Higher-Order Region Type System (UDFs)

Thursday, July 10, 2014 3:52 PM

FJ with Regions

The Language

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

Notes

1. A note on notation: We write $\bar{a}$ to denote sequence of a's (i.e., $a_0, a_1, \ldots, a_i, \ldots, a_n$). Identifier a (without numberic subscript) is different from any a's in $\bar{a}$. When b is an identifier and $\odot$ denotes a binary relation, we write $b \odot \bar{a}$ to denote either $b \odot a_0 \wedge \cdots \wedge b \odot a_i \wedge \cdots$ or the set $\{b \odot a_0, \ldots, b \odot a_i, \ldots\}$, depending on the context. Similarlly, $\bar{b} \odot \bar{a}$ or $\overline{(b \odot a)}$ denotes point-wise $\odot$ relation between b's and a's.
2. When we say $\alpha \lhd N$, instantiated class N is the bound of the type variable α.
3. A region name (ρ, p or π) is a static identifier for a region created by either a $letregion$ expression or a $new\ Region$ expression. If such an expression is inside a loop or a recursive function, it creates multiple regions at run-time, all of which have same static identifier. Any outlives relation that holds on two static identifiers also holds between corresponding regions at run-time. For eg, consider the following code:

```
let var x = …;
while (.…){
  letregion<R0> {
    letregion<R1> {
      …
    }
  }
}
```

The outlives relation (R0⩾R1) that holds between static identifiers R0 and R1 inside while loop, also holds between run-time regions denoted by by R0 and R1 in every iteration of the while loop. It is possible to create an object in R1 that holds a reference to an object in R0. However, the outlives relation does not hold between regions across iterations. Accordingly, it is not possible to create an object inside R0 or R1 in one iteration, and use its reference in later iterations. The only way this could have been done is by assigning the object reference to a variable (`x`) that is declared above while statement, and dereference `x` in later iterations, but this is disallowed by region type system as variable `x` has longer life time than the objects in R0 or R1. It should be observed that region type system did not allow incorrect outlives relation between run-time regions identified by R0 and R1 across multiple iterations, while still allowing sound outlives relation between regions created in a single iteration.

4. We have region-polymorphic classes and region-polymorphic methods. Examples are given at the end of this wiki (LinkedListNode and LinkedList). The syntax of a region-polymorphic class is given below:
   $\text{class } B\langle \rho^a \bar{\rho}\rangle\langle\overline{\alpha \lhd N}\rangle \lhd N \left\{\overline{\tau f}; k ; \bar{d}\right\}$
   ρ^a denotes the allocation context parameter of class B. $\bar{\rho}$ is the sequence of free region identifiers (also referred to as "region variables") in types ($\bar{\tau}$) of instance variables ($\bar{f}$) of B. To ensure the safety invariant that longer-living objects cannot contain references to shorter-living objects, we require that $\bar{\rho} \succcurlyeq \rho^a$. This constraint is implicit for all class definitions, and is not recorded anywhere. The constraint is checked when region parameters are instantiated with concrete region identifiers whenever a new object is created. For example, assuming $\pi^a$and $\bar{\pi}$ are concrete region identifiers, the expression is judged OK only if the relation $\bar{\pi} \succcurlyeq \pi^a$ holds.
5. The syntax of a region-polymorphic method (m) is given below:
   $\tau\, m\langle\rho^a \bar{\rho} \mid \phi\rangle\ (\overline{\tau\, x})\ \{s;\ \text{return } e;\}$
   As per our convention, ρ^a denotes allocation context of the method, and $\bar{\rho}$ denotes sequence of free region variables in argument and return types ($\bar{\tau}$ & τ). Note that ρ^a can also occur free in $\bar{\tau}$ and τ. Unlike region parameters of a class, we do not have any implicit "outlives" constraints on region parameters of a method. Instead, we explicitly maintain a conjunction of constraints as a refinement predicate (ϕ) over set of region parameters. The predicate ϕ constraints the set of concrete regions that can be used to instantiate the region parameters of the method, and should be viewed as the precondition for the method call to be region-safe.
6. All methods are parameterized over their allocation context; so, all methods are region-polymorphic w.r.t their allocation region. If a method requires its allocation context to be a specific region (eg: same region as the allocation context of its class), then this constraint needs to be stated as an explicit equality constraint in the refinement. For eg, method `refresh` in the following defn of class `Foo` requires its inAllocationContext to be a particular region:

```
class Foo<Ra0,R0> extends Object<Ra0> {
  Object<R0> x;
  unit refresh<Ra1 | Ra1 = R0>(unit u) {
    this.x = new Object<Ra1>();
    return ();
  }
}
```

7. Consider the following recursive method:

```
unit foo<Ra>(Object<Ra> x) {
  letregion<R0> {
    …
    Object<R0> y = new Object<R0>();
    foo<R0>(y);
    …
  }
}
```

The method `foo` is region-polymorphic with its inAllocationContext (`Ra`) as the only region parameter. The outlives relation (Ra≽R0) holds between foo's inAllocationContext (Ra) and newly created static region (R0). We allow region-polymorphic recursion, letting foo to pass region R0 as inAllocationContext to its recursive call. Since R0 is Ra for the recursive call, there exists outlives relation between different run-time regions with same static identifier (R0) across recursive calls. This outlives relation is captured statically via the relation Ra≽R0. It should be observed that region type system allowed sound outlives relation among regions created across recursive calls.

8. Base types, such as int and bool are unboxed and does not have region annotations. A Type variable itself does not have region annotations, but its bound class (the class it extends) tells us what region its values are allocated in .
9. $Region[\rho]\langle p^a\rangle\langle\tau\rangle$ to be treated as $\exists\rho.Region[\rho]\langle p^a\rangle\langle\tau\rangle$, with automatic packing and unpacking when opening and closing the region, respectively. Further, τ should be well-formed under $\Delta = \{\rho\}$.
10. Although we have a variable binding like $r\colon Region[\rho_0]\langle p^a\rangle\langle\tau\rangle$ in the context, the type $Pair\langle\rho_0\rangle\langle\alpha@\rho_0,\beta@\rho_0\rangle \rightarrow \alpha@\rho_0$ is still ill-formed, as $\rho_0 \notin \Delta$. The region name cannot be used to write types, or in region param instantiations until the region is open.
11. It is alright for two transferable regions to have same name (ρ_0 above). Our static semantics prevent both from being open at the same time, as a given ρ_0 cannot be bound twice in Δ (see rules for $open$ and $open^a$).

Auxiliary Definitions

Alloc Region: Allocation region argument of a class

$$allocRgn(A\langle\rho^a\bar{\rho}\rangle) = \rho^a$$

Fields: Type substitutions carried out in types of fields in current class, and types of fields in super class.

$$fields(\mathrm{T}) = \cdot \qquad \frac{\begin{array}{c} CT(B)= class\ B\langle\rho^a\overline{\rho}\rangle\langle\overline{\alpha\lhd N}\rangle \lhd N\{\overline{\tau_B\ f};\ldots\} \\ fields([\bar{p}/\overline{\rho}][\overline{\tau}/\overline{\alpha}]\ N)= \overline{\tau_A\ g} \end{array}}{fields(B\langle p^a\bar{p}\rangle\langle\overline{\tau}\rangle)=\overline{[\bar{p}/\overline{\rho}][p^a/\rho^a]}[\overline{\tau}/\overline{\alpha}]\tau_B\ f, \overline{\tau_A\ g}}$$

ctype: Types of constructor arguments of a class.

$$ctype(\mathrm{T}) = \cdot \qquad \frac{\begin{array}{c} CT(B)= class\ B\langle\rho^a\overline{\rho}\rangle\langle\overline{\alpha\lhd N}\rangle \lhd N\{\ldots ; k_B;\ldots\} \\ k_B{=}B(\overline{\tau\ x})\{\ldots\} \end{array}}{ctype(B\langle p^a\bar{p}\rangle\langle\overline{T}\rangle)=\overline{[\bar{p}/\overline{\rho}][p^a/\rho^a]}[\overline{T}/\overline{\alpha}]\ \tau}$$

Method Type Lookup: Method type now contains the allocation region for the closure, which is same as the allocation region of the class.

$$\frac{\begin{array}{c} CT(B){=}class\ B\langle\rho^a\overline{\rho}\rangle\langle\overline{\alpha\lhd N}\rangle \lhd N\ \ \{\overline{\tau_B\ f};\ k_B;\ d_B\} \\ \tau^2\ mn\langle\rho^a_m\overline{\rho_m}\mid\phi\rangle\ (\overline{\tau^1\ x})\ \{return\ e;\}\in d_B \end{array}}{mtype(mn,B\langle p^a\bar{p}\rangle\langle\overline{T}\rangle) = [\bar{p}/\overline{\rho}][p^a/\rho^a][\overline{T}/\overline{\alpha}]\ \langle\rho^a_m\overline{\rho_m}\mid\phi\rangle\ \overline{\tau^1}\xrightarrow{\rho^a}\tau^2}$$

$$\frac{\begin{array}{c} CT(B){=}class\ B\langle\rho^a\overline{\rho}\rangle\langle\overline{\alpha\lhd N}\rangle \lhd N\ \{\overline{\tau_B\ f};\ k_B;\ d_B\} \\ mn\notin FMN(d_B) \end{array}}{mtype(mn,B\langle p^a\bar{p}\rangle\langle\overline{T}\rangle) =\ mtype(mn,[\bar{p}/\overline{\rho}][p^a/\rho^a][\overline{T}/\overline{\alpha}]N)}$$

Method Def Lookup: We need the definition of mn in class B.

$$\frac{\begin{array}{c} CT(B){=}class\ B\langle\rho^a\overline{\rho}\rangle\langle\overline{\alpha\lhd N}\rangle \lhd N\ \{\overline{\tau_B\ f};\ k_B;\ d_B\} \\ \tau^2\ mn\langle\rho^a_m\overline{\rho_m}\mid\phi\rangle\ (\overline{\tau^1\ x})\ \{return\ e;\}\in d_B \end{array}}{mdef(mn,B\langle p^a\bar{p}\rangle\langle\overline{T}\rangle) = (\{\rho^a_m\overline{\rho_m}\mid\phi\},\overline{x},\ [\bar{p}/\overline{\rho}][p^a/p]\ [\overline{T}/\overline{\alpha}]e)}$$

$$\frac{\begin{array}{c} CT(B){=}class\ B\langle\rho^a\overline{\rho}\rangle\langle\overline{\alpha\lhd N}\rangle \lhd N\ \{\overline{\tau_B\ f};\ k_B;\ d_B\} \\ mn\notin FMN(d_B) \end{array}}{mdef(mn,B\langle p^a\bar{p}\rangle\langle\overline{T}\rangle) =\ mdef(mn,[\bar{p}/\overline{\rho}][p^a/\rho^a][\overline{T}/\overline{\alpha}]\ N)}$$

Method Override: If a method overrides a method in super class, does it have same type or not? We define equality of types modulo alpha renaming of bound region variables (allocation context variables need to be renamed independent of other region variables). Observe the implication in premise, which indicates that if the method with same name does not exist in the super class, then overriding is trivially sound.

$$\dfrac{\begin{array}{c} CT(B)=class\ B\langle\rho^a\overline{\rho}\rangle\langle\overline{\alpha\lhd N}\rangle \lhd N\ \{\overline{\tau_B\ f};\ k_B;\ d_B\} \\ mtype(m, [\bar{p}/\overline{\rho}][p^a/\rho^a][\overline{T}/\overline{\alpha}\,]\ N)= \langle\pi_1^a\overline{\pi_1}\,|\ \phi_1\rangle\ \overline{\tau^{11}}\xrightarrow{\rho_{c1}^a}\tau^{12} \Rightarrow \\ \langle\pi_1^a\overline{\pi_1}\,|\ \phi_1\rangle\ \overline{\tau^{11}}\xrightarrow{\rho_{c1}^a}\tau^{12} \equiv_\alpha \langle\pi_2^a\overline{\pi_2}\ |\ \phi_2\rangle\ \overline{\tau^{21}}\xrightarrow{\rho_{c2}^a}\tau^{22} \end{array}}{override\Big(m, B\langle p^a\bar{p}\rangle\langle\overline{T}\rangle,\langle\pi_2^a\overline{\pi_2}\ |\ \phi_2\rangle\overline{\tau^{21}}\xrightarrow{\rho_{c2}^a}\tau^{22}\Big)}$$

Notes on Typing Rules

- We define an environment Γ to map variables to types, and environment Δ to map type variables to their bounds. We abuse Δ and use it as a set of currently live region names. For clarity, Δ can be considered as a pair of contexts Δ_α and Δ_ρ, such that
  - Δ_α maps type variables to their bounds, and
  - Δ_ρ is a set of live region names.
- Subtyping is reflexive transitive closure of subclass relationship. Subclass relation is defined by the user over region-annotated classes. The only restriction we impose is that a class and its super class must have same allocation parameter (ρ^a). For a type variable, subclass relation is between the variable and its bound, as declared in the class definition.
- As usual (with Featherweight Java based systems), subtyping, typing and well-formedness judgments are parameterized over the class table (*CT*).
- 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.
  $bound_\Delta(\alpha) = \Delta(\alpha)$
  $bound_\Delta(N) = N$
  $bound_\Delta(T) = T$
- We need well-formedness judgment to check:
  - All type variables are in scope
  - All region names are live
  - Type instantiation satisfies stated subclass constraints.
  - Region parameter instantiation satisfies stated outlives constraints.
- Closures are typed under an environment that does not contain bindings with existential types. For an environment Γ, we define $\dot{\Gamma}$ as Γ without existential bindings.

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

$$\dfrac{\square}{\Sigma;\Delta;\gamma\vdash\tau<:\tau} \qquad \dfrac{\square}{\Sigma;\Delta;\gamma\vdash\alpha <:\Delta(\alpha)} \qquad \dfrac{\begin{array}{c} CT(B)=class\ B\langle\rho^a\overline{\rho}\ |\ \phi\rangle\langle\overline{\alpha\lhd N}\rangle \lhd N\,\{\ ...\ \} \\ \Sigma;\Delta;\gamma\vdash B\langle p^a\,\bar{p}\rangle\langle\bar{\tau}\rangle\ OK \quad \Sigma;\Delta;\gamma\vdash[\bar{p}/\overline{\rho}][p^a/\rho^a][\bar{\tau}/\overline{\alpha}\,]\ N\ OK \end{array}}{\Sigma;\Delta;\gamma\vdash B\langle p^a\bar{p}\rangle\langle\bar{\tau}\rangle <:\ [\bar{p}/\overline{\rho}][p^a/\rho^a][\bar{\tau}/\overline{\alpha}\,]\ N}$$

$$\dfrac{\Sigma;\Delta;\gamma\vdash\tau_1<:\tau_2 \quad \Sigma;\Delta;\gamma\vdash\tau_2<:\tau_3}{\Sigma;\Delta;\gamma\vdash\tau_1<:\tau_3} \qquad \dfrac{\rho'\notin frv(\tau)}{\Sigma;\Delta;\gamma\vdash\exists\rho.\tau<:\exists\rho'.[\rho'/\rho]\tau}$$

$$\dfrac{\begin{array}{c} \exists f:\overline{\pi}\rightarrow\{\overline{\rho}\cup\Delta_\rho\} \quad \Psi=[\rho^a/\pi^a][f(\overline{\pi})/\overline{\pi}] \quad \gamma\vdash\pi_c^a\succcurlyeq\rho_c^a \\ \tau^{21}<:\Psi\big(\overline{\tau^{11}}\big) \quad \gamma\vdash\phi_2\Rightarrow\Psi(\phi_1) \quad \Psi(\tau^{12})<:\tau^{22} \end{array}}{\Sigma;\Delta;\gamma\vdash\langle\pi^a\overline{\pi}\ |\ \phi_1\rangle\overline{\tau^{11}}\xrightarrow{\pi_c^a}\tau^{12} <: \langle\rho^a\overline{\rho}\ |\ \phi_2\rangle\overline{\tau^{21}}\xrightarrow{\rho_c^a}\tau^{22}}$$

Note: in the function subtyping rule, if we require the substitution map (f) to be a bijection of type f: $\overline{\pi} \rightarrow \overline{\rho}$ and , and then replace $\Rightarrow$ with $\Leftrightarrow$, and $<:$ with $\equiv_\alpha$, we get the rule for alpha-equivalence of two function types.

$\boxed{\Sigma;\Delta;\ \gamma \vdash \tau\ OK \quad \Delta \vdash \phi\ OK}$

$$\frac{p^{a} \in \Delta}{\Sigma;\, \Delta;\gamma \vdash \mathrm{Object}\langle p^{a}\rangle\ OK} \qquad \frac{\alpha \in dom(\Delta)}{\Sigma;\, \Delta;\gamma \vdash \alpha\ OK} \qquad \frac{\rho_0, \rho_1 \in \Delta}{\Delta \vdash \rho_0 \succcurlyeq \rho_1\ OK} \qquad \frac{\Delta \vdash \phi_0 OK \quad \Delta \vdash \phi_1 OK}{\Delta \vdash \phi_0 \wedge \phi_1\ OK}$$

$$\frac{\begin{array}{c} CT(B) = class\ B\langle \rho^{a}\overline{\rho}\ |\phi\rangle\langle\overline{\alpha \lhd N}\rangle \lhd N\,\{\,\ldots\,\} \\ p^{a}, \overline{p} \in \Delta \quad \gamma \vdash \overline{p} \succcurlyeq p^{a} \quad S = [\overline{p}/\overline{\rho}][p^{a}/\rho^{a}][\overline{\tau}/\overline{\alpha}\,] \quad \gamma \vdash S(\phi) \\ \Sigma;\Delta;\gamma \vdash \overline{\tau}\ OK \quad \Sigma;\Delta;\gamma \vdash S(\overline{N})\ \mathrm{OK} \quad \Sigma;\Delta;\gamma \vdash \overline{\tau} <: S(\overline{N}) \end{array}}{\Sigma;\, \Delta;\gamma \vdash B\langle p^{a}\overline{p}\rangle\langle\overline{\tau}\rangle\ OK}$$

$$\frac{p^{a} \in \Delta \quad \rho \in \Sigma \quad \Sigma;\{\rho\};\emptyset \vdash \tau\ OK}{\Sigma;\Delta;\gamma \vdash Region[\rho]\langle p^{a}\rangle\langle\tau\rangle\ OK} \qquad \frac{\rho \notin \Sigma \quad \Sigma \cup \{\rho\};\, \Delta;\, \gamma \vdash Region[\rho]\langle p^{a}\rangle\langle\tau\rangle\ \ OK}{\Sigma;\Delta;\gamma \vdash \exists\rho.Region[\rho]\langle p^{a}\rangle\langle\tau\rangle\ \ OK}$$

$$\frac{\begin{array}{c} \rho_c^{a} \in \Delta \quad \Delta' = \Delta \cup \{\rho^{a}, \overline{\rho}\} \quad \Delta' \vdash \phi\ OK \\ \gamma' = \gamma \wedge \phi \quad \Sigma;\, \Delta';\gamma' \vdash \overline{\tau^{1}}\ OK \quad \Sigma;\, \Delta';\gamma' \vdash \tau^{2}\ OK \end{array}}{\Sigma;\, \Delta;\gamma \vdash \langle \rho^{a}\overline{\rho}\ |\ \phi\ \rangle \overline{\tau^{1}} \xrightarrow{\rho_c^{a}} \tau^{2}\ OK}$$

$$\boxed{\Sigma;\Delta;\gamma;p^{a};\Gamma \vdash e:\tau}$$

$$\frac{x:\tau \in \Gamma}{\Sigma;\Delta;\gamma;p^{a};\Gamma \vdash x:\tau} \qquad \frac{\begin{array}{c} \Sigma;\, \Delta;\, \gamma;\, p^{a};\, \Gamma \vdash e:\tau' \\ k:\tau \in fields\big(bound_{\Delta}(\tau')\big) \end{array}}{\Sigma;\, \Delta;\, \gamma;\, p^{a};\, \Gamma \vdash e.k:\tau} \qquad \frac{\Sigma;\Delta;\gamma \vdash \tau_{\lhd}\ OK}{\Sigma;\Delta;\gamma;p^{a};\Gamma \vdash Null : \tau_{\lhd}}$$

$$\frac{\begin{array}{c} \Sigma;\Delta;\gamma \vdash N\ OK \quad allocRgn(N) = p \quad \gamma \vdash p^{a} \succcurlyeq p \\ ctype(N) = \overline{\tau} \quad \Sigma;\, \Delta;\, \gamma;\, p^{a};\, \Gamma \vdash \bar{e}:\overline{\tau_e} \quad \Sigma;\Delta;\gamma \vdash \overline{\tau_e} <: \overline{\tau} \end{array}}{\Sigma;\, \Delta;\, \gamma;\, p^{a};\, \Gamma \vdash new\ N(\bar{e}) : N}$$

$$\frac{\begin{array}{c} \Sigma;\Delta;\gamma;p^{a};\Gamma \vdash e:\tau \\ mtype\big(k, bound_{\Delta}(\tau)\big) = \langle \rho^{a}\overline{\rho}\ |\ \phi\rangle\ \overline{\tau^{1}} \xrightarrow{\rho_c^{a}} \tau^{2} \end{array}}{\Sigma;\Delta;\gamma;p^{a};\Gamma \vdash e.k:\langle \rho^{a}\overline{\rho}\ |\ \phi\rangle\ \overline{\tau^{1}} \xrightarrow{\rho_c^{a}} \tau^{2}} \qquad \frac{\Delta;\gamma \vdash \tau_{\lhd}\ OK}{\Delta;\gamma;p^{a};\Gamma \vdash Null : \tau_{\lhd}}$$

$$\frac{\begin{array}{c} \Delta_0 \subseteq \Delta \quad \gamma \vdash \Delta_0 \succcurlyeq p^{a} \quad \Delta' = \Delta_0 \cup \rho^{a} \cup \overline{\rho} \quad \Sigma' = \Delta' \quad \Delta_0 \vdash \gamma_0\ OK \\ \Delta \vdash \gamma \Rightarrow \gamma_0 \quad \gamma' = \gamma_0 \wedge \phi \quad \Sigma';\Delta';\gamma' \vdash \overline{\tau^{1}}\ OK \quad \Sigma';\Delta';\gamma' \vdash \tau^{2}\ OK \\ \Sigma';\Delta';\gamma';\, \rho^{a};\, \dot{\Gamma}, \overline{x:\tau^{1}} \vdash s \Rightarrow \Gamma';\Sigma'' \quad \Sigma'';\, \Delta';\gamma';\, \rho^{a};\, \Gamma' \vdash e:\tau \quad \Sigma'';\Delta';\gamma' \vdash \tau <: \tau^{2} \end{array}}{\Sigma;\Delta;\gamma;p^{a};\Gamma \vdash \lambda\langle \rho^{a}\overline{\rho}\ |\ \phi\rangle\big(\overline{\tau^{1}\ x}\big).\{s; return\ e;\}:\langle \rho^{a}\overline{\rho}\ |\ \phi\rangle\ \overline{\tau^{1}} \xrightarrow{p^{a}} \tau^{2}}$$

$$\frac{\begin{array}{c} \Sigma;\Delta;\gamma;p^{a};\Gamma \vdash e_0 : \langle \rho^{a}\overline{\rho}\ |\ \phi\rangle\ \overline{\tau^{1}} \xrightarrow{\rho_c^{a}} \tau^{2} \quad \overline{p} \in \Delta \\ |\overline{p}| = |\overline{\rho}| \quad S = [\overline{p}/\overline{\rho}][p^{a}/\rho^{a}] \quad \Sigma;\Delta;\gamma \vdash S(\overline{\tau^{1}})\ OK \\ \Sigma;\Delta;\gamma \vdash S(\tau^{2})\ OK \quad \Sigma;\Delta;\gamma;p^{a};\Gamma \vdash \overline{e : \tau_e} \quad \Sigma;\Delta;\, \gamma \vdash \overline{\tau_e} <: S(\overline{\tau^{1}}) \quad \gamma \vdash S(\phi) \end{array}}{\Delta;\gamma;p^{a};\Gamma \vdash e_0\langle p^{a}\overline{p}\rangle(\bar{e}) : S(\tau^{2})}$$

$$\frac{\rho \in \Delta \quad \Sigma;\Delta;\gamma;\rho^{a};\Gamma \vdash e:Region[\rho]\langle \rho_k^{a}\rangle\langle\tau\rangle}{\Sigma;\Delta;\gamma;\rho^{a};\Gamma \vdash e.get():\tau} \qquad \frac{\Sigma;\Delta;\gamma;p^{a};\Gamma \vdash e:[\rho_0/\rho]\tau \quad \Sigma;\Delta;\gamma \vdash \exists\rho.\tau\ OK}{\Sigma;\Delta;\gamma;p^{a};\Gamma \vdash pack[\rho_0, e]\ as\ \exists\rho.\tau : \exists\rho.\tau}$$

$$\frac{\Sigma;\Delta;\gamma\vdash\exists\rho.Region[\rho]\langle \mathrm{p^a}\rangle\langle\tau\rangle\ OK}{\Sigma;\Delta;\gamma;\mathrm{p^a};\Gamma\vdash newRgn\langle\rho\rangle\langle\tau\rangle():\exists\rho.Region[\rho]\langle \mathrm{p^a}\rangle\langle\tau\rangle}$$

$$\boxed{\Sigma;\ \Delta;\ \gamma;\ \rho^{a};\ \Gamma\vdash s\Rightarrow\Gamma';\Sigma'}$$

$$\frac{\begin{array}{c}\Sigma;\Delta;\gamma\vdash\tau\ OK\\ \Sigma;\Delta;\gamma;\rho^{a};\Gamma\vdash e{:}\tau_1\qquad\Sigma;\Delta;\gamma\vdash\tau_1<:\tau\end{array}}{\Sigma;\Delta;\gamma;\rho^{a};\Gamma\vdash let\ \tau\ x{=}e\Rightarrow\Gamma,x{:}\tau;\ \Sigma}
\qquad
\frac{\begin{array}{c}e_1\in\{x,e.f\}\quad\Sigma;\Delta;\gamma;\rho^{a};\Gamma\vdash e_1{:}\tau_1\\ \Sigma;\Delta;\gamma;\rho^{a};\Gamma\vdash e_2{:}\tau_2\qquad\Sigma;\Delta;\gamma\vdash\tau_2<:\tau_1\end{array}}{\Sigma;\Delta;\gamma;\rho^{a};\Gamma\vdash e_1=e_2\Rightarrow\Gamma;\Sigma}$$

$$\frac{\rho\notin\Sigma\quad\Sigma\cup\rho;\ \Delta\cup\rho;\ \gamma\cup\{\Delta\succcurlyeq\rho\},\ \rho,\ \Gamma\vdash s\Rightarrow\Gamma';\Sigma'}{\Sigma;\Delta;\gamma;\rho^{a};\Gamma\vdash letregion\langle\rho\rangle\{\ s\ \}\Rightarrow\Gamma;\Sigma}
\qquad
\frac{\begin{array}{c}\Sigma;\ \Delta;\ \gamma;\ \rho^{a};\ \Gamma\vdash s_1\Rightarrow\Gamma';\Sigma'\\ \Sigma';\ \Delta;\ \gamma;\ \rho^{a};\ \Gamma'\vdash s_2\Rightarrow\Gamma'';\Sigma''\end{array}}{\Sigma;\ \Delta;\ \gamma;\ \rho^{a};\ \Gamma\vdash s_1;s_2\Rightarrow\Gamma'';\Sigma''}$$

$$\frac{\begin{array}{c}\Sigma;\Delta;\gamma;\rho^{a};\Gamma\vdash e{:}Region[\rho]\langle\rho^{a}_{k}\rangle\langle\tau\rangle\\ \rho\notin\Delta\quad\Sigma;\ \Delta\cup\rho;\ \gamma;\ \rho^{a};\ \Gamma\vdash s\Rightarrow\Gamma';\Sigma'\end{array}}{\Sigma;\Delta;\gamma;\rho^{a};\Gamma\vdash open\ e\ \{\ s\ \}\Rightarrow\Gamma;\Sigma}
\qquad
\frac{\begin{array}{c}\Sigma;\Delta;\gamma;\rho^{a};\Gamma\vdash e{:}Region[\rho]\langle\rho^{a}_{k}\rangle\langle\tau\rangle\\ \rho\notin\Delta\quad\Sigma;\Delta\cup\rho;\ \gamma;\ \rho;\ \Gamma\vdash s\Rightarrow\Gamma';\Sigma'\end{array}}{\Sigma;\Delta;\gamma;\rho^{a};\Gamma\vdash open^{a}\ e\ \{\ s\ \}\Rightarrow\Gamma;\Sigma}$$

$$\frac{\begin{array}{c}\Sigma;\Delta;\gamma;\rho^{a};\Gamma\vdash e{:}Region[\rho]\langle\rho^{a}_{k}\rangle\langle\tau\rangle\\ \Sigma;\Delta;\gamma;\rho^{a};\Gamma\vdash e_1{:}\tau\end{array}}{\Sigma;\Delta;\gamma;\rho^{a};\Gamma\vdash e.set(e_1)\Rightarrow\Gamma;\Sigma}
\qquad
\frac{\begin{array}{c}a\in\{transfer,giveUp\}\\ \Sigma;\Delta;\gamma;\rho^{a};\Gamma\vdash e{:}Region[\rho]\langle\rho^{a}_{k}\rangle\langle\tau\rangle\end{array}}{\Sigma;\Delta;\gamma;\rho^{a};\Gamma\vdash e.a()\Rightarrow\Gamma;\Sigma}$$

$$\frac{\Sigma;\ \Delta;\ \gamma;\ \rho^{a};\ \Gamma\vdash e{:}\exists\rho.\tau\quad\rho_0\notin\Sigma\quad\tau'=[\rho_0/\rho]\tau}{\Delta;\ \gamma;\ \rho^{a};\ \Gamma\vdash let\ (\rho_0,\tau'\ x)=unpack\ e\Rightarrow\Gamma,x{:}\tau';\ \Sigma\cup\rho_0}$$

$$\boxed{d\ OK\ in\ B}$$

$$\frac{\begin{array}{c}\Sigma=\rho^{a}\cup\overline{\rho}\cup\rho^{a}_{m}\cup\overline{\rho_m}\quad\Delta=(\overline{\alpha\lhd N};\Sigma)\quad\Delta\vdash\phi\ OK\\ \gamma=\phi\cup(\overline{\rho}\succcurlyeq\rho^{a})\cup\phi_m\quad\Sigma;\Delta;\gamma\vdash\overline{\tau^1}\ OK\quad\Sigma;\Delta;\gamma\vdash\tau^2\ OK\\ CT(B)=class\ B\langle\rho^{a}\overline{\rho}\mid\phi\rangle\langle\overline{\alpha\lhd N}\rangle\lhd N\{\ \dots\ \}\quad\Gamma=\cdot,\overline{x{:}\tau^1},\text{this}{:}\ B\langle\rho^{a}\overline{\rho}\rangle\langle\overline{\alpha}\rangle\\ override\left(m,N,\langle\rho^{a}_{m}\overline{\rho_m}\mid\phi_m\rangle\overline{\tau^1}\overset{\rho^{a}}{\dashrightarrow}\tau^2\right)\quad\Sigma;\Delta;\ \gamma;\ \rho^{a}_{m};\ \Gamma\vdash s\Rightarrow\Gamma';\Sigma'\quad\Sigma';\Delta;\gamma;\ \rho^{a}_{m};\ \Gamma'\vdash e{:}\tau^2\end{array}}{\tau^2\ mn\langle\rho^{a}_{m}\overline{\rho_m}\mid\phi_m\rangle\ (\overline{\tau^1\ x})\ \{s;\ return\ e;\}\ OK\ in\ B}$$

$$\boxed{B\ OK}$$

$$\frac{\begin{array}{c}\Delta=(\overline{\alpha\lhd N};\rho^{a}\cup\overline{\rho})\quad\Sigma=\rho^{a}\cup\overline{\rho}\quad\Delta\vdash\phi\ \mathrm{OK}\quad\gamma=\overline{\rho}\succcurlyeq\rho^{a};\phi\\ \Sigma;\Delta;\gamma\vdash\overline{N}\ OK\quad allocRgn(N)=\rho^{a}\quad\Sigma;\Delta;\gamma\vdash N\ OK\quad\Sigma;\Delta;\gamma\vdash\overline{\tau^B}OK\\ \overline{d}\ OK\ in\ B\quad ctype\ (N)=\overline{\tau^A}\quad\Gamma=\cdot,\ \text{this}{:}\ B\langle\rho^{a}\overline{\rho}\rangle\langle\overline{\alpha}\rangle,\ \overline{x{:}\tau}\\ k=B(\overline{\tau\ x})\{super(\overline{v_g});\ \overline{this.f=v_f;}\}\quad\Sigma;\ \Delta;\gamma;\ \rho^{a};\ \Gamma\vdash\overline{v_g{:}\tau^A}\quad\Sigma;\Delta;\ \gamma;\ \rho^{a};\ \Gamma\vdash\overline{this.f=v_f}{:}\ unit\end{array}}{class\ B\langle\rho^{a}\overline{\rho}\mid\phi\rangle\langle\overline{\alpha\lhd N}\rangle\lhd N\left\{\ \overline{\tau^B\ f};k\ ;\overline{d}\ \right\}\ OK}$$

Examples

Example 1 (Pair) Here is an implementation of generic pair class:

```
class A extends Object {
    A() { super(); }
}
class B extends Object {
    B() { super(); }
}
class Pair<X extends Object,Y extends Object> extends Object {
    X fst;
    Y snd;
    Pair(X fst, Y snd) {
        super(); this.fst=fst; this.snd=snd;
    }
    X getfst() {
        return this.fst;
    }
    unit setfst(X newfst) {
        this.fst = newfst;
        return ();
    }
}
```

And here is its region-annotated version:

```
$class\ A\langle R^a \rangle\ extends\ Object$ {
  $A()$ { $super()$; }
}
$class\ B\langle R^a \rangle\ extends\ Object$ {
  $B()$ { $super()$; }
}
$class\ Pair\langle R^a, R0, R1\rangle\big\langle X\ extends\ Object\langle R0\rangle, Y\ extends\ Object\langle R1\rangle\big\rangle\ extends\ Object\langle R^a\rangle$ {
  $X\ fst$;
  $Y\ snd$;
  $Pair(X\ fst, Y\ snd)$ {
    $super()$; $this.fst = fst$; $this.snd = snd$;
  }
  $X\ getfst\langle R_m^a\rangle()$ {
    $return\ this.fst$;
  }
  $unit\ setfst\langle R_m^a\rangle(X\ newfst)$ {
    $this.fst = newfst$;
    $return\ ()$;
  }
}
```

Example 2 (Linked List) Here is an implementation of linked list class:

```
class LinkedListNode<T extends Object>  extends Object  {
    T val;
    LinkedListNode<T>  prev;
    LinkedListNode<T>  next;
    LinkedListNode(T val) {
        super();
        this.val = val;
        this.prev = Null;
        this.head = Null;
    }
}
class LinkedList<T extends Object>  extends Object  {
    LinkedListNode<T>  head;
    int count;
    LinkedList(T v) {
        super();
        this.head = new LinkedListNode<T>(v);
        this.count=1;
    }
    unit add(T v) {
        let LinkedListNode<T>  n = new LinkedListNode<T>(v);
        n.next = this.head;
        this.head.prev = n;
        this.head = n;
        this.count=this.count+1;
        return ();
    }
    T head(unit u) {
        return this.head.val;
    }
    LinkedList<T>   reverse(unit u) {
        let LinkedList<T> xs = new LinkedList<T>  (this.head.val);
        let LinkedListNode<T> cur = this.head.next;
        while(not (cur== Null)) {
              xs.add(cur.val);
              cur = cur.next;
           }
        return xs;
    }
}
```

Its elaborated (region-annotated) version is given below.

```
class LinkedListNode⟨$R^a$, R0⟩⟨T extends Object⟨R0⟩⟩ extends Object⟨$R^a$⟩ {
  T val;
  LinkedListNode⟨$R^a$,R0⟩⟨T⟩ prev;
  LinkedListNode⟨$R^a$,R0⟩⟨T⟩ next;
  LinkedListNode(T val) {
    super();
    this.val = val;
    this.prev = Null;
    this.head = Null;
  }
}
class LinkedList⟨$R^a$, R1⟩⟨T extends Object⟨R1⟩⟩ extends Object⟨$R^a$⟩ {
  LinkedListNode⟨$R^a$,R1⟩⟨T⟩ head;
  int count;
  LinkedList(T v) {
    super();
    this.head = new LinkedListNode⟨$R^a$, R1⟩⟨T⟩(v);
    this.count = 1;
  }
  unit add⟨$R^a_m$ | $R^a_m \succcurlyeq R^a$⟩(T v) {
    let LinkedListNode⟨$R^a$, R1⟩⟨T⟩ n = new LinkedListNode⟨$R^a$, R1⟩⟨T⟩(v);
    n.next = this.head;
    this.head.prev = n;
    this.head = n;
    this.count = this.count + 1;
    $return\ ();$
  $\}$
  $T\ head\langle R_m^a\rangle(unit\ u)\ \{$
    $return\ this.head.val;$
  $\}$
  $LinkedList\langle R_m^a, R1\rangle\langle T\rangle\ reverse\langle R_m^a|\ R1 \succcurlyeq R_m^a\rangle(unit\ u)\ \{$
    $let\ LinkedList\langle R_m^a, R1\rangle\langle T\rangle\ xs\ =\ new\ LinkedList\langle R_m^a, R1\rangle\langle T\rangle\ (this.head.val);$
    $let\ LinkedListNode\langle R^a, R1\rangle\langle T\rangle\ cur\ =\ this.head.next;$
      $while(not\ (cur == \ Null))\ \{$
        $xs.add(cur.val);$
        $cur\ =\ cur.next;$
      $\}$
    $return\ xs;$
  $\}$
$\}$
```

Example 3 (An Actor): Here is an implementation of an actor that accepts a region with list of objects, and transfers a new region with a single object, which is the last object of the list in input region:

```
class AnActor extends Object {
    AnActor () {
        super();
    }
    unit onRecv(Region[Rin]<LinkedList<Object>> rin) {
        let Region[Rout]<Object> rout = new Region[Rout]<Object>;
        open rin {
            let LinkedList<Object> xs = rin.get();
            letregion<Rs0> {
                let LinkedList<Object> sx = xs.reverse();
                let Object v = sx.head;
                openAlloc rout {
                    rout.suck(v);
                }
            }
        }
        rin.giveUp();
        rout.transfer();
        return ();
    }
}
```

Its region-annotated version is given below. Reader has to be convinced that elaboration adheres to the rules given previously.

```
$class\ AnActor\langle R^a\rangle\ extends\ Object\langle R^a\rangle\ \{$
  $AnActor\ ()\ \{$
    $super();$
  $\}$
  $unit\ onRecv\langle R_m^a, R_0^a\rangle(Region[Rin]\langle R_0^a\rangle\left\langle LinkedList\langle Rin, Rin\rangle\langle Object\langle Rin\rangle\rangle\right\rangle\ \ rin)\ \{$
    $let\ Region[Rout]\langle R_m^a\rangle\langle Object\langle Rout\rangle\rangle\ rout\ =\ new\ Region[Rout]\langle Object\langle Rout\rangle\rangle;$
    $open\ rin\ \{$
      $let\ LinkedList\langle Rin, Rin\rangle\langle Object\langle Rin\rangle\rangle\ xs\ =\ rin.get();$
      $letregion\langle Rs0\rangle\ \{$
        // $Rs0$ is the new inAllocationContext
        // below call to reverse type checks because Rin ≽ Rs0
        let LinkedList⟨Rs0, Rin⟩⟨Object⟨Rin⟩⟩ sx  =  xs.reverse⟨Rs0⟩();
        let Object⟨Rin⟩ v  =  sx.head⟨Rs0⟩();
        open^a rout {
          rout.suck(v);
        }
      }
    }
    rin.giveUp();
    rout.transfer();
    return ();
  }
}
```

-

- Immediately needed extensions:
  - Ability to state outlives constraints explicitly over region params of class (this requirement goes hand -in-hand with above two)
  - Interfaces
  - Way to call super class methods, when they are overridden in subclass
  - Iteration.
  - Polymorphic methods
- A fallout of only allowing $\bar{\rho} \succcurlyeq \rho^a$, but not allowing to state outlives constraints within $\bar{\rho}$ is that an object contained in class that expects two or more region params cannot be allocated in a different region. Such an instance variable cannot even be declared in current setting.