CIS 500 — Software Foundations

Final Exam

December 14, 2005

If taking the exam for WPE credit:

WPE-I id: ______________________________

Otherwise:

Name: ______________________________

Email ______________________________

Status ____ registered for the course

____ not registered

	Score
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
Total	

Instructions

- This is a closed-book exam: you may not make use of any books or notes.
- You have 120 minutes to answer all of the questions. The entire exam is worth 120 points.
- Questions vary significantly in difficulty, and the point values of questions may not be exactly proportional to their difficulty. Do not spend too much time on any one question.
- It is a good idea to read through the entire exam before starting to work hard on individual problems.
- Partial credit will be given wherever possible. All correct answers are short. The back side of each page may be used as a scratch pad.
- Good luck!

True/False questions

For each of the following statements, circle T if the sentence is true or F otherwise.

1. (9 points)

   (a) T F The small-step evaluation relation of a language must be deterministic (i.e. for any term there should be only way for it to take a step) for the preservation theorem to hold.

   (b) T F The uniqueness of types property (i.e., in a given context Γ, a term $\mathtt{t}$ has at most one type $\mathtt{T}$. Furthermore, there is exactly one derivation of $\Gamma \vdash \mathtt{t} : \mathtt{T}$.) must be true about a language to prove the preservation theorem.

   (c) T F If the preservation theorem is true for a language, removing a typing rule may cause it to become untrue.

   (d) T F If the progress theorem is true for a language, removing a typing rule may cause it to become untrue.

   (e) T F In the pure untyped lambda calculus (without booleans, natural numbers, or anything other than functions) all closed terms will either diverge or evaluate to a value. (For reference, this language is shown on page 1 of the companion handout.)

   (f) T F The algorithmic rules for the lambda calculus with subtyping has the uniqueness of types property. (For reference, this language is shown on page 4 of the companion handout.)

   (g) T F The declarative rules for the lambda calculus with subtyping has the uniqueness of types property. (For reference, this language is shown on page 5 of the companion handout.)

   (h) T F In Featherweight Java (FJ), all programs will either diverge or evaluate to a value. (For reference, this language is shown on page 9 of the companion handout.)

   (i) T F FJ is specified with a large-step operational semantics.

Inductive definitions

2. (14 points) We can define a simple term language as follows:

```
t ::= □
      *
      t ∧ t
```

Now consider the following binary relation between these terms, specified by the following inference rules:

$$\frac{}{* \sim \square}\ \textsc{Ax1} \qquad \frac{}{\square \sim *}\ \textsc{Ax2}$$

$$\frac{\mathtt{t}_1 \sim \mathtt{t}_4 \qquad \mathtt{t}_2 \sim \mathtt{t}_3}{(\mathtt{t}_1 \land \mathtt{t}_2) \sim (\mathtt{t}_3 \land \mathtt{t}_4)}\ \textsc{Amp1} \qquad \frac{(\mathtt{t}_2 \land \mathtt{t}_1) \sim (\mathtt{t}_4 \land \mathtt{t}_3)}{(\mathtt{t}_1 \land \mathtt{t}_2) \sim (\mathtt{t}_3 \land \mathtt{t}_4)}\ \textsc{Amp2}$$

(a) Draw a derivation for $(* \land \square) \sim (* \land \square)$

(b) As you may have noticed, this set of rules is not-syntax directed. Give a different derivation for $(* \land \square) \sim (* \land \square)$.

(c) Fortunately, we can remove exactly one rule and produce a relation that (a) is syntax-directed and (b) equivalent to the previous relation. Which rule should we eliminate?

(d) For any x does there exist at least one y such that $x \sim y$? If yes, prove it. If no, show a counterexample. Be explicit in your answer, but to the point. Points may be deducted for any extraneous information (true or false).

Untyped lambda-calculus

The following questions refer to the untyped lambda-calculus. The syntax and evaluation rules for this system are given on page 1 of the companion handout.

3. (12 points) Circle the normal forms of the following lambda calculus terms, if one exists. If there is no normal form, circle NONE. (Recall that the normal form of `t` is a term `u` such that `t` $\longrightarrow^*$ `u` and `u` $\not\longrightarrow$.)

   (a) `(λy. (λz. x y)) (λx. z)`

   i. `(λy. (λz. x y)) (λx. z)`
   ii. `(λz. x (λx. z))`
   iii. `(λw. (λx.z) (λx.z))`
   iv. `(λw. x (λx.z))`
   v. NONE

   (b) `(λy. (λz. z z ) y)(λx. x)`

   i. `(λy. (λz. z z ) y)(λx. x)`
   ii. `(λz. z z)(λx.x)`
   iii. `(λx. x)`
   iv. `(λy. y y)(λz.z)`
   v. NONE

   (c) `(λx. x x x) (λx. x x x)`

   i. `(λx. x x x) (λx. x x x)`
   ii. `(λx. x x x)`
   iii. `(λx. x x x)(λx. x x x)(λx. x x x)`
   iv. `x x x`
   v. NONE

   (d) `(λx. (λy. y y)(λz. z z))`

   i. `(λx. (λy. y y)(λz. z z))`
   ii. `(λx. (λy. y y))`
   iii. `(λy. (λz. z z))`
   iv. `(λy. y y)(λz. z z)`
   v. NONE

Recall the encoding of booleans and numbers in the untyped lambda calculus from Chapter 5 of TAPL. The next two questions concern that encoding.

tru = λt. λf. t

fls = λt. λf. f

c_0 = λs.λz.z

scc = λn. λs. λz. s (n s z)

4. (3 points) Which of these lambda calculus terms implements xor (the exclusive or function, which returns tru when exactly one of its arguments is tru.)

   (a) λx. λy. x (y fls tru) (y tru fls)

   (b) λx. λy. x y y

   (c) λx. λy. tru x y

   (d) λx. λy. x y fls

5. (3 points) Which of these lambda calculus terms implements odd, a function that returns tru if its argument (the encoding of a natural number) is odd and fls otherwise.

   (a) λm. m (λn. n fls tru) fls

   (b) λm. m fls (λn. tru fls)

   (c) λm. fls (λn. n m tru)

   (d) λm. m (λn. tru) fls

Implementing simple type systems

The following questions refer to the Arith language, a typed calculus with booleans and natural numbers. The syntax, typing, and evaluation rules for this system are given on page 2 of the companion handout.

6. (12 points) The `eval1` function below implements the small-step evaluation relation *almost* correctly, but there are mistakes in the `TmIf`, `TmSucc`, `TmPred` and `TmIsZero` cases of the outer match. Show how to change the code to repair at least **one mistake in each branch**. (For simplicity, file information has been omitted from the datatype.)

```
type exp =
  TmZero | TmSucc of exp | TmPred of exp
| TmIsZero of exp| TmTrue | TmFalse | TmIf of exp * exp * exp

let rec isnumericval t = ... (* returns true when t is a numeric value *)
let rec isval t = ... (* returns true when t is a value. *)

exception NoRuleApplies

let rec eval1 t = match t with

   v when isval v → raise NoRuleApplies

| TmIf(t1,t2,t3) →

    (match t1 with

       TmTrue → t2

     | TmFalse → t3

     | _ → raise NoRuleApplies)

| TmSucc(t1) → TmSucc(t1)

| TmPred(t1) →

    (match t1 with

       TmZero → TmZero

     | TmSucc(nv1) → nv1

     | _ → TmPred(eval1 t1))

| TmIsZero(t1) →

    (match t1 with

    TmZero → TmFalse

     | TmSucc(nv1) when isnumericval nv1 → TmFalse

     | _ → TmIsZero(eval1 t1))
```

Simply typed lambda-calculus with subtyping, records, and references

The following questions refer to the simply typed lambda-calculus with subtyping, records, and references. The syntax, typing, and evaluation rules for this system are given on page 5 of the companion handout.

7. (12 points)

   What is the minimal (or principal) type of the following expressions in the simply-typed lambda-calculus with subtyping, records and references? If a term does not type check, write NONE.

   (a) `(λx:Top. x) {a=2, b=3}`

   (b) `λy:{}→Ref Top. λx:Top. y x`

   (c)
   ```
   if true then λx:Ref Nat. { y={b=!x}, d=!x }
           else λx:Ref Nat. { y={a=2, b=3} }
   ```

   (d)
   ```
   if true then λx:Ref Top. !x
           else λx:Ref Nat. !x
   ```

8. (6 points) Suppose we add a new axiom

$$\texttt{Top} \to \texttt{Top} \; \texttt{<:} \; \texttt{\{\}}$$

to the rules defining the subtype relation. Does the progress theorem remain true in the new system? Briefly explain why or why not.

9. (6 points) Suppose, instead, that we add the subtyping axiom

$$\texttt{Top} \; \texttt{<:} \; \texttt{Top} \to \texttt{Top}$$

to the original system. Does the progress theorem remain true? Briefly explain why or why not.

10. (5 points) Subtyping references is quite harsh: the rule S-REF requires both covariance and contravariance for the type parameter. But we can do better.

Suppose we replace the type `Ref T` with the type `Ref T U`. The first parameter `T` is used when the reference is read, the second `U` is when the reference is written. When a reference is created, both of these parameters are the same.

We make this change by modifying the following typing rules for references:

$$\frac{\Gamma \vdash \texttt{t} : \texttt{Ref S T}}{\Gamma \vdash \texttt{!t} : \texttt{S}} \quad \text{(T-DEREF)}$$

$$\frac{\Gamma \vdash \texttt{t} : \texttt{Ref S T} \quad \Gamma \vdash \texttt{u} : \texttt{T}}{\Gamma \vdash \texttt{t := u} : \texttt{Unit}} \quad \text{(T-ASSIGN)}$$

$$\frac{\Gamma \vdash \texttt{t} : \texttt{T}}{\Gamma \vdash \texttt{ref t} : \texttt{Ref T T}} \quad \text{(T-REF)}$$

For example, suppose we have the following type abbreviations:

```
T = {a:Bool}
U = {a:Bool,b:Nat}
```

With these rules, the following function `f`, should type check. If the argument `z` has type `Ref T U`, then the dereference (`!z`) has type `T={a:Bool}` so we are allowed to access the `a` component of the record. When we assign to `z`, we use the type `U` which matches the type of the two records.

```
f = λz:Ref T U → Unit.
      if (!z).a then
        z := {a = false, b=1}
      else
        z := {a = true, b=3}
```

We should be able to apply the function `f` to, for example, an argument of type `Ref T T` or of type `Ref U U`. However, we cannot do that unless we have a way of showing that `Ref T T <: Ref T U` and that `Ref U U <: Ref T U`. (Note that, in the original system, there is no common supertype of the types `Ref T` and `Ref U`.)

Therefore, we need to be careful when designing the rule for subtyping reference types. What should the preconditions of this rule be?

$$\frac{\qquad\qquad\qquad\qquad\qquad\qquad}{\texttt{Ref}\ \texttt{S}_1\ \texttt{T}_1 <: \texttt{Ref}\ \texttt{S}_2\ \texttt{T}_2} \quad \text{S-REF}$$

Featherweight Java

The following questions refer to the Featherweight Java language. The syntax, typing, and evaluation rules for this system are given on page 9 of the companion handout.

11. (5 points) The Preservation lemma for Featherweight Java is *not* stated as:

    If $\Gamma \vdash \mathtt{t} : \mathtt{C}$ and $\mathtt{t} \longrightarrow \mathtt{t}'$ then $\Gamma \vdash \mathtt{t}' : \mathtt{C}$.

    Why not? Justify your answer.

12. (15 points) The method overriding rule is rather restrictive in FJ. Any overridden method must have exactly the same type as it appears in the super class.

We could relax this rule as follows:

$$\frac{mtype(\texttt{m}, \texttt{D}) = \overline{\texttt{D}} \rightarrow \texttt{D}_0 \text{ implies } \overline{\texttt{D}} \texttt{ <: } \overline{\texttt{C}} \text{ and } \texttt{C}_0 \texttt{ <: } \texttt{D}_0}{override(\texttt{m}, \texttt{D}, \overline{\texttt{C}} \rightarrow \texttt{C}_0)}$$

For an example of a program that uses this rule, see the next problem.

Now, given the following lemma:

Override Lemma: If $\texttt{mtype(m,D}_0\texttt{)} = \overline{\texttt{D}} \rightarrow \texttt{D}$ then for all $\texttt{C}_0 \texttt{ <: } \texttt{D}_0$, $\texttt{mtype(m,C}_0\texttt{)} = \overline{\texttt{C}} \rightarrow \texttt{C}$ where $\texttt{C} \texttt{ <: } \texttt{D}$ and $\overline{\texttt{D}} \texttt{ <: } \overline{\texttt{C}}$.

Show part of the proof for the substitution lemma for FJ:

Substitution Lemma: If $\Gamma, \overline{\texttt{x}}\texttt{:}\overline{\texttt{B}} \vdash \texttt{t : D}$ and $\Gamma \vdash \overline{\texttt{s}}\texttt{:}\ \overline{\texttt{A}}$ where $\overline{\texttt{A}}\texttt{<:}\overline{\texttt{B}}$ then $\Gamma \vdash [\overline{\texttt{x}} \mapsto \overline{\texttt{s}}]\texttt{t : C}$ for some $\texttt{C} \texttt{ <: } \texttt{D}$.

This lemma is proved by induction on the typing derivation $\Gamma, \overline{\texttt{x}}\texttt{:}\overline{\texttt{B}} \vdash \texttt{t : D}$. **You need only show the case when** T-INVK **is the last rule of that derivation.** Furthermore, you may make use of the *Override Lemma* **without** proving it. Be explicit in your answer, but to the point. Points may be deducted for any extraneous information (true or false).

Object encodings

13. (12 points) Consider the following Java class definitions (with the relaxed rule for method overriding as discussed in the previous question):

```
class A extends Object {
  Object f1;
  A(Object f1) { super(); this.f1 = f1; }
  Object m(Object x) { return this.n(); }
  Object n() { return this.m(this.f1); }
}
class B extends A {
  A f2;
  B(Object f1, A f2) { super(f1); this.f2 = f2; }
  A m(Object x) { return this.f2; }
  Object n() { return super.n(); }
}
```

Complete the encoding of the classes A and B into the simply-typed lambda-calculus with subtyping (as shown on page 5 of the companion handout), in the style of Section 18.11 of TAPL.

```
RepA = { f1 : Ref Object }
RepB = { f1 : Ref Object, f2 : Ref A }
A    = { m  : Object → Object, n : Unit → Object}


B    =  ______________________________________________

aClass = λr:RepA. λthis:Unit → A. λ_:Unit.


   { m = λx:Object.  ______________________________________________ ,


     n = λ_:Unit.  ______________________________________________  }
bClass = λr:RepB. λthis:Unit → B. λ_:Unit.
   let super = aClass r this unit in
   { m = λx:A. !(r.f2),


     n = λ_:Unit. ______________________________________________  }
newA = λf1val:Object.
    let r = { f1 = ref f1val } in


     ______________________________________________
newB = λf1val:Object. λf2val:A.
    let r = { f1 = ref f1val, f2 = ref f2val } in


     ______________________________________________
```

14. (6 points)

    Suppose that `v` and `w` are values of type `A`. Write down the final results of evaluating the following terms, using the **encoding** from the previous problem. If there is no final result, write *diverges*.

    (a) `(newA v).n unit`

    (b) `(newB v w).m w`

    (c) `(newB v w).n unit`

Companion handout

Full definitions of the systems used in the exam

Untyped lambda-calculus

Syntax

$\mathtt{t}$ ::=		*terms*
	$\mathtt{x}$	*variable*
	$\lambda\mathtt{x.t}$	*abstraction*
	$\mathtt{t\ t}$	*application*
$\mathtt{v}$ ::=		*values*
	$\lambda\mathtt{x.t}$	*abstraction value*

Evaluation $\boxed{\mathtt{t} \longrightarrow \mathtt{t}'}$

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{t}_1\ \mathtt{t}_2 \longrightarrow \mathtt{t}_1'\ \mathtt{t}_2} \quad \text{(E-App1)}$$

$$\frac{\mathtt{t}_2 \longrightarrow \mathtt{t}_2'}{\mathtt{v}_1\ \mathtt{t}_2 \longrightarrow \mathtt{v}_1\ \mathtt{t}_2'} \quad \text{(E-App2)}$$

$$(\lambda\mathtt{x.t}_{12})\ \mathtt{v}_2 \longrightarrow [\mathtt{x} \mapsto \mathtt{v}_2]\mathtt{t}_{12} \quad \text{(E-AppAbs)}$$

For reference: Boolean and arithmetic expressions

Syntax

`t`	::=		*terms*
		`true`	*constant true*
		`false`	*constant false*
		`if t then t else t`	*conditional*
		`0`	*constant zero*
		`succ t`	*successor*
		`pred t`	*predecessor*
		`iszero t`	*zero test*
`v`	::=		*values*
		`true`	*true value*
		`false`	*false value*
		`nv`	*numeric value*
`nv`	::=		*numeric values*
		`0`	*zero value*
		`succ nv`	*successor value*
`T`	::=		*types*
		`Bool`	*type of booleans*
		`Nat`	*type of numbers*

Evaluation

$$\texttt{if true then } t_2 \texttt{ else } t_3 \longrightarrow t_2 \qquad \text{(E-IfTrue)}$$

$$\texttt{if false then } t_2 \texttt{ else } t_3 \longrightarrow t_3 \qquad \text{(E-IfFalse)}$$

$$\frac{t_1 \longrightarrow t_1'}{\texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3 \longrightarrow \texttt{if } t_1' \texttt{ then } t_2 \texttt{ else } t_3} \qquad \text{(E-If)}$$

$$\frac{t_1 \longrightarrow t_1'}{\texttt{succ } t_1 \longrightarrow \texttt{succ } t_1'} \qquad \text{(E-Succ)}$$

$$\texttt{pred 0} \longrightarrow \texttt{0} \qquad \text{(E-PredZero)}$$

$$\texttt{pred (succ } nv_1\texttt{)} \longrightarrow nv_1 \qquad \text{(E-PredSucc)}$$

$$\frac{t_1 \longrightarrow t_1'}{\texttt{pred } t_1 \longrightarrow \texttt{pred } t_1'} \qquad \text{(E-Pred)}$$

$$\texttt{iszero 0} \longrightarrow \texttt{true} \qquad \text{(E-IszeroZero)}$$

$$\texttt{iszero (succ } nv_1\texttt{)} \longrightarrow \texttt{false} \qquad \text{(E-IszeroSucc)}$$

$$\frac{t_1 \longrightarrow t_1'}{\texttt{iszero } t_1 \longrightarrow \texttt{iszero } t_1'} \qquad \text{(E-IsZero)}$$

continued on next page...

Typing

$$\texttt{true} : \texttt{Bool} \tag{T-True}$$

$$\texttt{false} : \texttt{Bool} \tag{T-False}$$

$$\frac{\texttt{t}_1 : \texttt{Bool} \qquad \texttt{t}_2 : \texttt{T} \qquad \texttt{t}_3 : \texttt{T}}{\texttt{if}\ \texttt{t}_1\ \texttt{then}\ \texttt{t}_2\ \texttt{else}\ \texttt{t}_3 : \texttt{T}} \tag{T-If}$$

$$\texttt{0} : \texttt{Nat} \tag{T-Zero}$$

$$\frac{\texttt{t}_1 : \texttt{Nat}}{\texttt{succ}\ \texttt{t}_1 : \texttt{Nat}} \tag{T-Succ}$$

$$\frac{\texttt{t}_1 : \texttt{Nat}}{\texttt{pred}\ \texttt{t}_1 : \texttt{Nat}} \tag{T-Pred}$$

$$\frac{\texttt{t}_1 : \texttt{Nat}}{\texttt{iszero}\ \texttt{t}_1 : \texttt{Bool}} \tag{T-IsZero}$$

Pure simply typed lambda calculus with subtyping (no records) — algorithmic rules

Syntax

$\mathtt{t}$ ::=		*terms*
	$\mathtt{x}$	*variable*
	$\lambda\mathtt{x}{:}\mathtt{T}.\mathtt{t}$	*abstraction*
	$\mathtt{t}\ \mathtt{t}$	*application*
$\mathtt{v}$ ::=		*values*
	$\lambda\mathtt{x}{:}\mathtt{T}.\mathtt{t}$	*abstraction value*
$\mathtt{T}$ ::=		*types*
	$\mathtt{Top}$	*maximum type*
	$\mathtt{T}\to\mathtt{T}$	*type of functions*
Γ ::=		*contexts*
	$\emptyset$	*empty context*
	$\Gamma, \mathtt{x}{:}\mathtt{T}$	*term variable binding*

Evaluation $\boxed{\mathtt{t} \longrightarrow \mathtt{t}'}$

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{t}_1\ \mathtt{t}_2 \longrightarrow \mathtt{t}_1'\ \mathtt{t}_2} \quad \text{(E-APP1)}$$

$$\frac{\mathtt{t}_2 \longrightarrow \mathtt{t}_2'}{\mathtt{v}_1\ \mathtt{t}_2 \longrightarrow \mathtt{v}_1\ \mathtt{t}_2'} \quad \text{(E-APP2)}$$

$$(\lambda\mathtt{x}{:}\mathtt{T}_{11}.\mathtt{t}_{12})\ \mathtt{v}_2 \longrightarrow [\mathtt{x} \mapsto \mathtt{v}_2]\mathtt{t}_{12} \quad \text{(E-APPABS)}$$

Algorithmic subtyping $\boxed{\mapsto \mathtt{S} <: \mathtt{T}}$

$$\mapsto \mathtt{S} <: \mathtt{Top} \quad \text{(SA-TOP)}$$

$$\frac{\mapsto \mathtt{T}_1 <: \mathtt{S}_1 \qquad \mapsto \mathtt{S}_2 <: \mathtt{T}_2}{\mapsto \mathtt{S}_1\to\mathtt{S}_2 <: \mathtt{T}_1\to\mathtt{T}_2} \quad \text{(SA-ARROW)}$$

Algorithmic typing $\boxed{\Gamma \mapsto \mathtt{t} : \mathtt{T}}$

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

$$\frac{\Gamma, \mathtt{x}{:}\mathtt{T}_1 \mapsto \mathtt{t}_2 : \mathtt{T}_2}{\Gamma \mapsto \lambda\mathtt{x}{:}\mathtt{T}_1.\mathtt{t}_2 : \mathtt{T}_1\to\mathtt{T}_2} \quad \text{(TA-ABS)}$$

$$\frac{\begin{array}{cc}\Gamma \mapsto \mathtt{t}_1 : \mathtt{T}_1 & \mathtt{T}_1 = \mathtt{T}_{11}\to\mathtt{T}_{12} \\ \Gamma \mapsto \mathtt{t}_2 : \mathtt{T}_2 & \mapsto \mathtt{T}_2 <: \mathtt{T}_{11}\end{array}}{\Gamma \mapsto \mathtt{t}_1\ \mathtt{t}_2 : \mathtt{T}_{12}} \quad \text{(TA-APP)}$$

Simply typed lambda calculus with subtyping
(and records, references, recursion, booleans, numbers)

Syntax

`t ::=`	*terms*
`x`	*variable*
`λx:T.t`	*abstraction*
`t t`	*application*
$\{l_i = t_i\ ^{i\in 1..n}\}$	*record*
`t.l`	*projection*
`unit`	*constant* `unit`
`ref t`	*reference creation*
`!t`	*dereference*
`t:=t`	*assignment*
`l`	*store location*
`true`	*constant true*
`false`	*constant false*
`if t then t else t`	*conditional*
`0`	*constant zero*
`succ t`	*successor*
`pred t`	*predecessor*
`iszero t`	*zero test*
`let x=t in t`	*let binding*
`fix t`	*fixed point of* `t`
`v ::=`	*values*
`λx:T.t`	*abstraction value*
$\{l_i = v_i\ ^{i\in 1..n}\}$	*record value*
`unit`	*constant* `unit`
`l`	*store location*
`true`	*true value*
`false`	*false value*
`nv`	*numeric value*
`T ::=`	*types*
$\{l_i : T_i\ ^{i\in 1..n}\}$	*type of records*
`Top`	*maximum type*
`T→T`	*type of functions*
`Unit`	*unit type*
`Ref T`	*type of reference cells*
`Bool`	*type of booleans*
`Nat`	*type of natural numbers*
`Γ ::=`	*contexts*
`∅`	*empty context*
`Γ, x:T`	*term variable binding*
`μ ::=`	*stores*
`∅`	*empty store*

		$\mu, \mathtt{l} = \mathtt{v}$	*location binding*
Σ	::=		*store typings*
		$\emptyset$	*empty store typing*
		$\Sigma, \mathtt{l}\!:\!\mathtt{T}$	*location typing*
$\mathtt{nv}$	::=		*numeric values*
		$\mathtt{0}$	*zero value*
		$\mathtt{succ}\ \mathtt{nv}$	*successor value*

Evaluation $\boxed{\mathtt{t} \mid \mu \longrightarrow \mathtt{t}' \mid \mu'}$

$$\frac{\mathtt{t}_1 \mid \mu \longrightarrow \mathtt{t}_1' \mid \mu'}{\mathtt{t}_1\ \mathtt{t}_2 \mid \mu \longrightarrow \mathtt{t}_1'\ \mathtt{t}_2 \mid \mu'} \qquad \text{(E-App1)}$$

$$\frac{\mathtt{t}_2 \mid \mu \longrightarrow \mathtt{t}_2' \mid \mu'}{\mathtt{v}_1\ \mathtt{t}_2 \mid \mu \longrightarrow \mathtt{v}_1\ \mathtt{t}_2' \mid \mu'} \qquad \text{(E-App2)}$$

$$(\lambda \mathtt{x}\!:\!\mathtt{T}_{11}.\mathtt{t}_{12})\ \mathtt{v}_2 \mid \mu \longrightarrow [\mathtt{x} \mapsto \mathtt{v}_2]\mathtt{t}_{12} \mid \mu \qquad \text{(E-AppAbs)}$$

$$\{\mathtt{l}_i\mathtt{=}\mathtt{v}_i\ ^{i \in 1..n}\}.\mathtt{l}_j \mid \mu \longrightarrow \mathtt{v}_j \mid \mu \qquad \text{(E-ProjRcd)}$$

$$\frac{\mathtt{t}_1 \mid \mu \longrightarrow \mathtt{t}_1' \mid \mu'}{\mathtt{t}_1.\mathtt{l} \mid \mu \longrightarrow \mathtt{t}_1'.\mathtt{l} \mid \mu'} \qquad \text{(E-Proj)}$$

$$\frac{\mathtt{t}_j \mid \mu \longrightarrow \mathtt{t}_j' \mid \mu'}{\begin{array}{c}\{\mathtt{l}_i\mathtt{=}\mathtt{v}_i\ ^{i \in 1..j-1}, \mathtt{l}_j\mathtt{=}\mathtt{t}_j, \mathtt{l}_k\mathtt{=}\mathtt{t}_k\ ^{k \in j+1..n}\} \mid \mu \\ \longrightarrow \{\mathtt{l}_i\mathtt{=}\mathtt{v}_i\ ^{i \in 1..j-1}, \mathtt{l}_j\mathtt{=}\mathtt{t}_j', \mathtt{l}_k\mathtt{=}\mathtt{t}_k\ ^{k \in j+1..n}\} \mid \mu'\end{array}} \qquad \text{(E-Rcd)}$$

$$\frac{\mathtt{l} \notin dom(\mu)}{\mathtt{ref}\ \mathtt{v}_1 \mid \mu \longrightarrow \mathtt{l} \mid (\mu, \mathtt{l} \mapsto \mathtt{v}_1)} \qquad \text{(E-RefV)}$$

$$\frac{\mathtt{t}_1 \mid \mu \longrightarrow \mathtt{t}_1' \mid \mu'}{\mathtt{ref}\ \mathtt{t}_1 \mid \mu \longrightarrow \mathtt{ref}\ \mathtt{t}_1' \mid \mu'} \qquad \text{(E-Ref)}$$

$$\frac{\mu(\mathtt{l}) = \mathtt{v}}{!\mathtt{l} \mid \mu \longrightarrow \mathtt{v} \mid \mu} \qquad \text{(E-DerefLoc)}$$

$$\frac{\mathtt{t}_1 \mid \mu \longrightarrow \mathtt{t}_1' \mid \mu'}{!\mathtt{t}_1 \mid \mu \longrightarrow\ !\mathtt{t}_1' \mid \mu'} \qquad \text{(E-Deref)}$$

$$\mathtt{l}\mathtt{:=}\mathtt{v}_2 \mid \mu \longrightarrow \mathtt{unit} \mid [\mathtt{l} \mapsto \mathtt{v}_2]\mu \qquad \text{(E-Assign)}$$

$$\frac{\mathtt{t}_1 \mid \mu \longrightarrow \mathtt{t}_1' \mid \mu'}{\mathtt{t}_1\mathtt{:=}\mathtt{t}_2 \mid \mu \longrightarrow \mathtt{t}_1'\mathtt{:=}\mathtt{t}_2 \mid \mu'} \qquad \text{(E-Assign1)}$$

$$\frac{\mathtt{t}_2 \mid \mu \longrightarrow \mathtt{t}_2' \mid \mu'}{\mathtt{v}_1\mathtt{:=}\mathtt{t}_2 \mid \mu \longrightarrow \mathtt{v}_1\mathtt{:=}\mathtt{t}_2' \mid \mu'} \qquad \text{(E-Assign2)}$$

$$\mathtt{if}\ \mathtt{true}\ \mathtt{then}\ \mathtt{t}_2\ \mathtt{else}\ \mathtt{t}_3 \mid \mu \longrightarrow \mathtt{t}_2 \mid \mu \qquad \text{(E-IfTrue)}$$

$$\mathtt{if}\ \mathtt{false}\ \mathtt{then}\ \mathtt{t}_2\ \mathtt{else}\ \mathtt{t}_3 \mid \mu \longrightarrow \mathtt{t}_3 \mid \mu \qquad \text{(E-IfFalse)}$$

$$\frac{\mathtt{t}_1 \mid \mu \longrightarrow \mathtt{t}_1' \mid \mu'}{\mathtt{if}\ \mathtt{t}_1\ \mathtt{then}\ \mathtt{t}_2\ \mathtt{else}\ \mathtt{t}_3 \mid \mu \longrightarrow \mathtt{if}\ \mathtt{t}_1'\ \mathtt{then}\ \mathtt{t}_2\ \mathtt{else}\ \mathtt{t}_3 \mid \mu'} \qquad \text{(E-If)}$$

$$\frac{t_1 | \mu \longrightarrow t_1' | \mu'}{\texttt{succ}\ t_1 | \mu \longrightarrow \texttt{succ}\ t_1' | \mu'} \qquad \text{(E-Succ)}$$

$$\texttt{pred}\ 0 | \mu \longrightarrow 0 | \mu \qquad \text{(E-PredZero)}$$

$$\texttt{pred}\ (\texttt{succ}\ nv_1) | \mu \longrightarrow nv_1 | \mu \qquad \text{(E-PredSucc)}$$

$$\frac{t_1 | \mu \longrightarrow t_1' | \mu'}{\texttt{pred}\ t_1 | \mu \longrightarrow \texttt{pred}\ t_1' | \mu} \qquad \text{(E-Pred)}$$

$$\texttt{iszero}\ 0 | \mu \longrightarrow \texttt{true} | \mu \qquad \text{(E-IszeroZero)}$$

$$\texttt{iszero}\ (\texttt{succ}\ nv_1) | \mu \longrightarrow \texttt{false} | \mu \qquad \text{(E-IszeroSucc)}$$

$$\frac{t_1 | \mu \longrightarrow t_1' | \mu'}{\texttt{iszero}\ t_1 | \mu \longrightarrow \texttt{iszero}\ t_1' | \mu'} \qquad \text{(E-IsZero)}$$

$$\texttt{let}\ x{=}v_1\ \texttt{in}\ t_2 | \mu \longrightarrow [x \mapsto v_1]t_2 | \mu \qquad \text{(E-LetV)}$$

$$\frac{t_1 | \mu \longrightarrow t_1' | \mu'}{\texttt{let}\ x{=}t_1\ \texttt{in}\ t_2 | \mu \longrightarrow \texttt{let}\ x{=}t_1'\ \texttt{in}\ t_2 | \mu'} \qquad \text{(E-Let)}$$

$$\texttt{fix}\ (\lambda x{:}T_1.t_2) | \mu \longrightarrow [x \mapsto (\texttt{fix}\ (\lambda x{:}T_1.t_2))]t_2 | \mu \qquad \text{(E-FixBeta)}$$

$$\frac{t_1 | \mu \longrightarrow t_1' | \mu'}{\texttt{fix}\ t_1 | \mu \longrightarrow \texttt{fix}\ t_1' | \mu} \qquad \text{(E-Fix)}$$

Subtyping $\boxed{S <: T}$

$$S <: S \qquad \text{(S-Refl)}$$

$$\frac{S <: U \qquad U <: T}{S <: T} \qquad \text{(S-Trans)}$$

$$S <: \texttt{Top} \qquad \text{(S-Top)}$$

$$\frac{T_1 <: S_1 \qquad S_2 <: T_2}{S_1 {\to} S_2 <: T_1 {\to} T_2} \qquad \text{(S-Arrow)}$$

$$\{l_i{:}T_i\ ^{i \in 1..n+k}\} <: \{l_i{:}T_i\ ^{i \in 1..n}\} \qquad \text{(S-RcdWidth)}$$

$$\frac{\text{for each } i \quad S_i <: T_i}{\{l_i{:}S_i\ ^{i \in 1..n}\} <: \{l_i{:}T_i\ ^{i \in 1..n}\}} \qquad \text{(S-RcdDepth)}$$

$$\frac{\{k_j{:}S_j\ ^{j \in 1..n}\} \text{ is a permutation of } \{l_i{:}T_i\ ^{i \in 1..n}\}}{\{k_j{:}S_j\ ^{j \in 1..n}\} <: \{l_i{:}T_i\ ^{i \in 1..n}\}} \qquad \text{(S-RcdPerm)}$$

$$\frac{S_1 <: T_1 \qquad T_1 <: S_1}{\texttt{Ref}\ S_1 <: \texttt{Ref}\ T_1} \qquad \text{(S-Ref)}$$

Typing $\boxed{\Gamma | \Sigma \vdash \mathtt{t} : \mathtt{T}}$

$$\frac{\text{for each } i \quad \Gamma | \Sigma \vdash \mathtt{t}_i : \mathtt{T}_i}{\Gamma | \Sigma \vdash \{\mathtt{l}_i\mathtt{=t}_i \;^{i\in 1..n}\} : \{\mathtt{l}_i\mathtt{:T}_i \;^{i\in 1..n}\}} \qquad \text{(T-Rcd)}$$

$$\frac{\Gamma | \Sigma \vdash \mathtt{t}_1 : \{\mathtt{l}_i\mathtt{:T}_i \;^{i\in 1..n}\}}{\Gamma | \Sigma \vdash \mathtt{t}_1\mathtt{.l}_j : \mathtt{T}_j} \qquad \text{(T-Proj)}$$

$$\frac{\mathtt{x}\mathtt{:T} \in \Gamma}{\Gamma | \Sigma \vdash \mathtt{x} : \mathtt{T}} \qquad \text{(T-Var)}$$

$$\frac{\Gamma, \mathtt{x}\mathtt{:T}_1 | \Sigma \vdash \mathtt{t}_2 : \mathtt{T}_2}{\Gamma | \Sigma \vdash \lambda\mathtt{x}\mathtt{:T}_1\mathtt{.t}_2 : \mathtt{T}_1 \to \mathtt{T}_2} \qquad \text{(T-Abs)}$$

$$\frac{\Gamma | \Sigma \vdash \mathtt{t}_1 : \mathtt{T}_{11} \to \mathtt{T}_{12} \qquad \Gamma | \Sigma \vdash \mathtt{t}_2 : \mathtt{T}_{11}}{\Gamma | \Sigma \vdash \mathtt{t}_1\;\mathtt{t}_2 : \mathtt{T}_{12}} \qquad \text{(T-App)}$$

$$\frac{\Gamma | \Sigma \vdash \mathtt{t} : \mathtt{S} \qquad \mathtt{S} <: \mathtt{T}}{\Gamma | \Sigma \vdash \mathtt{t} : \mathtt{T}} \qquad \text{(T-Sub)}$$

$$\Gamma | \Sigma \vdash \mathtt{unit} : \mathtt{Unit} \qquad \text{(T-Unit)}$$

$$\frac{\Sigma(l) = \mathtt{T}_1}{\Gamma | \Sigma \vdash l : \mathtt{Ref}\;\mathtt{T}_1} \qquad \text{(T-Loc)}$$

$$\frac{\Gamma | \Sigma \vdash \mathtt{t}_1 : \mathtt{T}_1}{\Gamma | \Sigma \vdash \mathtt{ref}\;\mathtt{t}_1 : \mathtt{Ref}\;\mathtt{T}_1} \qquad \text{(T-Ref)}$$

$$\frac{\Gamma | \Sigma \vdash \mathtt{t}_1 : \mathtt{Ref}\;\mathtt{T}_{11}}{\Gamma | \Sigma \vdash \mathtt{!t}_1 : \mathtt{T}_{11}} \qquad \text{(T-Deref)}$$

$$\frac{\Gamma | \Sigma \vdash \mathtt{t}_1 : \mathtt{Ref}\;\mathtt{T}_{11} \qquad \Gamma | \Sigma \vdash \mathtt{t}_2 : \mathtt{T}_{11}}{\Gamma | \Sigma \vdash \mathtt{t}_1\mathtt{:=t}_2 : \mathtt{Unit}} \qquad \text{(T-Assign)}$$

$$\Gamma | \Sigma \vdash \mathtt{true} : \mathtt{Bool} \qquad \text{(T-True)}$$

$$\Gamma | \Sigma \vdash \mathtt{false} : \mathtt{Bool} \qquad \text{(T-False)}$$

$$\frac{\Gamma | \Sigma \vdash \mathtt{t}_1 : \mathtt{Bool} \qquad \Gamma | \Sigma \vdash \mathtt{t}_2 : \mathtt{T} \qquad \Gamma | \Sigma \vdash \mathtt{t}_3 : \mathtt{T}}{\Gamma | \Sigma \vdash \mathtt{if}\;\mathtt{t}_1\;\mathtt{then}\;\mathtt{t}_2\;\mathtt{else}\;\mathtt{t}_3 : \mathtt{T}} \qquad \text{(T-If)}$$

$$\Gamma | \Sigma \vdash \mathtt{0} : \mathtt{Nat} \qquad \text{(T-Zero)}$$

$$\frac{\Gamma | \Sigma \vdash \mathtt{t}_1 : \mathtt{Nat}}{\Gamma | \Sigma \vdash \mathtt{succ}\;\mathtt{t}_1 : \mathtt{Nat}} \qquad \text{(T-Succ)}$$

$$\frac{\Gamma | \Sigma \vdash \mathtt{t}_1 : \mathtt{Nat}}{\Gamma | \Sigma \vdash \mathtt{pred}\;\mathtt{t}_1 : \mathtt{Nat}} \qquad \text{(T-Pred)}$$

$$\frac{\Gamma | \Sigma \vdash \mathtt{t}_1 : \mathtt{Nat}}{\Gamma | \Sigma \vdash \mathtt{iszero}\;\mathtt{t}_1 : \mathtt{Bool}} \qquad \text{(T-IsZero)}$$

$$\frac{\Gamma | \Sigma \vdash \mathtt{t}_1 : \mathtt{T}_1 \qquad \Gamma, \mathtt{x}\mathtt{:T}_1 | \Sigma \vdash \mathtt{t}_2 : \mathtt{T}_2}{\Gamma | \Sigma \vdash \mathtt{let}\;\mathtt{x=t}_1\;\mathtt{in}\;\mathtt{t}_2 : \mathtt{T}_2} \qquad \text{(T-Let)}$$

$$\frac{\Gamma | \Sigma \vdash \mathtt{t}_1 : \mathtt{T}_1 \to \mathtt{T}_1}{\Gamma | \Sigma \vdash \mathtt{fix}\;\mathtt{t}_1 : \mathtt{T}_1} \qquad \text{(T-Fix)}$$

Featherweight Java

Syntax

CL	::=	$\texttt{class C extends C } \{\overline{\texttt{C}}\ \overline{\texttt{f}}\texttt{; K } \overline{\texttt{M}}\}$	*class declarations*
K	::=	$\texttt{C}(\overline{\texttt{C}}\ \overline{\texttt{f}})\ \{\texttt{super}(\overline{\texttt{f}})\texttt{; this.}\overline{\texttt{f}}\texttt{=}\overline{\texttt{f}}\texttt{;}\}$	*constructor declarations*
M	::=	$\texttt{C m}(\overline{\texttt{C}}\ \overline{\texttt{x}})\ \{\texttt{return t;}\}$	*method declarations*
t	::=		*terms*
		x	*variable*
		t.f	*field access*
		$\texttt{t.m}(\overline{\texttt{t}})$	*method invocation*
		$\texttt{new C}(\overline{\texttt{t}})$	*object creation*
		(C) t	*cast*
v	::=		*values*
		$\texttt{new C}(\overline{\texttt{v}})$	*object creation*

Subtyping $\boxed{\texttt{C<:D}}$

$$\texttt{C} <: \texttt{C}$$

$$\frac{\texttt{C} <: \texttt{D} \qquad \texttt{D} <: \texttt{E}}{\texttt{C} <: \texttt{E}}$$

$$\frac{CT(\texttt{C}) = \texttt{class C extends D } \{\ldots\}}{\texttt{C} <: \texttt{D}}$$

Field lookup $\boxed{\mathit{fields}(\texttt{C}) = \overline{\texttt{C}}\ \overline{\texttt{f}}}$

$$\mathit{fields}(\texttt{Object}) = \bullet$$

$$\frac{CT(\texttt{C}) = \texttt{class C extends D } \{\overline{\texttt{C}}\ \overline{\texttt{f}}\texttt{; K } \overline{\texttt{M}}\} \qquad \mathit{fields}(\texttt{D}) = \overline{\texttt{D}}\ \overline{\texttt{g}}}{\mathit{fields}(\texttt{C}) = \overline{\texttt{D}}\ \overline{\texttt{g}}, \overline{\texttt{C}}\ \overline{\texttt{f}}}$$

Method type lookup $\boxed{\mathit{mtype}(\texttt{m}, \texttt{C}) = \overline{\texttt{C}} \to \texttt{C}}$

$$\frac{CT(\texttt{C}) = \texttt{class C extends D } \{\overline{\texttt{C}}\ \overline{\texttt{f}}\texttt{; K } \overline{\texttt{M}}\} \qquad \texttt{B m } (\overline{\texttt{B}}\ \overline{\texttt{x}})\ \{\texttt{return t;}\} \in \overline{\texttt{M}}}{\mathit{mtype}(\texttt{m}, \texttt{C}) = \overline{\texttt{B}} \to \texttt{B}}$$

$$\frac{CT(\texttt{C}) = \texttt{class C extends D } \{\overline{\texttt{C}}\ \overline{\texttt{f}}\texttt{; K } \overline{\texttt{M}}\} \qquad \texttt{m} \text{ is not defined in } \overline{\texttt{M}}}{\mathit{mtype}(\texttt{m}, \texttt{C}) = \mathit{mtype}(\texttt{m}, \texttt{D})}$$

Method body lookup $\boxed{\mathit{mbody}(\texttt{m}, \texttt{C}) = (\overline{\texttt{x}}, \texttt{t})}$

$$\frac{CT(\texttt{C}) = \texttt{class C extends D } \{\overline{\texttt{C}}\ \overline{\texttt{f}}\texttt{; K } \overline{\texttt{M}}\} \quad \texttt{B m } (\overline{\texttt{B}}\ \overline{\texttt{x}})\ \{\texttt{return t;}\} \in \overline{\texttt{M}}}{mbody(\texttt{m}, \texttt{C}) = (\overline{\texttt{x}}, \texttt{t})}$$

$$\frac{CT(\texttt{C}) = \texttt{class C extends D } \{\overline{\texttt{C}}\ \overline{\texttt{f}}\texttt{; K } \overline{\texttt{M}}\} \quad \texttt{m} \text{ is not defined in } \overline{\texttt{M}}}{mbody(\texttt{m}, \texttt{C}) = mbody(\texttt{m}, \texttt{D})}$$

Valid method overriding $\boxed{override(\texttt{m}, \texttt{D}, \overline{\texttt{C}} \to \texttt{C}_0)}$

$$\frac{mtype(\texttt{m}, \texttt{D}) = \overline{\texttt{D}} \to \texttt{D}_0 \text{ implies } \overline{\texttt{C}} = \overline{\texttt{D}} \text{ and } \texttt{C}_0 = \texttt{D}_0}{override(\texttt{m}, \texttt{D}, \overline{\texttt{C}} \to \texttt{C}_0)}$$

Evaluation $\boxed{\texttt{t} \longrightarrow \texttt{t}'}$

$$\frac{fields(\texttt{C}) = \overline{\texttt{C}}\ \overline{\texttt{f}}}{(\texttt{new C}(\overline{\texttt{v}})).\texttt{f}_i \longrightarrow \texttt{v}_i} \quad \text{(E-ProjNew)}$$

$$\frac{mbody(\texttt{m}, \texttt{C}) = (\overline{\texttt{x}}, \texttt{t}_0)}{(\texttt{new C}(\overline{\texttt{v}})).\texttt{m}(\overline{\texttt{u}}) \longrightarrow [\overline{\texttt{x}} \mapsto \overline{\texttt{u}}, \texttt{this} \mapsto \texttt{new C}(\overline{\texttt{v}})]\texttt{t}_0} \quad \text{(E-InvkNew)}$$

$$\frac{\texttt{C <: D}}{(\texttt{D})(\texttt{new C}(\overline{\texttt{v}})) \longrightarrow \texttt{new C}(\overline{\texttt{v}})} \quad \text{(E-CastNew)}$$

$$\frac{\texttt{t}_0 \longrightarrow \texttt{t}_0'}{\texttt{t}_0.\texttt{f} \longrightarrow \texttt{t}_0'.\texttt{f}} \quad \text{(E-Field)}$$

$$\frac{\texttt{t}_0 \longrightarrow \texttt{t}_0'}{\texttt{t}_0.\texttt{m}(\overline{\texttt{t}}) \longrightarrow \texttt{t}_0'.\texttt{m}(\overline{\texttt{t}})} \quad \text{(E-Invk-Recv)}$$

$$\frac{\texttt{t}_i \longrightarrow \texttt{t}_i'}{\texttt{v}_0.\texttt{m}(\overline{\texttt{v}}, \texttt{t}_i, \overline{\texttt{t}}) \longrightarrow \texttt{v}_0.\texttt{m}(\overline{\texttt{v}}, \texttt{t}_i', \overline{\texttt{t}})} \quad \text{(E-Invk-Arg)}$$

$$\frac{\texttt{t}_i \longrightarrow \texttt{t}_i'}{\texttt{new C}(\overline{\texttt{v}}, \texttt{t}_i, \overline{\texttt{t}}) \longrightarrow \texttt{new C}(\overline{\texttt{v}}, \texttt{t}_i', \overline{\texttt{t}})} \quad \text{(E-New-Arg)}$$

$$\frac{\texttt{t}_0 \longrightarrow \texttt{t}_0'}{(\texttt{C})\texttt{t}_0 \longrightarrow (\texttt{C})\texttt{t}_0'} \quad \text{(E-Cast)}$$

Term typing $\boxed{\Gamma \vdash \texttt{t : C}}$

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

$$\frac{\Gamma \vdash \texttt{t}_0 : \texttt{C}_0 \qquad fields(\texttt{C}_0) = \overline{\texttt{C}}\ \overline{\texttt{f}}}{\Gamma \vdash \texttt{t}_0.\texttt{f}_i : \texttt{C}_i} \quad \text{(T-Field)}$$

$$\frac{\Gamma \vdash \texttt{t}_0 : \texttt{C}_0 \quad mtype(\texttt{m}, \texttt{C}_0) = \overline{\texttt{D}} \to \texttt{C} \quad \Gamma \vdash \overline{\texttt{t}} : \overline{\texttt{C}} \qquad \overline{\texttt{C}} <: \overline{\texttt{D}}}{\Gamma \vdash \texttt{t}_0.\texttt{m}(\overline{\texttt{t}}) : \texttt{C}} \quad \text{(T-Invk)}$$

$$\frac{\begin{array}{c} \mathit{fields}(\mathtt{C}) = \overline{\mathtt{D}}\ \overline{\mathtt{f}} \\ \Gamma \vdash \overline{\mathtt{t}} : \overline{\mathtt{C}} \qquad \overline{\mathtt{C}} <: \overline{\mathtt{D}} \end{array}}{\Gamma \vdash \mathtt{new}\ \mathtt{C}(\overline{\mathtt{t}}) : \mathtt{C}} \qquad \text{(T-New)}$$

$$\frac{\Gamma \vdash \mathtt{t}_0 : \mathtt{D} \qquad \mathtt{D} <: \mathtt{C}}{\Gamma \vdash (\mathtt{C})\mathtt{t}_0 : \mathtt{C}} \qquad \text{(T-UCast)}$$

$$\frac{\Gamma \vdash \mathtt{t}_0 : \mathtt{D} \qquad \mathtt{C} <: \mathtt{D} \qquad \mathtt{C} \neq \mathtt{D}}{\Gamma \vdash (\mathtt{C})\mathtt{t}_0 : \mathtt{C}} \qquad \text{(T-DCast)}$$

$$\frac{\begin{array}{c} \Gamma \vdash \mathtt{t}_0 : \mathtt{D} \qquad \mathtt{C} \not<: \mathtt{D} \qquad \mathtt{D} \not<: \mathtt{C} \\ \mathit{stupid\ warning} \end{array}}{\Gamma \vdash (\mathtt{C})\mathtt{t}_0 : \mathtt{C}} \qquad \text{(T-SCast)}$$

Method typing $\boxed{\mathtt{M}\ \mathtt{OK\ in}\ \mathtt{C}}$

$$\frac{\begin{array}{c} \overline{\mathtt{x}} : \overline{\mathtt{C}}, \mathtt{this} : \mathtt{C} \vdash \mathtt{t}_0 : \mathtt{E}_0 \qquad \mathtt{E}_0 <: \mathtt{C}_0 \\ \mathit{CT}(\mathtt{C}) = \mathtt{class}\ \mathtt{C}\ \mathtt{extends}\ \mathtt{D}\ \{\ldots\} \\ \mathit{override}(\mathtt{m}, \mathtt{D}, \overline{\mathtt{C}} \to \mathtt{C}_0) \end{array}}{\mathtt{C}_0\ \mathtt{m}\ (\overline{\mathtt{C}}\ \overline{\mathtt{x}})\ \{\mathtt{return}\ \mathtt{t}_0;\}\ \mathtt{OK\ in}\ \mathtt{C}}$$

Class typing $\boxed{\mathtt{C}\ \mathtt{OK}}$

$$\frac{\begin{array}{c} \mathtt{K} = \mathtt{C}(\overline{\mathtt{D}}\ \overline{\mathtt{g}}, \overline{\mathtt{C}}\ \overline{\mathtt{f}}) \qquad \{\mathtt{super}(\overline{\mathtt{g}});\ \mathtt{this}.\overline{\mathtt{f}} = \overline{\mathtt{f}};\} \\ \mathit{fields}(\mathtt{D}) = \overline{\mathtt{D}}\ \overline{\mathtt{g}} \qquad \overline{\mathtt{M}}\ \mathtt{OK\ in}\ \mathtt{C} \end{array}}{\mathtt{class}\ \mathtt{C}\ \mathtt{extends}\ \mathtt{D}\ \{\overline{\mathtt{C}}\ \overline{\mathtt{f}};\ \mathtt{K}\ \overline{\mathtt{M}}\}\ \mathtt{OK}}$$