cd ::= q_C class C extends D { $\overline{\text{fd}}$; kd $\overline{\text{md}}$ } class
fd ::= q a C f field
kd ::= q C (t C g, t C f) { super(g); this.f = f; } constructor
md ::= t C m (t C this, t C x) { $\overline{\text{t C y}}$ $\overline{\text{s}}$; return z } instance method
e ::= x | x.f expression
s ::= x = e | x.f = y | x = y.m(z) | super(g) | this(g) | x = new C() | s;s statement
t ::= k q qualifier type
k ::= initialized | underinitialized | unknowninitialized | fbcbottom
initializatioin qualifier
q ::= readonly | mutable | polymutable | substitutablepolymutable | receiverdependantmutable | immutable | bottom
immutability qualifier

a :: = assignable | receiverdependantassignable | final assignability qualifier

Qualifier Hierarchy

```
             unknowninitialized
               /            \
      underinitialized    initialized
                           (default)

               \            /
                 fbcbottom

                              readonly
      /      /                   |                    \                    \
mutable  polymutable  substitutablepolymutable  receiverdependantmutable immutable
  (default)            (polymorphic qual in CF)
      \      \                   |                    /                    /
                                bottom


  assignable    receiverdependantassignable    final
```

Figure 1 Combination of qualifiers. First two qualifier hierarchies are orthogonal. If an object is under initialization, its immutability guarantee is not satisfied. So even immutable and receiverdependantmutable objects can also be modifed when under initialization. Third one is only used on field declarations, and not included in atms.

Subtype relations

$k_1\ q_1 <: k_2\ q_2 <=> k_1 <: k_2 \wedge q_1 <: q_2$

Helper Functions

$$\frac{q\ a\ C\ f}{fType(f) = q\ a}$$

***Note**:*

1) No initialization modifier on field declarations. In actual implementation, to have circular initialization, *@NotOnlyInitialized* can be used on field declaration. However, it doesn't belong to initialization qualifier hierarchy.

2) The field is unique within the whole type hierarchy

fields(C) returns all fields directly declared in C.

cBody(kd) returns constructor body of kd.

mBody(md) returns method body of md.

Viewpoint Adaptation Rules

_ ▹ mutable = mutable

_ ▹ readonly = readonly

_ ▹ immutable = immutable

_ ▹ bottom = bottom

_ ▹ polymutable = substitutablepolymutable

q ▹ receiverdependantmutable = q

Note: substitutablepolymutable only exists shortly after viewpoint adaptation is done, but will must be subsituted by another qualifier immediately by QualifierPolymorphism. So, substitutablepolymutable should not appear on left or right side of viewpoint adaptation triangle.

Special Rules

- Forbid polymutable fields; readonly or polymutable constructor return type and readonly instantiation of objects
- Forbid assignability qualifier – receiverdependantassignable on locations other than instance fields.
- Forbid initialization modifier on fields, constructor return type and new statement
- Forbid bottom except on (implicit/explicit) lower bounds and null literal.
- Forbid explicit use of substitutablepolymutable everywhere.

TODO: Should we allow polymutable constructor return type?

Typing Rules

$$\frac{x \in \Gamma}{\Gamma \vdash x : \Gamma(x)} \quad \text{(T-VAR)}$$

$$\frac{\Gamma(x) = k_x\, q_x \quad fType(f) = q_f\ \_ \quad q = q_x \rhd q_f \qquad k = \begin{cases} \text{initialized} & \textit{if } k_x = \textit{initialized} \\ \text{unknowninitialized} & \textit{otherwise} \end{cases}}{\Gamma \vdash x.f : k\, q} \quad \text{(T-FLD)}$$

Figure 2 Expression typing

$$\frac{\Gamma \vdash e = t_e \quad t_e <: \Gamma(x)}{\Gamma \vdash x = e} \quad \text{(T-VARASS)}$$

$\Gamma(x) = k_x\, q_x \quad \Gamma(y) = k_y\, q_y \quad \mathrm{typeof}(f) = q_f\, a_f$
$q_x = \mathrm{mutable}\ \lor$
$(k_x = \mathrm{underinitialized}\ \land\ q_x = \mathrm{immutable})\ \lor$
$(k_x = \mathrm{underinitialized}\ \land\ q_x = \mathrm{receiverdependantmutable})\ \lor$
$(a_f = \mathrm{assignable}\ \land\ (q_x \neq \mathit{readonly}\ \lor\ q_f \neq \mathit{receiverdependantmutable})\ )$
$q_y <: q_x \rhd q_f$
$k_x = \mathrm{underinitialized}\ \lor\ k_y = \mathrm{initialized}$

(T-FLDASS)

$$\Gamma \vdash x.f = y$$

* **Note :**

- Every assignment to instance fields without explicit receiver has implicit receiver *this*. In constructor, $q_{this} = q_{ret;}$ In initialization blocks, $q_{this} = q_{C;}$ In instance field declarations(with initializers), $q_{this} = q_C$.
- PICO only handles assignable and receiverdependantassignable fields cases. Final fields are enforced by Java compiler and doesn't need PICO to do anything.

$\Gamma(x) = k_x\, q_x \quad \Gamma(y) = k_y\, q_y \quad \Gamma(\overline{z}) = \overline{k}_z\, \overline{q}_z \quad \mathrm{typeof}(m) = k_{this}\, q_{this}, \overline{k}_p\, \overline{q}_p \rightarrow k_{ret}\, q_{ret}$
$k_y <: k_{this} \quad \overline{k}_z <: \overline{k}_p \quad k_{ret} <: k_x$

$q_{this\text{-}vp} = q_y \rhd q_{this} \qquad \overline{q}_{p\text{-}vp} = q_y \rhd \overline{q}_p \qquad q_{ret\text{-}vp} = q_y \rhd q_{ret}$

$q_{this\text{-}vp} = \mathit{substitutablepolymutable}\ \lor\ \overline{q}_{p\text{-}vp} = \mathit{substitutablepolymutable}\ \lor\ q_{ret\text{-}vp} = \mathit{substitutablepolymutable}\ \Rightarrow\ s$ exists

$$q_y <: \begin{cases} q_{this\text{-}vp} & \mathit{if}\ \ q_{this\text{-}vp} \neq \mathit{substitutablepolymutable} \\ s & \mathit{else} \end{cases}$$

$$\overline{q}_z <: \begin{cases} \overline{q}_{p\text{-}vp} & \mathit{if}\ \ \overline{q}_{p\text{-}vp} \neq \mathit{substitutablepolymutable} \\ s & \mathit{else} \end{cases}$$

$$q_x :> \begin{cases} q_{ret\text{-}vp} & \mathit{if}\ \ q_{ret\text{-}vp} \neq \mathit{substitutablepolymutable} \\ s & \mathit{else} \end{cases}$$

$$\Gamma \vdash x = y.[s]m(\overline{z})$$

(T-CALL)

Note: inference of s is another subproblem. It is disussed in the last page.

$$kd \text{ in } C \qquad C <: D \qquad typeof(D) = \overline{k}_{p\text{-}D}\ \overline{q}_{p\text{-}D} \rightarrow q_{ret\text{-}D} \qquad typeof(kd) = \overline{\_\ \_} \rightarrow q_{ret\text{-}C}$$

$$q_{ret\text{-}C} = \begin{cases} \_ & \textit{if } q_{ret\text{-}D} = \text{receiverdependantmutable} \\ \text{immutable} & \textit{if } q_{ret\text{-}D} = \text{immutable} \\ \text{mutable} & \textit{if } q_{ret\text{-}D} = \text{mutable} \end{cases}$$

$$\frac{\Gamma(z) = k_z\ q_z \qquad \overline{k}_z <: \overline{k}_{p\text{-}D} \qquad \overline{q}_z <: q_{ret\text{-}C} \rhd \overline{q}_{p\text{-}D}}{\Gamma \vdash \text{super}(\overline{z}) \text{ in } kd} \qquad (\text{ T-SUPER })$$

* Previously, when $q_{ret\text{-}D}$ = mutable, $q_{ret\text{-}C}$ can still be immutable. Because at that time, immutable constructors only have immutable or receiverdependantmutable parameters(does not exist anymore), thus any mutable objet created locally cannot escape and be captured by outside objects; Neither outside mutable objects will be captured by the receiverdependantmutable field when invoking mutable super constructor in immutable constructor. But now, immutable and receiverdependantmutable constructors don't have such restrictions(mutable parameters are allowed in both cases) any more, so outside mutable objects can be captured by receiverdependantmutable field. If we allow calling mutable super() in immutable subclass constructor, when we use $this_{sub}$.rdmf to access the field, the result is not guarantee to be immutable(may be the mutable object assigned in super mutable constructor). Therefore, we don't allow this kinds of flexibility and require subclass and superclass constructors should have the exact same qualifier if $q_{ret\text{-}D} \neq$ receiverdependantmutable

(T-THIS) (omitted)

* *Note:* In real Java code, one class can have multiple overloaded consturctors. One constructor can invoke the other by "this(..., ...)". The type rule T-THIS is very much the same as T-SUPER except that the constructor invoked by "this(..., ...)" comes from the same class.

$$
\frac{\begin{array}{l}
\Gamma(x) = k_x\, q_x \quad \Gamma(\overline{y}) = \overline{k_y}\, \overline{q_y} \quad \mathrm{typeof}(C) = \overline{k_p}\, \overline{q_p} \rightarrow q_{ret} \\
\overline{q_y} <: q \triangleright \overline{q_P} \quad q <: q \triangleright q_{ret} \quad q \neq \mathrm{readonly} \\
\overline{k_y} <: \overline{k_p} \\
q <: q_x \quad k <: k_x \qquad k = \begin{cases} \mathrm{initialized} & \textit{if } \overline{k_p} = \textit{initialized} \\ \mathrm{underinitialized} & \textit{otherwise} \end{cases}
\end{array}}{\Gamma \vdash x = \mathrm{new}\ q\ C(\overline{y})} \quad \text{(T-NEW)}
$$

$$
\frac{\Gamma \vdash s_1 \quad \Gamma \vdash s_2}{\Gamma \vdash s_1;s_2} \quad \text{(T-SEQ)}
$$

Figure 3 Statement typing

Well-formdness Rules

$$
\frac{\mathrm{fType}(fd) = q\ \_ \quad q \neq \mathrm{polymutable} \quad fd \notin \mathrm{fields}(D)}{\vdash_C\ fd \text{ is OK}} \quad \text{(WF-FLD)}
$$

$$
\frac{\begin{array}{l}
\mathrm{cBody}(kd) = \mathrm{super}(g);\ \mathrm{this}.f = f \qquad \mathrm{typeof}(kd) = \overline{k_p}\ \overline{q_p} \rightarrow q_{ret} \\
q_{ret} = \mathrm{mutable} \lor q_{ret} = \mathrm{immutable} \lor q_{ret} = \mathrm{receiverdependantmutable} \\
q_C = \mathrm{mutable} \Rightarrow q_{ret} = \mathrm{mutable} \\
q_C = \mathrm{immutable} \Rightarrow q_{ret} = \mathrm{immutable} \\
\Gamma = (\mathrm{this} : \mathrm{underinitialized}\ q_{ret},\ \overline{p} : \overline{k_p}\ \overline{q_p},\ \overline{y} : \overline{k_{local}}\ \overline{q_{local}}) \\
\Gamma \vdash \mathrm{super}(\overline{y})\ \mathrm{in}\ kd \qquad \Gamma \vdash \mathrm{this}.f = f
\end{array}}{\vdash_C kd\ \mathrm{is\ OK}} \quad (\text{WF-CONS})
$$

Note: $\vdash_{C\ kd}$ reads "constructor kd in class C is well-formed".

$$
\frac{\begin{array}{l}
\mathrm{mBody}(md) = \overline{s};\mathrm{return}\ z \qquad \mathrm{typeof}(md) = k_{this}\ q_{this},\ \overline{k_p}\ \overline{q_p} \rightarrow k_{ret}\ q_{ret} \\
\Gamma = (\mathrm{this} : k_{this}\ q_{this},\ \overline{p} : \overline{k_p}\ \overline{q_p},\ \overline{y} : \overline{k_{local}}\ \overline{q_{local}}) \quad \Gamma \vdash \overline{s} \quad \Gamma(z) <: t_{ret} \\
q_C = \mathrm{mutable} \Rightarrow q_{this} \neq \mathrm{immutable} \\
q_C = \mathrm{immutable} \Rightarrow q_{this} \neq \mathrm{mutable} \\
\mathrm{typeof}(md\text{-}super) = k_{this\text{-}super}\ q_{this\text{-}super},\ \overline{k_{p\text{-}super}}\ \overline{q_{p\text{-}super}} \rightarrow k_{ret\text{-}super}\ q_{ret\text{-}super} \\
k_{this\text{-}super} <: k_{this} \qquad k_{p\text{-}super} <: k_p \qquad k_{ret} <: k_{ret\text{-}super} \\
q_C \triangleright q_{this\text{-}super} <: q_{this} \qquad q_C \triangleright q_{p\text{-}super} <: q_p \qquad q_{ret} <: q_C \triangleright q_{ret\text{-}super}
\end{array}}{\vdash_C md\ \mathrm{is\ OK}} \quad (\text{WF-METH})
$$

Note: $\vdash_{C\ md}$ reads "method md in class C is well-formed".

$$
\frac{q_D = \mathrm{mutable} \Rightarrow q_C = \mathrm{mutable} \qquad q_D = \mathrm{immutable} \Rightarrow q_C = \mathrm{immutable}}{\vdash C <: D\ \mathrm{is\ OK}} \quad (\text{WF-EXTEND})
$$

Note: 1) q_D is annotation on declaration of class D.
2) In the formalization, implements is not supported. But in real Java, implements are treated the same as extends.

$$
\frac{\begin{array}{l}
\vdash_C \overline{fd}\ \mathrm{is\ OK} \quad \vdash_C kd\ \mathrm{is\ OK} \quad \vdash_C \overline{md}\ \mathrm{is\ OK} \quad \vdash D\ \mathrm{is\ OK} \quad \vdash C <: D\ \mathrm{is\ OK} \\
\quad q_C = \mathrm{mutable} \lor q_C = \mathrm{immutable} \lor q_C = \mathrm{receiverdependantmutable}
\end{array}}{\vdash C\ \mathrm{is\ OK}} \quad (\text{WF-CLASS})
$$

$$\frac{\vdash C \text{ is OK} \quad q_C = \text{mutable} \Rightarrow q_{use} \neq \text{immutable} \quad q_C = \text{immutable} \Rightarrow q_{use} \neq \text{mutable}}{\vdash q_{use}\ C \text{ is OK}} \quad \text{(WF-TYPEUSE)}$$

Figure 4 Well-formdness typing

Extension to real Java with statics and blocks

In real Java, there are static fields, static methods, initialization blocks.

Helper Method

usedQualifiers($\overline{s}$) returns all immutability qualifiers used in $\overline{s}$ recursively

$cd ::= q_C$ class C extends D { $\overline{sfd}$ $\overline{fd}$; $\overline{sib}$ $\overline{ib}$ kd $\overline{smd}$ $\overline{md}$ }	class
sfd ::= static q a C sf	static field
smd ::= static t C sm (t C x) { $\overline{t\ C\ y}$ $\overline{s}$; return z; }	static method
sib ::= static{$\overline{s}$;}	static initialization block
ib ::= {$\overline{s}$;}	initialization block

$$\frac{\text{fType(sfd)} = q\ a \quad q \neq \text{polymutable} \wedge q \neq \text{receiverdependantmutable} \quad a \neq \text{receiverdependantassignable}}{\vdash \text{sfd is OK}} \quad \text{(WF-STATIC-FLD)}$$

$$\frac{\begin{array}{l}\text{mBody(smd)} = \overline{s};\text{return } z \qquad \text{typeof(smd)} = \overline{k_p}\ \overline{q_p} \rightarrow t_{ret} \\ \Gamma = (\overline{p} : \overline{k_p}\ \overline{q_p},\ \overline{y} : \overline{k_{local}}\ \overline{q_{local}}) \qquad \Gamma \vdash \overline{s} \qquad \Gamma(z) <: t_{ret} \\ \overline{q_p} \neq \text{receiverdependantmutable} \wedge q_{ret} \neq \text{receiverdependantmutable} \\ \text{receiverdependantmutable} \notin \text{usedQualifiers}(\overline{s};\text{return } z)\end{array}}{\vdash \text{smd is OK}} \quad \text{(WF-STATIC-METH)}$$

$$\frac{\Gamma \vdash \overline{s} \qquad receiverdependantmutable \notin usedQualifiers(\overline{s})}{\vdash sib \text{ is OK}} \quad \text{(WF-STATIC-BLK)}$$

$$\frac{\Gamma \vdash \overline{s}}{\vdash_C ib \text{ is OK}} \quad \text{(WF-BLK)}$$

$$\frac{\begin{array}{l}\vdash \overline{sfd} \text{ is OK} \quad \vdash_C \overline{fd} \text{ is OK} \quad \vdash \overline{sib} \text{ is OK} \quad \vdash_C \overline{ib} \text{ is OK} \\ \vdash_C kd \text{ is OK} \quad \vdash \overline{smd} \text{ is OK} \quad \vdash_C \overline{md} \text{ is OK} \quad \vdash D \text{ is OK} \quad \vdash C <: D \text{ is OK} \\ q_C = mutable \lor q_C = immutable \lor q_C = receiverdependantmutable\end{array}}{\vdash C \text{ is OK}} \quad \text{(WF-CLASS)}$$

Inference of immutability qualifier for polymutable methods

After viewpoint adapting m() at the invocation site, if $q_{this\text{-}vp}$, $\overline{q}_{p\text{-}vp}$, $q_{ret\text{-}vp}$ are NOT substitutablepolymutable, standard subtyping rules apply:

$q_y <: q_{this\text{-}vp}$ $\qquad \overline{q}_z <: \overline{q}_{p\text{-}vp}$ $\qquad q_{ret\text{-}vp} <: q_x$

But if any of them is substitutablepolymutable, we use a variable **s** to replace corresponding $q_{this\text{-}vp}$, $\overline{q}_{p\text{-}vp}$, $q_{ret\text{-}vp}$ and add it/them to constraints set. After collecting all the constraints, we try to find a solution s that satisfies all the subtype constraints. If there is such a solution, then method invocation typechecks; Otherwise, it doesn't typecheck.

For example, assuming we have a method after viewpoint adaptation with signature:
substitutablepolymutable Object m(substitutablepolymutable A this, substitutablepolymutable Object p);
If we invoke it as:
immutable A a;
readonly Object ro = a.m(new immutable Object());

Constraints are collected in this way:
immutable <: s $\qquad$ immutable <: s $\qquad$ s <: readonly

We'll have a solution:
s = immutable(or readonly), so this method invocation typechecks.

But if we invoke the method as:
mutable Object ro = a.m(new immutable Object());
Then we have constraints:
immutable <: s   immutable <: s   s <: mutable
There is NO solution for s, so the type system rejects this method invocation.