu)_{\backslash P} \sqsubseteq (s, \mu')_{\backslash P'} \end{array}\right.$$

6) This suffices to conclude, since

$$(w_1\, w_2) = ((\mu f.\lambda x.u_1)\, w_2) \longrightarrow [f \to w_1][x \to w_2]\, u_1 \longrightarrow^* u'$$

▶ *Case:* REF

0) In this case, the function v_1 is ref, and $w_1 = (\lambda x.\, ((), x))$. The argument v_2 has a type of the form τ and is translated as w_2. Moreover, there exists a fresh location l such that $s' = s \uplus [l \mapsto v_2]$ and $t' = l$.

1) By inversion on the typing of v_2 (rule OUT-VAL), $P = \emptyset$ and:

$$\mu;\, \emptyset \vdash v_2 : \tau \rhd w$$

2) Let σ be a fresh singleton region and let $\mu' = \mu \uplus [\sigma \mapsto l]$. Let $P' = (\{l\} \uplus \{\sigma\})$. And let $u' = ((), w_2)$.

3) The reduction $(w_1\, w_2) \longrightarrow u'$ is satified. The relation $(s, \mu)_{\setminus P} \sqsubseteq (s', \mu')_{\setminus P'}$ holds. The typing derivation for the output term is:

$$\dfrac{\dfrac{}{s';\, \mu';\, ;\, \emptyset \models (\mu'[\sigma] = l) : [\sigma] \rhd ()}\ \text{RGN} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mu \sqsubseteq \mu' \qquad \mu;\, \emptyset \vdash v_2 : \tau \rhd w}{\mu';\, \emptyset \vdash v_2 : \tau \rhd w}\ \text{STABLE-VAL}}{s';\, \mu' \vdash (s[l] = v_2) : \tau \measuredangle \emptyset \rhd w_2}\ \text{MEM-VAL}}{s';\, \mu' \vdash (\mu'[\sigma] = l) : \text{ref}\,\tau \measuredangle \{l\} \rhd w_2}\ \text{MEM-LOC}}{s';\, \mu' \vdash \{\sigma : \text{ref}\,\tau\} \measuredangle (\{l\} \uplus \{\sigma\}) \rhd w_2}\ \text{CAP-SNG}}{s';\, \mu';\, ;\, (\{l\} \uplus \{\sigma\}) \models l : ([\sigma] * \{\sigma : \text{ref}\,\tau\}) \rhd ((), w_2)}\ \text{EXE-FRAME}$$
$$\dfrac{}{s';\, \mu';\, ;\, (\{l\} \uplus \{\sigma\}) \models l : \exists\sigma.([\sigma] * \{\sigma : \text{ref}\,\tau\}) \rhd ((), w_2)}\ \text{EXE-}\exists\text{-INTRO}$$

▶ *Case:* GET

1) By inversion on the typing of v_2 (only rules OUT-FRAME then OUT-VAL can apply), and then by inversion on the typing of the capability $\{\sigma : \text{ref}\,\tau\}$ (only rules CAP-SNG then MEM-LOC can apply), and then applying the MEM-TO-VAL lemma, we obtain a typing judgment for the value contained in the store at the location being read. The value w_2 must be a pair of the form $((), w_{22})$.

2) Let $\mu' = \mu$, $P' = P = (\{l\} \uplus \{\sigma\})$, and let $u' = (w_{22}, w_{22})$.

3) The rest follows using similar techniques as for REF.

▶ *Case:* SET

1) As for GET, we obtain a typing judgment for the value contained in the store at the location being read. The value w_2 must be a triple of the form $((), w_{22}, w_{23})$

2) Let $\mu' = \mu$, $P' = P = (\{l\} \uplus \{\sigma\})$, and let $u' = ((), w_{23})$.

3) The rest follows using similar techniques as for REF.

▶ *Case:* PROJ^1

1) By inversion on the typing of v_2 (one rule OUT-CONJ applies), the value v_2 has type σ and its translation w_2 is of the form $((), w_{2c}$. Moreover, the capability $\{\sigma : \tau_1 \times \theta_2\}$ is translated as w_{22}.

2) By inversion of v_2 having type σ (by rules OUT-VAL then SNG), we have $\mu[\sigma] = v_2$.

3) By inversion of the typing of this capability, $\mu[\sigma]$, which is equal to v_2 has memory type $\tau_1 \times \theta_2$. Thus, v_2 must be a pair of the form (v_{21}, w_{22}) and its translation w_{2c} must be a pair of the form (w_{21}, w_{22}).

4) The reduction rule for proj^1 applies, and returns v_{21}. The translation returns w_{21} which is indeed the translation of v_{21} at type τ_1.

5) The function returns as well the input capability, which is translated in the target language as a copy of the translation of the input capability.

6) To conclude, we take $\mu' = \mu$, $P' = P$, and let $u' = (w_{21}, (w_{21}, w_{22}))$.

▶ *Case:* CASE

Similar arguments as for PROJ^i.

5 Simulation

A step of reduction in a well-typed source is reflected by one or more steps of reduction in its translation. During this process, regions can only grow and the term cannot affect or acquire a piece of state that it does not own. (Variable P describes the piece of state owned.)

Lemma SIMULATION

$$\left\{ \begin{array}{l} t \mathbin{/} s \longrightarrow t' \mathbin{/} s' \\ s;\, \mu;\, \bar{\alpha};\, P \parallel\!\models t : \chi \rhd u \end{array} \right. \Rightarrow \exists\, \mu'\, P'\, u',\, \left\{ \begin{array}{l} u \longrightarrow^{+} u' \\ s';\, \mu';\, \bar{\alpha};\, P' \parallel\!\models t' : \chi \rhd u' \\ (s, \mu)_{\setminus P} \sqsubseteq (s', \mu')_{\setminus P'} \end{array} \right.$$

Proof. By induction on the typing derivation of t.

▶ *Case:* EXE-VAL

This case is impossible because a value is not reducible.

▶ *Case:* EXE-APP

The term t is of the form $(v_1\, t_2)$ and u is of the form $(w_1\, w_2)$. The typing derivation ends with an instance of the EXE-APP rule.

$$\frac{\mu;\, \bar{\alpha} \vdash v_1 : (\chi_1 \to \chi) \rhd w_1 \qquad s;\, \mu;\, \bar{\alpha};\, P \Vdash t_2 : \chi \rhd u_2}{s;\, \mu;\, \bar{\alpha};\, P \Vdash (v_1\, t_2) : \chi \rhd (w_1\, u_2)}\ \text{EXE-APP}$$

There are two cases to consider, depending on whether t_2 is already a value or not.

⋆ *Sub-case: The argument is not a value*

0) The only reduction rules that applies is the CTX rule:

$$\frac{t_2 \mathbin{/} s \longrightarrow t_2' \mathbin{/} s'}{(v_1\, t_2) \mathbin{/} s \longrightarrow (v_1\, t_2') \mathbin{/} s'}\ \text{CTX}$$

1) By induction hypothesis, there exists μ', P' and u_2' such that

$$\begin{cases} u \longrightarrow^+ u' \\ s';\, \mu';\, \bar{\alpha};\, P' \Vdash t_2' : \chi_1 \rhd u_2' \\ (s, \mu)_{\setminus P} \sqsubseteq (s', \mu')_{\setminus P'} \end{cases}$$

2) The conclusion comes from the following derivation:

$$\frac{\dfrac{\mu;\, \bar{\alpha} \vdash v_1 : (\chi_1 \to \chi) \rhd w_1}{\mu';\, \bar{\alpha} \vdash v_1 : (\chi_1 \to \chi) \rhd w_1}\ \text{STABLE-VAL}\ \mu \sqsubseteq \mu' \qquad s';\, \mu';\, \bar{\alpha};\, P' \Vdash t_2' : \chi_1 \rhd u_2'}{s';\, \mu';\, \bar{\alpha};\, P' \Vdash (v_1\, t_1') : \chi \rhd (w_1\, u_2')}\ \text{EXE-APP}$$

and the corresponding reduction steps in the target language:

$$(w_1\, u_2) \longrightarrow^+ (w_1\, u_2')$$

⋆ *Sub-case: The argument is a value*

0) Let v_2 stand for the value t_2. The premises of EXE-APP are:

$$\begin{cases} \mu;\, \bar{\alpha} \vdash v_1 : (\chi_1 \to \chi) \rhd w_1 \\ s;\, \mu;\, \bar{\alpha};\, P \Vdash v_2 : \chi_1 \rhd u_2 \end{cases}$$

1) Lemma TERMINATION applied to the second proposition gives μ'', P'' and w_2 such that:

$$\begin{cases} u_2 \longrightarrow^* w_2 \\ s;\, \mu'';\, \bar{\alpha};\, P'' \models v_2 : \chi_1 \rhd w_2 \\ (s, \mu)_{\setminus P} \sqsubseteq (s, \mu'')_{\setminus P''} \end{cases}$$

2) By lemma REDUCTION applied to the typing derivation of v_1 from step (0) and to the typing derivation of v_2 from step (1), there exists μ', P' and u' such that:

$$\begin{cases} (w_1\, w_2) \longrightarrow^+ u' \\ s';\, \mu';\, \bar{\alpha}_1;\, P' \Vdash t' : \chi \rhd u' \\ (s, \mu'')_{\setminus P''} \sqsubseteq (s', \mu')_{\setminus P'} \end{cases}$$

3) Conclusions are obtained as follows:

$$\begin{cases} u \longrightarrow^+ u' & \text{since } u = (w_1\, u_2) \longrightarrow^* (w_1\, w_2) \longrightarrow^+ u' \\ s';\, \mu';\, \bar{\alpha};\, P' \Vdash t' : \chi \rhd u' & \text{from step (2)} \\ (s, \mu)_{\setminus P} \sqsubseteq (s', \mu')_{\setminus P'} & \text{by transitivity, from steps (1) and (2)} \end{cases}$$

▶ *Case:* FRAME

0) The input state is typed as

$$\frac{s;\, \mu;\, \bar{\alpha};\, P_1 \Vdash t : \chi_1 \rhd u_1 \qquad s;\, \mu \vdash C_2 \between P_2 \rhd w_2}{s;\, \mu;\, \bar{\alpha};\, (P_1 \uplus P_2) \Vdash t : (\chi_1 * C_2) \rhd (u_1, w_2)}\ \text{EXE-FRAME}$$

1) By induction hypothesis, there exists μ', P_1' and u_1' such that

$$\begin{cases} u_1 \longrightarrow^+ u_1' \\ s';\, \mu';\, \bar{\alpha};\, P_1' \Vdash t' : \chi_1 \rhd u_1' \\ (s, \mu)_{\setminus P_1} \sqsubseteq (s', \mu')_{\setminus P_1'} \end{cases}$$

2) Because $(s,\mu)_{\setminus P_1} \sqsubseteq (s',\mu')_{\setminus P_1'}$ and since P_1 and P_2 are disjoint sets:

$$(s,\mu) \sqsubseteq_{|P_2} (s',\mu')$$

3) The first conclusion is obtained by reduction under context:

$$u = (u_1, w_2) \longrightarrow^+ (u_1', w_2) = u'$$

The second conclusion comes from the following typing derivation:

$$\dfrac{s;\, \mu';\, \bar{\alpha};\, P_1' \Vdash t' : \chi_1 \rhd u_1' \qquad \dfrac{\dfrac{(s,\mu) \sqsubseteq_{|P_2} (s',\mu')}{s;\, \mu \vdash C_2 \angle P_2 \rhd w_2}\,\textsc{stable-cap}}{s;\, \mu' \vdash C_2 \angle P_2 \rhd w_2}}{s;\, \mu';\, \bar{\alpha};\, (P_1' \uplus P_2) \Vdash t' : (\chi_1 * C_2) \rhd (u_1', w_2)}\,\textsc{exe-frame}$$

The third conclusion $(s,\mu)_{\setminus (P_1 \uplus P_2)} \sqsubseteq (s',\mu')_{\setminus (P_1' \uplus P_2)}$ is a direct consequence of $(s,\mu)_{\setminus P_1} \sqsubseteq (s',\mu')_{\setminus P_1'}$.

▶ *Case:* EXE-SUB

0) The input state is typed as

$$\dfrac{s;\, \mu;\, \bar{\alpha};\, P \Vdash t : \chi_1 \rhd u_1 \qquad \chi_1 \leq \chi \rhd w}{s;\, \mu;\, \bar{\alpha};\, P \Vdash t : \chi \rhd (w\, u_1)}\,\textsc{exe-sub}$$

1) By induction hypothesis, there exists μ', P' and u_1' such that

$$\begin{cases} u_1 \longrightarrow^+ u_1' \\ s';\, \mu';\, \bar{\alpha};\, P' \Vdash t' : \chi_1 \rhd u_1' \\ (s,\mu)_{\setminus P} \sqsubseteq (s',\mu')_{\setminus P'} \end{cases}$$

2) Conclusions are obtained by the derivation:

$$\dfrac{s;\, \mu';\, \bar{\alpha};\, P' \Vdash t : \chi_1 \rhd u_1' \qquad \chi_1 \leq \chi \rhd w}{s;\, \mu';\, \bar{\alpha};\, P' \Vdash t : \chi \rhd (w\, u_1')}\,\textsc{exe-sub}$$

and the reduction sequence:

$$(w\, u_1) \longrightarrow^+ (w\, u_1')$$

▶ *Cases:* EXE-∃-INTRO, EXE-∀-INTRO, EXE-∀-ELIM

Trivial by induction hypothesis.

When the source term has reached a value, its translation converges to a corresponding value after a finite number of steps.

Lemma TERMINATION

$$\left\{\ s;\,\mu;\,\bar{\alpha};\,P\ \Vdash\ v\ :\ \chi\ \rhd\ u\quad\Rightarrow\ \exists\,\mu'\,P'\,w',\ \begin{cases} u\longrightarrow^{*} w' \\ s;\,\mu';\,\bar{\alpha};\,P'\ \models\ v\ :\ \chi\ \rhd\ w' \\ (s,\mu)_{\backslash P}\ \sqsubseteq\ (s,\mu')_{\backslash P'} \end{cases}\right.$$

Proof. By induction on the typing derivation.

▶ *Case:* EXE-VAL

In this case, u is a value. Let $\mu' = \mu$ and $P' = P$ and $w' = u$. The typing derivation for v is constructed using OUT-VAL.

▶ *Case:* EXE-APP

This case is impossible because an application is not a value.

▶ *Case:* EXE-FRAME

0) The input state is typed as

$$\frac{s;\,\mu;\,\bar{\alpha};\,P_1\ \Vdash\ v\ :\ \chi_1\ \rhd\ u_1 \qquad s;\,\mu\ \vdash\ C_2\ \angle\ P_2\ \rhd\ w_2}{s;\,\mu;\,\bar{\alpha};\,(P_1\uplus P_2)\ \Vdash\ v\ :\ (\chi_1 * C_2)\ \rhd\ (u_1,w_2)}\ \text{EXE-FRAME}$$

1) By induction hypothesis, there exists μ', P_1' and w_1 such that

$$\begin{cases} u_1\longrightarrow^{*} w_1 \\ s;\,\mu';\,\bar{\alpha};\,P_1'\ \models\ v\ :\ \chi\ \rhd\ w \\ (s,\mu)_{\backslash P}\ \sqsubseteq\ (s,\mu')_{\backslash P_1'} \end{cases}$$

2) Because $(s,\mu)_{\backslash P_1}\ \sqsubseteq\ (s,\mu')_{\backslash P_1'}$ and since P_1 and P_2 are disjoint sets:

$$(s,\mu)\ \sqsubseteq_{|P_2}\ (s,\mu')$$

3) Now let $\mu' = \mu'$ and $P' = (P_1'\uplus P_2)$ and $w' = (w_1,w_2)$. The reduction sequence in the target language is given by:

$$u = (u_1,w_2)\longrightarrow^{*}(w_1,w_2) = w'$$

The typing derivation for the output is built as:

$$\dfrac{s;\,\mu';\,\bar{\alpha};\,P_1' \models v : \chi_1 \rhd w_1 \qquad \dfrac{\dfrac{(s,\mu) \sqsubseteq_{|P_2} (s,\mu')}{s;\,\mu \vdash C_2 \angle P_2 \rhd w_2}\,\text{STABLE-CAP}}{s;\,\mu' \vdash C_2 \angle P_2 \rhd w_2}}{s;\,\mu';\,\bar{\alpha};\,(P_1' \uplus P_2) \models v : (\chi_1 * C_2) \rhd (w_1, w_2)}\,\text{OUT-FRAME}$$

The third conclusion $(s,\mu)_{\backslash(P_1 \uplus P_2)} \sqsubseteq (s,\mu')_{\backslash(P_1' \uplus P_2)}$ is a direct consequence of $(s,\mu)_{\backslash P_1} \sqsubseteq (s,\mu')_{\backslash P_1'}$.

▶ *Case:* EXE-SUB

0) The input state is typed as

$$\dfrac{s;\,\mu;\,\bar{\alpha};\,P \Vdash v : \chi_1 \rhd u_1 \qquad \chi_1 \leq \chi \rhd w}{s;\,\mu;\,\bar{\alpha};\,P \Vdash v : \chi \rhd (w\,u_1)}\,\text{EXE-SUB}$$

1) By induction hypothesis, there exists μ'', P'' and w_1 such that

$$\begin{cases} u_1 \longrightarrow^* w_1 \\ s;\,\mu'';\,\bar{\alpha};\,P'' \models v : \chi_1 \rhd w_1 \\ (s,\mu)_{\backslash P} \sqsubseteq (s,\mu'')_{\backslash P''} \end{cases}$$

2) By lemma REDUCE-SUB-OUT there exists μ', P' and w' such that

$$\begin{cases} (w\,w_1) \longrightarrow^+ w' \\ s;\,\mu';\,\bar{\alpha};\,P' \models v : \chi \rhd w' \\ (s,\mu'')_{\backslash P''} \sqsubseteq (s,\mu')_{\backslash P'} \end{cases}$$

3) The reduction in the target language is

$$(w\,u_1) \longrightarrow^* (w\,w_1) \longrightarrow w'$$

the output state is typed as

$$s;\,\mu';\,\bar{\alpha};\,P' \models v : \chi \rhd w'$$

and the inclusion $(s, \mu)_{\setminus P} \sqsubseteq (s, \mu')_{\setminus P'}$ is obtained by transitivity.

▶ *Case:* EXE-∃-INTRO

Apply the induction hypothesis, and conclude using rule OUT-∃-INTRO.

▶ *Case:* EXE-∀-INTRO

Apply the induction hypothesis, and conclude using rule OUT-∀-INTRO.

▶ *Case:* EXE-∀-ELIM

0) The typing derivation ends with:

$$\frac{s;\, \mu;\, \bar{\alpha};\, P \;\|\!\models\; v \;:\; (\forall \alpha.\tau) \;\rhd\; u}{s;\, \mu;\, \bar{\alpha};\, P \;\|\!\models\; v \;:\; ([\alpha \to o]\,\tau) \;\rhd\; u}\ \text{EXE-}\forall\text{-ELIM}$$

1) By induction hypothesis, there exists μ', P' and w' such that:

$$\begin{cases} u \longrightarrow^* w' \\ s;\, \mu';\, \bar{\alpha};\, P' \models v \;:\; (\forall \alpha.\tau) \;\rhd\; w' \\ (s, \mu)_{\setminus P} \sqsubseteq (s, \mu')_{\setminus P'} \end{cases}$$

2) By inversion of typing (only rule OUT-VAL applies), $P' = \emptyset$ and:

$$\mu';\, \bar{\alpha} \vdash v \;:\; (\forall \alpha.\tau) \;\rhd\; w'$$

3) By type substitution (TYP-SUBST), it follows:

$$\mu';\, \bar{\alpha} \vdash v \;:\; ([\alpha \to o]\,\tau) \;\rhd\; w'$$

4) It remains to build the typing derivation for the output v:

$$\frac{\mu';\, \bar{\alpha} \vdash v \;:\; ([\alpha \to o]\,\tau) \;\rhd\; w'}{s;\, \mu';\, \bar{\alpha};\, \emptyset \models v \;:\; ([\alpha \to o]\,\tau) \;\rhd\; w'}\ \text{OUT-VAL}$$

6 Additional Results

A program well-typed in System F is typable in the system and its functional translation is the program itself.

Lemma CONSERVATIVITY-OVER-F

$$\begin{array}{lcl} \Delta \vdash_{F_\mu} v : \tau & \Rightarrow & \Delta \vdash v : \tau \rhd v \\ \Delta \vdash_{F_\mu} t : \tau & \Rightarrow & \Delta \Vdash t : \tau \rhd t \end{array}$$

Proof. Trivial by induction on the typing derivation, and using the two admissible rules:

$$\begin{array}{lcll} \mathrm{proj}^i & : & \tau_1 \to \tau_2 \to \tau_i & \rhd \quad \mathrm{proj}^i \\ \mathrm{case} & : & (\tau_1 \to \tau) \to (\tau_2 \to \tau) \to (\tau_1 + \tau_2) \to \tau & \rhd \quad \mathrm{case} \end{array}$$

Any translated program is well-typed in System F, and admit for type the translation of the type of its source program.

Lemma TRANSLATION-PRESERVES-TYPING

$$\begin{array}{lcl} \Delta \vdash v : \tau \rhd w & \Rightarrow & [\![\Delta]\!] \vdash_{F_\mu} w : [\![\tau]\!] \\ \Gamma \Vdash t : \chi \rhd u & \Rightarrow & [\![\Gamma]\!] \vdash_{F_\mu} u : [\![\chi]\!] \\ s; \mu; \bar{\alpha}; P \Vdash\!\models t : \chi \rhd u & \Rightarrow & \vdash_{F_\mu} u : [\![\chi]\!] \\ s; \mu; \bar{\alpha}; P \models v : \chi \rhd w & \Rightarrow & \vdash_{F_\mu} w : [\![\chi]\!] \\ s; \mu \vdash v : \theta \angle P \rhd w & \Rightarrow & \vdash_{F_\mu} w : [\![\theta]\!] \\ s; \mu \vdash C \angle P \rhd w & \Rightarrow & \vdash_{F_\mu} w : [\![C]\!] \\ \tau_1 \leq \tau_2 \rhd w & \Rightarrow & \vdash_{F_\mu} w : [\![\tau_1]\!] \to [\![\tau_2]\!] \\ \theta_1 \leq \theta_2 \rhd w & \Rightarrow & \vdash_{F_\mu} w : [\![\theta_1]\!] \to [\![\theta_2]\!] \\ C_1 \leq C_2 \rhd w & \Rightarrow & \vdash_{F_\mu} w : [\![C_1]\!] \to [\![C_2]\!] \\ \chi_1 \leq \chi_2 \rhd w & \Rightarrow & \vdash_{F_\mu} w : [\![\chi_1]\!] \to [\![\chi_2]\!] \end{array}$$

Proof. All judgments are immediate by induction. Notice that these properties hold by design of the system. □