## Constraint Based Typing

Γ ⊢ e:T | {C}
It means “expression e has type T under assumptions whenever the constraints C are satisfied”.

Γ ⊢ s | {C}
It means “statement s typechecks under assumptions whenever the constraints C are satisfied”.

Γ ⊢ X is well-formed | {C}
It means “element X is well-formed under assumptions whenever the constraints C are satisfied”. X can be field, constructor, method, class, blocks.

**Note:**
**1)** PICOInfer assumes the input program is well-typed in Freedom-Before-Committment(FBC) type system and initialization qualifiers are given already.
**2)** PICOInfer assumes that input program is well-formed in terms of assignability on fields(meaning one and only one assignability qualifier is used on a field; no @RDA is used on static fields) and assignability qualifiers are given already.
**3)** Therefore, PICOInfer **only infers solution in mutability hierarchy**, not the ~~initialization hierarchy or assignability dimension~~.
**4)** In PICOInfer formalization, we only write initialization and assigability qualifiers in assumptions that affect how we generate mutability constraints, but not the in the conclusions(because it doesn’t contribute to contraint generation and those two dimensions are assumed to be valid already).

## Qualifier Hierarchy

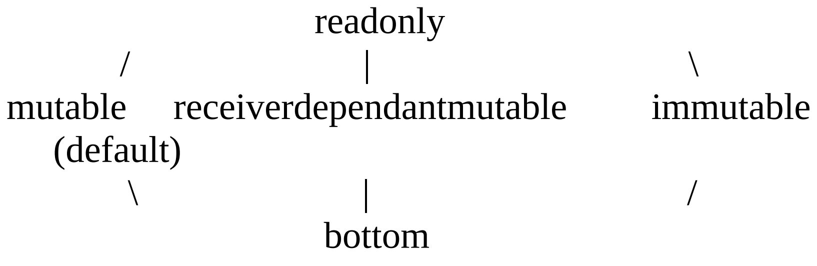

**Figure 1 Mutability Qualifiers** In PICOInfer we only infer solutions for reduced version of mutability qualifiers. Polymutable and substitutablepolymutable are removed and won’t be inferred. But later on, we may implement inferring them

**Type Environment**

$\Gamma = \Gamma_k \cup \boldsymbol{\Gamma_q} \cup \Gamma_a$

*** Note :** $\Gamma_k$ is the type environment that only stores initialization qualifiers of variables. $\Gamma_q$ is the type environment that only stores mutability qualifiers of variables. $\Gamma_a$ is the type environment that only stores assignability qualifiers of variables. We use only $\Gamma_q$ extensibely in the type rules because we are only interested in mutability qualifiers. If we need extra information such as initialization qualifiers or assignability qualifiers too, $\Gamma$ is used instead.

**Helper Functions**

fType(f), cBody(kd), mBody(md) returns only mutability qualifiers by default. But if initialization and assignability information are needed, then they are also returned.

**Viewpoint Adaptation Rules**

_ ▹ mutable = mutable
_ ▹ readonly = readonly
_ ▹ immutable = immutable
_ ▹ bottom = bottom
q ▹ receiverdependantmutable = q

**Special Rules**

- Generate inequlity constraint $q \neq$ bottom everywhere except on (implicit/explicit) lower bounds(null literal is implicitly bottom).

**Typing Rules (Constraint Based)**

$$\frac{x \in \Gamma_q}{\Gamma_q \vdash x : \Gamma_q(x) \mid \{\}} \qquad \text{(CT-VAR)}$$

$$\frac{\Gamma_q(x) = q_x \qquad fType(f) = q_f}{\Gamma_q \vdash x.f : q \mid \{q = q_x \rhd q_f\}} \qquad \text{( CT-FLD )}$$

**Figure 2 Expression typing**

Simple variable lookup from the environment doesn’t need precondition constraints and neither introduces new constraints.

$$\frac{\Gamma_q \vdash e = q_e \mid \{C\}}{\Gamma_q \vdash x = e \mid \{q_e <: \Gamma_q(x)\ , C\}} \qquad \text{( CT-VARASS )}$$

Expression judgement and statement judgement do need precondition constraints to hold and introduce new constraints, too.

$$\frac{\Gamma(x) = \ k_x\, q_x \qquad \Gamma_q(y) = \ q_y \qquad typeof(f) = q_f\ a_f}{\Gamma_q \vdash x.f = y \mid \{q_y <: q_x \rhd q_f\, ,\ fieldWrite(k_x\, , q_x\, , a_f)\ \}} \qquad \text{( CT-FLDASS)}$$

```
fieldWrite($k_x$ , $q_x$ , $a_f$) :: =
    if $a_f$ = assignable =>  $q_x \neq readonly$ | $q_f \neq receiverdependantmutable$
    else if  $k_x$ = underinitialized => $q_x$  = mutable | $q_x$ = immutable |  $q_x$ = receiverdependantmutable
    else =>  $q_x$  = mutable
```

_*Note : Red | means CFI currently doesn’ t support encoding disjunction. We can implement the second one by using mainIsNot(readonly), but the first one is disjunction of multiple variables and can’t currently be implemented in CFI. Maybe we can choose one that makes more sense(I think we should choose $q_x \neq$ readonly, because we prefer fields to be receiverdependantmutable, and $q_x$ can give use more information)_

$$\frac{\Gamma_q(x) = q_x \qquad \Gamma_q(y) = q_y \qquad \Gamma_q(\overline{z}) = \overline{q}_z \qquad \mathrm{typeof}(m) = q_{this}, \overline{q}_p \rightarrow q_{ret}}{\Gamma_q \vdash x = y.m(\overline{z}) \mid \{q_{this\text{-}vp} = q_y .\rhd q_{this}, \overline{q}_{p\text{-}vp} = q_y \rhd \overline{q}_p, q_{ret\text{-}vp} = q_y \rhd q_{ret}, q_y <: q_{this\text{-}vp}, \overline{q}_z <: \overline{q}_{p\text{-}vp}, q_{ret\text{-}vp} <: q_x\}} \quad \text{( CT-CALL )}$$

**TODO** : inference of polymutable methods will be implemented later.

$$\frac{\text{kd in C} \qquad C <: D \qquad \mathrm{typeof}(D) = \overline{q}_{p\text{-}D} \rightarrow q_{ret\text{-}D} \qquad \mathrm{typeof}(kd) = \overline{\_} \rightarrow q_{ret\text{-}C} \qquad \Gamma_q(\overline{z}) = \overline{q}_z}{\Gamma_q \vdash \mathrm{super}(\overline{z}) \text{ in kd} \mid \{q_{ret\text{-}C} <: q_{ret\text{-}C} \rhd q_{ret\text{-}D}, \overline{q}_z <: q_{ret\text{-}C} \rhd \overline{q}_{p\text{-}D}\}} \quad \text{( CT-SUPER )}$$

( CT-THIS ) (omitted)
***Note**:* As the type rule in the typechecking side, constraint based type rule CT-THIS is also the same as CT-SUPER except that the constructor invoked by “this(..., ...)” comes from the same class.

$$\frac{\Gamma_q(x) = q_x \qquad \Gamma_q(\overline{y}) = \overline{q}_y \qquad \mathrm{typeof}(C) = \overline{q}_p \rightarrow q_{ret}}{\Gamma_q \vdash x = \text{new } q\ C(\overline{y}) \mid \{\overline{q}_y <: q \rhd \overline{q}_P, q <: q \rhd q_{ret}, q <: q_x, q \neq \text{readonly}\}} \quad \text{(CT-NEW)}$$

$$\frac{\Gamma_q \vdash s_1 \mid \{C1\} \qquad \Gamma_q \vdash s_2 \mid \{C2\}}{\Gamma_q \vdash s_1;s_2 \mid \{C1 , C2\}} \quad \text{(CT-SEQ )}$$

**Figure 3 Statement typing**

## Well-formdness Rules (Constraint Based)

$$\frac{\mathrm{fType}(fd) = q \qquad C <: D \qquad fd \notin \mathrm{fields}(D)}{\vdash_C \ fd \text{ is OK} \mid \{\}} \qquad \text{(CWF-FLD)}$$

$$\frac{\qquad}{\vdash_{object} \ kd \text{ is OK} \mid \{\}} \qquad \text{(CWF-CONS-OBJECT)}$$

$$\frac{\begin{array}{l} \mathrm{cBody}(kd) = \mathrm{super}(g);\ \mathrm{this}.f = f \qquad \mathrm{typeof}(kd) = \overline{q_p} \rightarrow q_{ret} \\ \Gamma = (\mathrm{this} : \mathrm{underinitialized}\ q_{ret},\ \overline{p} : \overline{q_p},\ \overline{y} : \overline{q_{local}}) \\ \Gamma_q \vdash \mathrm{super}(\overline{y}) \text{ in } kd \mid \{C1\} \quad \Gamma_q \vdash \mathrm{this}.f = f \mid \{C2\} \end{array}}{\vdash_C kd \text{ is OK} \mid \{q_{ret} \neq \mathrm{readonly}\ ,\ q_{ret} \neq \mathrm{polymutable}\ ,\ C1\ ,\ C2\}} \qquad \text{(CWF-CONS)}$$

*Note*: $\vdash_C$ kd reads “constructor kd in class C is well-formed”.

$$\frac{\begin{array}{l} \mathrm{mBody}(md) = \overline{s};\mathrm{return}\ z \qquad \mathrm{typeof}(md) = q_{this},\ \overline{q_p} \rightarrow q_{ret} \\ \Gamma_q = (\mathrm{this} : q_{this},\ \overline{p} : \overline{q_p},\ \overline{y} : \overline{q_{local}}) \quad \Gamma_q \vdash \overline{s} \mid \{C1\} \ \ \Gamma_q(z) <: q_{ret} \mid \{C2\} \\ \text{Standard method overriding rule holds} \mid \{C3\} \end{array}}{\vdash_C md \text{ is OK} \mid \{C1\ ,\ C2\ ,\ C3\}} \qquad \text{(CWF-METH)}$$

$$\frac{\vdash_C \overline{fd} \text{ is OK } \mid \{C1\} \qquad \vdash_C kd \text{ is OK} \mid \{C2\} \qquad \vdash_C \overline{md} \text{ is OK} \mid \{C3\}}{\vdash C \text{ is OK} \mid \{C1 , C2 , C3\}} \quad \text{(CWF-CLASS)}$$

**Figure 4 Well-formdness typing**

## Extension to real Java with statics and blocks

$$\frac{fType(sfd) = q}{\vdash\ sfd \text{ is OK} \mid \{q \neq receiverdependantmutable\}} \quad \text{(CWF-STATIC-FLD)}$$

$$\frac{\begin{array}{l} mBody(smd) = \overline{s};\text{return } z \qquad typeof(smd) = \overline{q}_p \rightarrow q_{ret} \\ \Gamma_q = (\overline{p} : \overline{q}_p,\ \overline{y} : \overline{q}_{local}) \qquad \Gamma_q \ \vdash\ \overline{s} \mid \{C1\} \qquad \Gamma_q(z) <: q_{ret} \mid \{C2\} \end{array}}{\begin{array}{l} \vdash smd \text{ is OK} \mid \{C1 , C2 , \overline{q}_p \neq receiverdependantmutable , q_{ret} \neq \\ receiverdependantmutable , \overline{q}_{local} \neq receiverdependantmutable , \text{foreach } q \text{ in} \\ usedQualifiers(\overline{s};\text{return } z) : q \neq receiverdependantmutable\} \end{array}} \quad \text{(CWF-STATIC-METH)}$$

$$\frac{\Gamma_q \vdash \overline{s} \mid \{C\}}{\vdash\ sib \text{ is OK} \mid \{C , \text{foreach } q \text{ in } usedQualifiers(\overline{s}) : q \neq receiverdependantmutable \}} \quad \text{(CWF-STATIC-BLK)}$$

$$\frac{\Gamma_q \vdash\ \overline{s} \mid \{C\}}{\vdash_C\ ib \text{ is OK} \mid \{C\}} \quad \text{(CWF-BLK)}$$

$$\frac{\begin{array}{l} \vdash\ \overline{sfd} \text{ is OK} \mid \{C1\} \quad \vdash_C \overline{fd} \text{ is OK} \mid \{C2\} \quad \vdash_C kd \text{ is OK} \mid \{C3\} \quad \vdash \overline{smd} \text{ is OK} \mid \{C4\} \\ \vdash_C \overline{md} \text{ is OK} \mid \{C5\} \quad \vdash\ sib \text{ is OK} \mid \{C6\} \quad \vdash_C\ \overline{ib} \text{ is OK} \mid \{C7\} \end{array}}{\vdash C \text{ is OK} \mid \{C1 , C2 , C3 , C4 , C5 , C6 , C7\}} \quad \text{(CWF-CLASS)}$$