Simplified Type Inference

Tuesday, August 12, 2014 8:50 PM

In this document, we describe a simplified rendition of the region type inference algorithm from previous wiki. Basically, we describe top-level elaboration process (i.e., elaboration of class header, constructor and methods) in form of a functional program, instead of using inference rules to described the elaboration. For expression and statement elaboration however, we retain the inference rule approach as it is already simple and easy to understand. Also, the section on nature of generated constraints and examples section do not require any changes, and are not reproduced in this wiki.

The Source Language

$cn \in Class\ Names\ (A, B, C \dots)$
$mn \in Method\ Names\ (m, n, \dots)$
$x, f \in Variables, fields$
$n \in Integers$
$Program = (CT, e)$
$c ::= n \mid () \mid true \mid false \mid Null$ //Constants
$N ::= cn\langle\overline{T}\rangle$ //Instantiated class type
$C ::= \text{class } cn\langle\overline{\alpha \lhd N}\rangle \lhd N \{\overline{T\ f}; k\ ; \overline{d}\}$ //Class Definitions
$k ::= cn\ (\overline{T\ x})\{\text{ super }(\overline{v});\ \overline{\text{this}.f = v;}\}$ //Constructors
$d ::= T\ mn\ (\overline{T\ x})\ \{s;\ \text{return } e;\}$ //Methods
$T ::= \alpha \mid N \mid Object \mid Region\langle T\rangle \mid int \mid bool \mid unit$ // Types
$v ::= c \mid x \mid new\ N(\overline{v})$
$s ::= \cdot \mid let\ T\ x = e \mid x = e \mid e.f = e \mid letregion\ \{ s \} \mid open\ e\ \{ s \}$
$\mid open^a\ e\ \{ s \} \mid s; s \mid e.set(e) \mid e.transfer() \mid e.giveUp()$
$e ::= c \mid x \mid e.f \mid e.mn(\overline{e}) \mid \text{new } N(\overline{e}) \mid (N)\ e \mid e.get()$ //Expressions

The Target Language

$\rho, p \in region\ names$
$cn \in Class\ Names\ (A, B, C \dots)$
$mn \in Method\ Names\ (m, n, \dots)$
$x, f \in Variables, fields$
$n \in Integers$
$Program = (CT, e)$
$c ::= n \mid () \mid true \mid false \mid Null$ //Constants
$N ::= cn\langle p^a \overline{p} \mid \phi\rangle\langle\overline{\tau}\rangle$ //Instantiated class type
$C ::= \text{class } cn\langle \rho^a \overline{\rho}\rangle\langle\overline{\alpha \lhd N}\rangle \lhd N \{\overline{\tau\ f}; k\ ; \overline{d}\}$ //Class Definitions
$k ::= cn\ (\overline{\tau\ x})\{\text{ super }(\overline{v});\ \overline{\text{this}.f = v;}\}$ //Constructors
$d ::= \tau\ mn\langle \rho^a \overline{\rho} \mid \phi\rangle\ (\overline{\tau\ x})\ \{s;\ \text{return } e;\}$ //Methods
$\phi ::= true \mid \rho \succcurlyeq \rho \mid \rho = \rho \mid \phi \wedge \phi$ //Outlives constraints on region params
$\tau_{\lhd} ::= \alpha \mid N$
$\tau ::= \tau_{\lhd} \mid Object\langle p^a\rangle \mid Region[\rho]\langle p^a\rangle\langle\tau\rangle \mid int \mid bool \mid unit$ //Last 3 are unboxed
$v ::= c \mid x \mid new\ N(\overline{v})$
$s ::= \cdot \mid let\ \tau\ x = e \mid x = e \mid e.f = e \mid letregion\langle\rho\rangle\ \{ s \} \mid open\ e\ \{ s \}$
$\mid open^a\ e\ \{ s \} \mid s; s \mid e.set(e) \mid e.transfer() \mid e.giveUp()$
$e ::= c \mid x \mid e.f \mid e.mn\langle p^a \overline{p}\rangle(\overline{e}) \mid \text{new } N(\overline{e}) \mid (N)\ e \mid e.get() \mid newRgn\langle\rho\rangle\langle\tau\rangle()$ //Expressions

Elaboration (Algorithm HM(ρ))

- The function `elaborate` describes an algorithm ($HM(\rho)$) to elaborate basic class definition to a class definition with region-annotated types (hereafter called as the elaborated definition). The algorithm generates constraints over region variables such that the elaborated definition is well-formed if and only if constraints are satisfiable. HM(ρ) uses a separate constraint solving algorithm (accessible through $normalize$ function) to solve constraints. The nature of constraints and constraint solving is described later in this wiki.
- the top-level `elaborate` function populates the class table (CT') with the elaborated definition of B. It makes use of `elaborate-header`, `elaborate-cons`, and `elaborate-methods` functions which elaborate header (signature and instance variables) of B, the constructor of B, and methods of B respectively. The three functions represent three kinds of occassions on which constraints are solved and solution is applied - after elaborating the header, after elaborating the constructor, and each time a method is elaborated.
- Rules make use of an environment Γ to map variables to their region-annotated types, an environment Δ to map type variables to their bounds, and a set Σ of region variables in scope.
- We define $bound_\Delta$ function over types (τ). For a given type, the $bound_\Delta$ function identifies the class where we need to look for fields or methods.
  $bound_\Delta(\alpha) = \Delta(\alpha)$
  $bound_\Delta(N) = N$
  $bound_\Delta(T) = T$

`fun elaborate(B) =`
  `let`
    `hdB = elaborate-header(B)`
    `consB = elaborate-cons(B,hdB)`
    `fullB = elaborate-methods(B,consB)`
  `in`
    `CT'[B ↦ fullB]`
  `end`

`fun elaborate-header(B) =`
  `let`
    $\text{class } B\langle\overline{\alpha \lhd N_s}\rangle \lhd N_s \{\overline{T\, f}; k_s ; \overline{d_s}\}$ `= CT(B)`
    $\text{class } B\langle\rho^a\overline{\rho} \mid \top\rangle\langle\overline{\alpha \lhd N}\rangle \lhd N \{\overline{\tau\, f}\}$ `= header-template(B)`
    `C1 = type-ok(`$\overline{N}$`)`
    `C2 = type-ok(`N`)`
    `C3 = type-ok(`$\overline{\tau}$`)`
    `C = C1 ∧ C2 ∧ C3 ∧` $\overline{\rho} \succcurlyeq \rho^a$
    `(D,`ψ_i`) = normalize(C)`
    $\overline{N_T} = \psi_i(\overline{N})$
    $N_T = \psi_i(N)$
    $\overline{\tau_T} = \psi_i(\overline{\tau})$
    $\rho^a_T = \psi_i(\rho^a)$
    $\overline{\rho_T} =$ `(frv(`$\overline{N_T}, N_T, \overline{\tau_T}$`))` $- \{\rho^a_T\}$
    $\phi = D - \{\overline{\rho_T} \succcurlyeq \rho^a_T\}$ (* We need not record implicit constraints*)
  `in`
    $\text{class } B\langle\rho^a_T\overline{\rho_T} \mid \phi\rangle\langle\overline{\alpha \lhd N_T}\rangle \lhd N_T \{\overline{\tau_T\, f}\}$
  `end`

`fun header-template (B) =`
  `let`
    $\text{class } B\langle\overline{\alpha \lhd N_s}\rangle \lhd N_s \{\overline{T\, f}; k_s ; \overline{d_s}\}$ `= CT(B)`
    $\overline{{}^{X}N} =$ `templateTy(`$\overline{N_s}$`)` (* templateTy is an auxiliary fn defined at the end *)

$^{X}\mathrm{N} = \texttt{templateTy}(\mathrm{N_s})$

$\overline{^{X}\tau} = \texttt{templateTy}(\overline{T})$

$\rho^{a} = \texttt{allocRgn}(^{X}\mathrm{N})$

$\overline{\rho} = (\texttt{frv}(\overline{^{X}N}, {^{X}N}, \overline{^{X}\tau})) - \{\rho^{a}\}$

$\psi_i = [B\langle\rho^a\overline{\rho}\rangle\langle\overline{\alpha}\rangle / B\langle\overline{\alpha}\rangle]$ (* templateTy does not templatize recursive occurances of B, because it doesn't know how many region params are there for B. But, now we know. We substitute the region annotated type of B for its simple type in the class defn. *)

$\overline{\mathrm{N}} = \psi_i(\overline{^{X}\mathrm{N}})$

$\mathrm{N} = \psi_i(^{X}\mathrm{N})$

$\overline{\tau} = \psi_i(\overline{^{X}\tau})$

`in`

class $B\langle\rho^a\overline{\rho} \mid \top\rangle\langle\overline{\alpha \lhd N}\rangle \lhd N\ \{\overline{\tau\ f}\}$

`end`

`fun elaborate-cons(B, hdB) =`

`let`

class $B\langle\overline{\alpha \lhd N_s}\rangle \lhd N_s\ \{\overline{T\ f}; k_s; \overline{d_s}\}$ `= CT(B)`

class $B\langle\rho_B^a\overline{\rho_B} \mid \phi_B\rangle\langle\overline{\alpha \lhd N_B}\rangle \lhd N_B\ \{\overline{\tau_B\ f}\}$ `= hdB`

$\overline{\tau_A}$ `= ctype(`N_B`)` (* Types of super class constructor args *)

$B(\overline{T_x\ x})\{\mathrm{super}(\overline{\upsilon_g});\ \overline{\mathrm{this}.f = \upsilon_f;}\}\ =\ k_s$

$\overline{\tau_a} = \texttt{templateTy}(\overline{T_x})$

`Ca = type-ok(`$\overline{\tau_a}$`)`

`_ = CT'[B` $\mapsto$ class $B\langle\rho_B^a\overline{\rho_B} \mid \phi\rangle\langle\overline{\alpha \lhd N_B}\rangle \lhd N_B\ \{\overline{\tau_B\ f;}\}$`]` (* temporarily update CT' so that "this.f" gives correct type for any field f of B*)

$\Gamma\ =\ \cdot, \mathrm{this}{:}\ B\langle\rho_B^a\overline{\rho_B}\rangle\langle\overline{\alpha}\rangle, \overline{x{:}\ \tau_x}$

$\Sigma = \rho_B^a \cup \overline{\rho_B}$

$\Delta = \overline{\alpha \lhd N_B}$

$(\overline{\upsilon_g'{:}\tau_g}$`, Cg) = elab-expr(`$\Sigma;\ \Delta;\ \Gamma;\ \rho_B^a \vdash \overline{\upsilon_g}$`)`

`Csub = subtype-ok(`$\Delta \vdash \overline{\tau_g} <: \overline{\tau_a}$`)` (* Actual types of args to super should be subtype of expected types. *)

$(\overline{\mathrm{this}.f = \upsilon_f'}$`, _, Cf) = elab-stmt(`$\Sigma;\ \Delta;\ \Gamma;\ \rho_B^a \vdash \overline{\mathrm{this}.f = \upsilon_f}$`)`

`C = Ca` $\wedge$ `Cg` $\wedge$ `Csub` $\wedge$ `Cf` $\wedge$ $(\overline{\rho_B} \succcurlyeq \rho_B^a) \wedge \phi_B$

`(D,`ψ_i`) = normalize(C)`

$(\overline{N}, N, \overline{\tau}, \overline{\tau_x}, \rho^a) = (\psi_i(\overline{N_B}), \psi_i(N_B), \psi_i(\overline{\tau_B}), \psi_i(\overline{\tau_a}), \psi_i(\rho_B^a))$

$\overline{\rho}$ `= (frv(`$\overline{N}, N, \overline{\tau}$`))` $- \{\rho^a\}$

ϕ `= project-constraints(D,{`$\rho^a, \overline{\rho}$`}) - (`$\overline{\rho} \succcurlyeq \rho^a$`)` (* Collect residual constraints over region params of class B that need to be recorded explicitly as refinement *)

$(\overline{\upsilon_g''},\ \overline{\upsilon_f''})\ =\ (\psi_i(\overline{\upsilon_g'}),\ \psi_i(\overline{\upsilon_f'}))$

`k =` $B(\overline{\tau_x\ x})\{\mathrm{super}(\overline{\upsilon_g''});\ \overline{\mathrm{this}.f = \upsilon_f'';}\}$

`in`

class $B\langle\rho^a\overline{\rho} \mid \phi\rangle\langle\overline{\alpha \lhd N}\rangle \lhd N\ \{\overline{\tau\ f}; k\}$

`end`

```
fun elaborate-methods (B, consB) =
    elaborate-methods-rec (CT(B), consB)

fun elaborate-methods-rec(Bdef, consB) = case Bdef of
```

class $B\langle\overline{\alpha \lhd N_s}\rangle \lhd N_s\ \{\overline{T\ f}; k_s;\}$ `=> consB` (* If there are no methods, we are done *)

`|` class $B\langle\overline{\alpha \lhd N_s}\rangle \lhd N_s\ \{\overline{T\ f}; k_s; \overline{d_s}d_s\}$ `=>`

`let`

`fullB' = elaborate-methods-rec (`class $B\langle\overline{\alpha \lhd N_s}\rangle \lhd N_s\ \{\overline{T\ f}; k_s; \overline{d_s}\}$`, consB)`

class $B\langle \rho^a_B \overline{\rho_B} \mid \phi \rangle \langle \overline{\alpha \lhd N_B} \rangle \lhd N_B \{\overline{\tau_B\ f};\ k;\ \overline{d_B}\}$ = fullB'
(* Our task is to elaborate method d_s*)
$T_r\ m(\overline{T_x\ x})\{s; \text{return } e;\}\ =\ d_s$
τ_p = templateTy(T_r)
$\overline{\tau_a}$ = templateTy(T_x)
$(\pi^a, \rho^a_m, \overline{\pi})$ = (new(), new(), frv($\tau_p, \overline{\tau_a}$)) (* π^a denotes allocation context param of "m".
ρ^a_m is to be used as a dummy variable to facilitate the unification of allocation contexts for recursive calls of "m" with π^a. In other words, no region polymorphic recursion *)
$d_t = \tau_p\ m\langle \rho^a_m | \rho^a_m = \pi^a \rangle (\overline{\tau_a\ x})\{\cdot\}$ (* We use this type of "m" to typecheck recursive applications.
Body of "m" is insignificant; We denote it with a hole. *)
_ = CT'[B $\mapsto$ class $B\langle \rho^a_B \overline{\rho_B} \mid \phi \rangle \langle \overline{\alpha \lhd N_B} \rangle \lhd N_B \{\overline{\tau_B\ f};\ k;\ \overline{d_B} d_t\}$] (* temporarily update CT' so that "this.m" gives correct type*)
$\Gamma\ =\ \cdot, \text{this}: B\langle \rho^a_B \overline{\rho_B} \rangle \langle \overline{\alpha} \rangle, x: \overline{\tau_a}$
$\Sigma = \rho^a_B \cup \overline{\rho_B} \cup \pi^a \cup \overline{\pi}$
$\Delta = \overline{\alpha \lhd N_B}$
(s', Γ', Cs) = elab-stmt($\Sigma;\ \Delta;\ \Gamma; \pi^a \vdash s$)
(e' : τ_q, Ce) = elab-expr($\Sigma;\ \Delta;\ \Gamma'; \pi^a \vdash e$)
Csub = subtype-ok($\Delta \vdash \tau_q <: \tau_p$) (* Actual return type must be subtype of expected return type *)
C = Cs $\wedge$ Ce $\wedge$ Csub $\wedge (\overline{\rho_B} \succcurlyeq \rho^a_B) \wedge \phi_B$ (* Set of all constraints *)
(D, ψ_i) = normalize(C)
$(\overline{N}, N, \overline{\tau}, \overline{\tau_x}, \tau_r, \rho^a) = (\psi_i(\overline{N_B}), \psi_i(N_B), \psi_i(\overline{\tau_B}), \psi_i(\overline{\tau_a}), \psi_i(\tau_p), \psi_i(\rho^a_B))$
$\overline{\rho}$ = (frv ($\overline{N}, N, \overline{\tau}$)) $- \{\rho^a\}$
Σ_ρ = $\rho^a \cup \overline{\rho}$ (* ρ^a and $\overline{\rho}$ are new region vars that replace ρ^a_B and $\overline{\rho_B}$ as region params of class B *)
(ρ^a_m, ϕ^a_m) = if $\psi_i(\pi^a) \in \Sigma_\rho$
then $(\pi^a, \pi^a = \psi_i(\pi^a))$ (* If allocation ctxt for method is required to be one of the
preexisting regions, then record it explicitly as an equality
constraint over allocation context parameter. *)
else $(\psi_i(\pi^a), \top)$ (* Else, simply do the substitution *)
$\overline{\rho_m}$ = (frv($\overline{\tau_x}, \tau_r$)) $- \{\rho^a_m\}$
Σ_π = $\rho^a_m \cup \overline{\rho_m}$
ϕ = project-constraints(D, Σ_ρ) - ($\overline{\rho} \succcurlyeq \rho^a$) (* Explicit constraints over region params of B *)
ϕ_m = project-constraints(D, $\Sigma_\rho \cup \Sigma_\pi$) (* Constraints over region params of method m *)
(s'', e'') = (ψ_i(s'), ψ_i(e'))
d = $\tau_x\ m\langle \rho^a_m \overline{\rho_m} | \phi^a_m \wedge \phi_m \rangle (\overline{\tau_x\ x})$ {$s''; return\ e''$}
in
class $B\langle \rho^a \overline{\rho} \mid \phi \rangle \langle \overline{\alpha \lhd N} \rangle \lhd N \{\overline{\tau\ f};\ k;\ \overline{d} d\}$
end

```
(* AUXILIARY FUNCTIONS *)

fun templateTy(T) = case T of
  α|int|bool|unit => T
| Object => Object<ρ> where new(ρ)
| A(T̄) => if A ∈dom(CT') ∧CT'(A) = class A⟨ρ^a ρ̄ | ϕ⟩⟨α ◁ N⟩ ◁ N then
```

```
            then A<πᵃπ̄><τ̄> where new(πᵃπ̄)∧ |π̄| = |ρ̄| ∧ τ̄ = templateTy(T̄)
            else T
| Region<T_root> =>let τ′= templateTy(T_root) in
                   let τ_root= [ρ/frv(τ′)] τ′ where new(ρ) in
                       Region[ρ]<π><τ_root> where new(π)

fun superClasses(B<πᵃπ̄><τ̄>) = case B of
  Object => {}
| Region[ρ] => Object<πᵃ>
| _ =>
    let class B⟨ρᵃρ̄ | ϕ⟩⟨α ⊲ N̄⟩ ⊲ N  = CT'(B)in
    let N' = [π̄/ρ̄][πᵃ/ρᵃ] N in
        {N'} ∪ superClasses(N')
  | superClasses _ => error()

fun allocRgn(B<πᵃπ̄><τ̄>)= πᵃ
  | _ => error()

fun project-constraints (D,S) = case D of
  true => D
| φ ∧ D'=>
    let ϕ = project-constraints (D',S) in
      if frv(φ)⊆ S then φ ∧ ϕ else ϕ
```

Auxiliary judgments

For `type-ok`, `subtype-ok`, `elab-expr` and `elab-stmt`, we retain the judgment notation that we had in previous wiki, for they are already simple and easy to understand. As usual, the judgments are of form $Ctxt; C \vdash Q$ denoting that Q is derivable under context $Ctxt$, given that constraint C is satisfied.

- $C \vdash \tau\ OK$ is equivalent to saying that C = `type-ok(τ)`.
- $\Delta; C \vdash \tau_1 <: \tau_2$ is equivalent to saying that C = sub`type-ok(Δ ⊢ τ₁ <: τ₂)`
- $\Sigma; \Delta; \Gamma; \rho^a; C \vdash e \hookrightarrow e' : \tau$ is equivalent to saying that $(e' : \tau, C)$ = `elab-expr(Σ; Δ; Γ ; ρᵃ ⊢e)`
- $\Sigma; \Delta; \Gamma; \rho^a; C \vdash s \hookrightarrow s' \dashv \Gamma'$ is equivalent to saying that (s', Γ', C) = `elab-expr(Σ; Δ; Γ ; ρᵃ ⊢s)`.

$\boxed{C \vdash \tau\ OK}$

$$\top \vdash \alpha\ OK, int\ OK, bool\ OK \qquad \top \vdash Object\langle\rho\rangle\ OK \qquad \frac{frv(\tau)=\{\rho\}}{\top \vdash Region[\rho]\langle\rho^a\rangle\langle\tau\rangle\ OK}$$

$$\frac{\begin{array}{c} CT'(B)=class\ B\langle\rho^a\bar{\rho} \mid \phi\rangle\langle\overline{\alpha\lhd N}\rangle\lhd N\ \{\ldots\} \\ C_\tau\vdash\bar{\tau}\ OK \quad \Delta=\overline{\alpha\lhd N} \quad new(\pi^a,\bar{\pi}) \quad |\bar{\pi}|=|\bar{\rho}| \quad \psi=[\bar{\pi}/\bar{\rho}][\pi^a/\rho^a] \\ \psi'=\psi\ o\ [\bar{\tau}/\bar{\alpha}] \quad C_\phi=\psi(\phi) \quad \Delta;C_\lhd\vdash\bar{\tau}<:\psi'(\bar{N}) \quad C=C_\tau\wedge C_\lhd\wedge C_\phi\wedge(\bar{\pi}\succcurlyeq\pi^a) \end{array}}{C\vdash B\langle\pi^a\bar{\pi}\rangle\langle\bar{\tau}\rangle\ OK}$$

$\boxed{\Delta;\ C \vdash \tau_1 <: \tau_2}$

$$\Delta; \top \vdash \tau <: \tau \qquad \Delta;\ \pi^a = \rho^a \vdash\ B\langle\pi^a\bar{\pi}\rangle\langle\bar{\tau}\rangle\ <:\ Object\langle\pi^a\rangle$$

$$\frac{\Delta; C \vdash \Delta(\alpha) <: \tau_2}{\Delta; C \vdash \alpha <: \tau_2} \qquad \frac{A\langle \pi_2^a \overline{\pi_2}\rangle\langle\overline{\tau_2}\rangle \in SuperClasses(B\langle\pi^a\overline{\pi}\rangle\langle\overline{\tau}\rangle)}{\Delta; \left(A\langle\pi_1^a\overline{\pi}_1\rangle\langle\overline{\tau}_1\rangle = A\langle\pi_2^a\overline{\pi_2}\rangle\langle\overline{\tau_2}\rangle\right) \vdash B\langle\pi^a\overline{\pi}\rangle\langle\overline{\tau}\rangle <: A\langle\pi_1^a\overline{\pi}_1\rangle\langle\overline{\tau}_1\rangle}$$

$$\boxed{\Sigma; \Delta;\ \Gamma; \rho^a; C \vdash e \hookrightarrow e' : \tau}$$

$$\frac{x{:}\tau \in \Gamma}{\Sigma;\Delta;\Gamma;\rho^a;\top \vdash x \hookrightarrow x{:}\tau} \qquad \frac{\begin{array}{c}\Sigma;\Delta;\Gamma;\rho^a;C \vdash e \hookrightarrow e'{:}\tau' \\ f{:}\tau \in fields(bound_\Delta(\tau'))\end{array}}{\Sigma;\Delta;\Gamma;\rho^a;C \vdash e.f \hookrightarrow e'.f{:}\tau}$$

$$\frac{\begin{array}{c}\rho \in \Sigma \\ \Sigma;\Delta;\Gamma;\rho^a;C_R \vdash e \hookrightarrow e'{:}Region[\rho]\langle\pi^a\rangle\langle\tau\rangle\end{array}}{\Sigma;\Delta;\Gamma;\rho^a;C_R \vdash e.get() \hookrightarrow e'.get(){:}\tau}$$

$$\frac{\begin{array}{c}N = templateTy(N_s) \quad C_N \vdash N\ OK \\ \pi^a = allocRgn(N) \quad \overline{\tau_B} = ctype(N) \quad \Sigma;\Delta;\Gamma;\rho^a;C_e \vdash \bar{e} \hookrightarrow \overline{e_1}{:}\overline{\tau_e} \\ \Delta; C_\lhd \vdash \overline{\tau_e} <: \overline{\tau_B} \quad C = C_N \wedge C_e \wedge C_\lhd \wedge (\rho^a \succcurlyeq \pi^a)\end{array}}{\Sigma;\Delta;\Gamma;\rho^a;C \vdash new\ N(\bar{e}) \hookrightarrow new\ N(\overline{e_1}){:}N} \qquad \frac{eraseRgn\ (\tau) \lhd Object}{\Sigma;\Delta;\Gamma;\rho^a;\top \vdash Null{:}\tau}$$

$$\frac{\begin{array}{c}\Sigma;\Delta;\Gamma;\rho^a;C_0 \vdash e_0 \hookrightarrow e_0'{:}\tau_0 \quad mtype\left(m, bound_\Delta(\tau_0)\right) = \langle\rho_m^a\overline{\rho_m}\mid\phi_m\rangle\overline{\tau_x}\to\tau \\ new(\overline{\pi}) \quad |\overline{\pi}| = |\overline{\rho_m}| \quad \psi = [\overline{\pi}/\overline{\rho_m}][\rho^a/\rho_m^a] \quad C_x \vdash \overline{\psi(\tau_x)}\ OK \quad C_r \vdash \psi(\tau)\ OK \\ \Sigma;\Delta;\Gamma;\rho^a;C_e \vdash \bar{e} \hookrightarrow \overline{e'}{:}\overline{\tau_e} \quad \Delta; C_\lhd \vdash \overline{\tau_e} <: \overline{\psi(\tau_x)} \quad C = C_x \wedge C_r \wedge C_e \wedge C_\lhd \wedge \psi(\phi_m)\end{array}}{\Sigma;\Delta;\Gamma;\rho^a;C \vdash e_0.m(\bar{e}) \hookrightarrow e_0'.m\langle\rho^a\overline{\pi}\rangle\left(\overline{e'}\right){:}\psi(\tau)}$$

$$\boxed{\Sigma; \Delta;\ \Gamma; \rho^a; C \vdash s \hookrightarrow s' \dashv \Gamma'}$$

$$\frac{\begin{array}{c}\tau = templateTy(T) \quad C_T \vdash \tau\ OK \quad \Gamma' = \Gamma, x{:}\tau \\ \Sigma;\Delta;\Gamma;\rho^a;C \vdash e \hookrightarrow e'{:}\tau_e \quad \Delta; C_\lhd \vdash \tau_e <: \tau \quad C = C_T \wedge C_e \wedge C_\lhd\end{array}}{\Sigma;\Delta;\Gamma;\rho^a;C \vdash let\ T\ x = e \hookrightarrow let\ \tau\ x = e' \dashv \Gamma'}$$

$$\frac{\begin{array}{c}e_1 \in \{x, e.f\} \quad \Sigma;\Delta;\Gamma;\rho^a;C_1 \vdash e_1 \hookrightarrow e_1'{:}\tau_1 \\ \Sigma;\Delta;\Gamma;\rho^a;C_2 \vdash e_2 \hookrightarrow e_2'{:}\tau_2 \quad \Delta; C_\lhd \vdash \tau_2 <: \tau_1 \\ C = C_1 \wedge C_2 \wedge C_\lhd\end{array}}{\Sigma;\Delta;\Gamma;\rho^a;C \vdash e_1 = e_2 \hookrightarrow e_1' = e_2' \dashv \Gamma}$$

$$\frac{\begin{array}{c}new(\rho) \quad \Sigma\cup\{\rho\};\Delta;\Gamma;\rho;C_s \vdash s \hookrightarrow s' \dashv \Gamma' \\ C = \rho \notin \Sigma \wedge \Sigma \succcurlyeq \rho \wedge C_s\end{array}}{\Sigma;\Delta;\Gamma;\rho^a;C \vdash letregion\ \{\ s\ \} \hookrightarrow letregion\langle\rho\rangle\ \{\ s'\ \} \dashv \Gamma} \qquad \frac{\begin{array}{c}\Sigma;\Delta;\Gamma;\rho^a;C_1 \vdash s_1 \hookrightarrow s_1' \dashv \Gamma_1 \\ \Sigma;\Delta;\Gamma_1;\rho^a;C_2 \vdash s_2 \hookrightarrow s_2' \dashv \Gamma' \quad C = C_1 \wedge C_2\end{array}}{\Sigma;\Delta;\Gamma;\rho^a;C \vdash s_1; s_2 \hookrightarrow s_1'; s_2' \dashv \Gamma'}$$

$$\frac{\begin{array}{c}\Sigma;\Delta;\Gamma;\rho^a;C_R \vdash e \hookrightarrow e'{:}Region[\rho]\langle\pi^a\rangle\langle\tau\rangle \\ \Sigma\cup\{\rho\};\Delta;\Gamma;\rho^a;C_s \vdash s \hookrightarrow s' \dashv \Gamma' \quad C = \rho \notin \Sigma \wedge C_R \wedge C_s\end{array}}{\Sigma;\Delta;\Gamma;\rho^a;C \vdash open\ e\ \{\ s\ \} \hookrightarrow open\ e'\ \{\ s'\ \} \dashv \Gamma'}$$

$$\frac{\begin{array}{c}\Sigma;\Delta;\Gamma;\rho^a;C_R \vdash e \hookrightarrow e'{:}Region[\rho]\langle\pi^a\rangle\langle\tau\rangle \\ \Sigma\cup\{\rho\};\Delta;\Gamma;\rho;C_s \vdash s \hookrightarrow s' \dashv \Gamma' \quad C = \rho \notin \Sigma \wedge C_R \wedge C_s\end{array}}{\Sigma;\Delta;\Gamma;\rho^a;C \vdash open^a\ e\ \{\ s\ \} \hookrightarrow open^a\ e'\ \{\ s'\ \} \dashv \Gamma'}$$

$$\frac{\begin{array}{c}\Sigma;\Delta;\Gamma;\rho^a;C_R \vdash e \hookrightarrow e'{:}Region[\rho]\langle\pi^a\rangle\langle\tau\rangle \quad \rho \in \Sigma \\ \Sigma;\Delta;\Gamma;\rho^a;C_2 \vdash e_2 \hookrightarrow e_2'{:}\tau' \quad C = C_R \wedge C_2 \wedge (\tau' = \tau)\end{array}}{\Sigma;\Delta;\Gamma;\rho^a;C \vdash e.set(e_2) \hookrightarrow e'.set\left(e_2'\right) \dashv \Gamma} \qquad \frac{\begin{array}{c}\Sigma;\Delta;\Gamma;\rho^a;C_R \vdash e \hookrightarrow e'{:}Region[\rho]\langle\pi^a\rangle\langle\tau\rangle \quad \rho \notin \Sigma \\ a \in \{transfer, giveUp\}\end{array}}{\Sigma;\Delta;\Gamma;\rho^a;C \vdash e.a() \hookrightarrow e'.a() \dashv \Gamma}$$