CIS 500 — Software Foundations

Final Exam

Answer key

December 14, 2005

True/False questions

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

1. (9 points)

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

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

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

(e) $\boxed{\text{T}}$ 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) $\boxed{\text{T}}$ 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 $\boxed{\text{F}}$ 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 $\boxed{\text{F}}$ 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 $\boxed{\text{F}}$ FJ is specified with a large-step operational semantics.

Grading scheme: Binary. 1 pt each.

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:

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

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

(a) Draw a derivation for $(* \wedge \Box) \sim (* \wedge \Box)$

Answer:
$$\frac{\dfrac{}{* \sim \Box}\ \textsc{Ax1} \qquad \dfrac{}{\Box \sim *}\ \textsc{Ax2}}{(* \wedge \Box) \sim (* \wedge \Box)}\ \textsc{Amp1}$$

Grading scheme: 2 points. -1 for "minor" errors.

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

Answer:
$$\frac{\dfrac{\dfrac{}{\Box \sim *}\ \textsc{Ax2} \qquad \dfrac{}{* \sim \Box}\ \textsc{Ax1}}{(\Box \wedge *) \sim (\Box \wedge *)}\ \textsc{Amp1}}{(* \wedge \Box) \sim (* \wedge \Box)}\ \textsc{Amp2}$$

Grading scheme: 2 points. -1 for "minor" errors.

(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?

Answer: AMP2

Grading scheme: 1 point.

(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).

Answer: YES. Proof by induction on the structure of x.

- *Case* $x = *$*: Choose* $y = \square$*, and derive* $x \sim y$ *by* AX1.
- *Case* $x = \square$*: Choose* $y = *$*, and derive* $x \sim y$ *by* AX2.
- *Case* $x = x_1 \wedge x_2$ *for some* x_1 *and* x_2*: By the induction hypothesis, there exists* y_1 *and* y_2 *such that* $x_1 \sim y_2$ *and* $x_2 \sim y_1$*. Choose* $y = y_1 \wedge y_2$*, and derive* $x \sim y$ *(from* $x_1 \sim y_2$ *and* $x_2 \sim y_1$*) by* AMP1.

Grading scheme: 9 points total for this problem.

- *1 point for saying YES.*
- *-1 for misc. confusions in a basically correct proof. (forgetting "choose* $y = \ldots$*", or for renaming* x *to* t *without saying so, etc.)*
- *-3 for missing "there exists* y_1 *and* $y_2 \ldots$*"*
- *-4 for induction on the wrong things*
- *-5 for no induction*
- *-2 for missing IH*
- *-2 for mangling the* $\wedge$ *case*

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

   Answer: (iv)

   (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

   Answer: iii

   (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

   Answer: v

   (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

   Answer: i

   Grading scheme: Binary. 3 points per part.

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

   Answer: (A)

   Grading scheme: Binary.

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

   Answer: (A)

   Grading scheme: Binary.

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))
```

Answer:

- *TmIf — last branch should be* `TmIf(eval1 t1, t2, t3)`
- *TmSucc — t1 should be* `eval1 t1`*.*
- *TmPred —* `TmSucc` *should check to see if* `nv1` *is a numeric val (*`when isnumericval nv1` *guard...)*
- *TmIsZero —* `TmFalse` *should be* `TmTrue` *in the* `TmZero` *case.*

Grading scheme: 3 points per branch.

- *-1 per incorrect line, if the correction was made (i.e., you also turned a correct line into an incorrect one)*
- *-3 if you missed the fix for a branch entirely*
- *-1 for trivial errors*

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}`

   Answer: `Top`

   (b) `λy:{}→Ref Top. λx:Top. y x`

   Answer: NONE

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

   Answer: `(Ref Nat) → {y:{b:Nat}}`

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

   Answer: `Top`

   Grading scheme: Binary. 3 points per part.

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

$$\texttt{Top} \rightarrow \texttt{Top} <: \{\}$$

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

Answer: Progress remains true: there is no way to do anything with a value of type `{}`, hence no way to test whether it is really a record (and get stuck if otherwise).

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

$$\texttt{Top} <: \texttt{Top} \rightarrow \texttt{Top}$$

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

Answer: Progress fails. For example, the term `{} {}` is well typed, but stuck.

Grading scheme:

- *2 points for "true" or "false"*
- *-1 for substantially correct, but difficult to follow.*
- *-2 for confused but mentioning a correct keyword or two*
- *-3 for better than nothing.*

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:

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

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

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

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 references types. What should the preconditions of this rule be?

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

Answer: $S_1 <: S_2$ $\quad$ $T_2 <: T_1$

Grading scheme:

- *-2 for including an extra premise (that was at least well-formed)*
- *-3 for getting one of the premises backwards.*
- *No credit for getting both premises backwards, or for premises that were ill-formed.*

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

    Why not? Justify your answer.

    Answer: It's not true because of the algorithmic subtyping in FJ.

    For example, $\vdash$ `(Object)new C() : Object` *and* `(Object)new C()` $\longrightarrow$ `new C()` *but* $\not\vdash$ `new C() : Object`.

    Grading scheme: We were looking for answers that explain that algorithmic subtyping is to blame, or provide a counterexample.

    Incorrect answers included just stating the right preservation lemma (that doesn't give an explanation why this one was false!), blaming casting (although this counterexample uses casting, there are many that do not), or giving a counterexample in the wrong language.

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:

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

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

Now, given the following lemma:

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

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

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

This lemma is proved by induction on the typing derivation $\Gamma, \overline{\texttt{x}}:\overline{\texttt{B}} \vdash \texttt{t} : \texttt{D}$. **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).

Answer: In this case, we know that:

$t = t_0.m(\overline{t})$

$\Gamma, \overline{x}:\overline{B} \vdash t_0 : D_0$

$\Gamma, \overline{x}:\overline{B} \vdash \overline{t} : \overline{D}$

$mtype(m, D_0) = \overline{E} \to D,$

$\overline{D} <: \overline{E}.$

By induction, we know that $\Gamma \vdash [\overline{x} \mapsto \overline{s}]t_0 : C_0$ *where* $C_0 <: D_0$ *and* $\Gamma \vdash [\overline{x} \mapsto \overline{s}]\overline{t} : \overline{C}$ *where* $\overline{C} <: \overline{D}$.

By Override Lemma, $mtype(m, C_0) = \overline{F} \to C$ *where* $C <: D$ *and* $\overline{E} <: \overline{F}$.

By transitivity (twice times!), $\overline{C} <: \overline{F}$

By T-INVK $\Gamma \vdash [\overline{x} \mapsto \overline{s}](t_0.m(\overline{t})) : C$ *and* $C <: D$.

Grading scheme: Strict. 3 points per proof step (2 uses of induction, using the override lemma correctly, using transitivity correctly, and using T-Invk correctly). Merely stating true facts (like the premises of T-Invk or the override lemma without doing these steps received no credit. Little to no credit for doing these proof steps incorrectly.

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    =  { m  : Object → A, n: Unit → Object}

aClass = λr:RepA. λthis:Unit → A. λ_:Unit.
   { m = λx:Object.  (this unit).n unit ,
     n = λ_:Unit.  (this unit).m (!(r.f1))   }
bClass = λr:RepB. λthis:Unit → B. λ_:Unit.
   let super = aClass r this unit in
   { m = λx:A. !(r.f2),
     n = λ_:Unit. super.n unit   }
newA = λf1val:Object.
    let r = { f1 = ref f1val } in
    fix (aClass r) unit
newB = λf1val:Object. λf2val:A.
    let r = { f1 = ref f1val, f2 = ref f2val } in
    fix (bClass r) unit
```

Grading scheme: 2 points per blank. 1 point for "minor" errors.

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`
Answer: diverges

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

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

Answer: w

Grading scheme: Binary. 2 points per part.

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*
	$\mathtt{t}\ \mathtt{t}$	*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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Syntax

$\mathtt{t}$ ::=		*terms*
	$\mathtt{x}$	*variable*
	$\lambda\mathtt{x}\mathtt{:}\mathtt{T}\mathtt{.}\mathtt{t}$	*abstraction*
	$\mathtt{t}\ \mathtt{t}$	*application*
	$\{\mathtt{l}_i\mathtt{=}\mathtt{t}_i{}^{i\in 1..n}\}$	*record*
	$\mathtt{t.l}$	*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`
$\mathtt{v}$ ::=		*values*
	$\lambda\mathtt{x}\mathtt{:}\mathtt{T}\mathtt{.}\mathtt{t}$	*abstraction value*
	$\{\mathtt{l}_i\mathtt{=}\mathtt{v}_i{}^{i\in 1..n}\}$	*record value*
	`unit`	*constant* `unit`
	l	*store location*
	`true`	*true value*
	`false`	*false value*
	`nv`	*numeric value*
$\mathtt{T}$ ::=		*types*
	$\{\mathtt{l}_i\mathtt{:}\mathtt{T}_i{}^{i\in 1..n}\}$	*type of records*
	`Top`	*maximum type*
	$\mathtt{T}\to\mathtt{T}$	*type of functions*
	`Unit`	*unit type*
	`Ref T`	*type of reference cells*
	`Bool`	*type of booleans*
	`Nat`	*type of natural numbers*
Γ ::=		*contexts*
	$\emptyset$	*empty context*
	$\Gamma, \mathtt{x}\mathtt{:}\mathtt{T}$	*term variable binding*
μ ::=		*stores*
	$\emptyset$	*empty store*

$\mu, l = v$ *location binding*

Σ ::= *store typings*

$\emptyset$ *empty store typing*

$\Sigma, l{:}\mathtt{T}$ *location typing*

$\mathtt{nv}$ ::= *numeric values*

$\mathtt{0}$ *zero value*

$\mathtt{succ}\ \mathtt{nv}$ *successor value*

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

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

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

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

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

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

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

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

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

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

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

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

$$\frac{\mathtt{t}_1 \mid \mu \longrightarrow \mathtt{t}_1' \mid \mu'}{\mathtt{t}_1{:=}\mathtt{t}_2 \mid \mu \longrightarrow \mathtt{t}_1'{:=}\mathtt{t}_2 \mid \mu'} \quad \text{(E-ASSIGN1)}$$

$$\frac{\mathtt{t}_2 \mid \mu \longrightarrow \mathtt{t}_2' \mid \mu'}{\mathtt{v}_1{:=}\mathtt{t}_2 \mid \mu \longrightarrow \mathtt{v}_1{:=}\mathtt{t}_2' \mid \mu'} \quad \text{(E-ASSIGN2)}$$

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

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

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

$$\frac{\texttt{t}_1|\,\mu \longrightarrow \texttt{t}_1'|\,\mu'}{\texttt{succ}\,\texttt{t}_1|\,\mu \longrightarrow \texttt{succ}\,\texttt{t}_1'|\,\mu'} \qquad \text{(E-Succ)}$$

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

$$\texttt{pred (succ}\,\texttt{nv}_1\texttt{)}|\,\mu \longrightarrow \texttt{nv}_1|\,\mu \qquad \text{(E-PredSucc)}$$

$$\frac{\texttt{t}_1|\,\mu \longrightarrow \texttt{t}_1'|\,\mu'}{\texttt{pred}\,\texttt{t}_1|\,\mu \longrightarrow \texttt{pred}\,\texttt{t}_1'|\,\mu} \qquad \text{(E-Pred)}$$

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

$$\texttt{iszero (succ}\,\texttt{nv}_1\texttt{)}|\,\mu \longrightarrow \texttt{false}|\,\mu \qquad \text{(E-IszeroSucc)}$$

$$\frac{\texttt{t}_1|\,\mu \longrightarrow \texttt{t}_1'|\,\mu'}{\texttt{iszero}\,\texttt{t}_1|\,\mu \longrightarrow \texttt{iszero}\,\texttt{t}_1'|\,\mu'} \qquad \text{(E-IsZero)}$$

$$\texttt{let x=v}_1\ \texttt{in}\ \texttt{t}_2|\,\mu \longrightarrow [\texttt{x} \mapsto \texttt{v}_1]\texttt{t}_2|\,\mu \qquad \text{(E-LetV)}$$

$$\frac{\texttt{t}_1|\,\mu \longrightarrow \texttt{t}_1'|\,\mu'}{\texttt{let x=t}_1\ \texttt{in}\ \texttt{t}_2|\,\mu \longrightarrow \texttt{let x=t}_1'\ \texttt{in}\ \texttt{t}_2|\,\mu'} \qquad \text{(E-Let)}$$

$$\begin{array}{c}\texttt{fix}\ (\lambda\texttt{x:T}_1\texttt{.t}_2)|\,\mu \\ \longrightarrow [\texttt{x} \mapsto \texttt{(fix}\ (\lambda\texttt{x:T}_1\texttt{.t}_2\texttt{))}]\texttt{t}_2|\,\mu\end{array} \qquad \text{(E-FixBeta)}$$

$$\frac{\texttt{t}_1|\,\mu \longrightarrow \texttt{t}_1'|\,\mu'}{\texttt{fix}\,\texttt{t}_1|\,\mu \longrightarrow \texttt{fix}\,\texttt{t}_1'|\,\mu} \qquad \text{(E-Fix)}$$

Subtyping $\boxed{\texttt{S <: T}}$

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

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

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

$$\frac{\texttt{T}_1\ \texttt{<:}\ \texttt{S}_1 \qquad \texttt{S}_2\ \texttt{<:}\ \texttt{T}_2}{\texttt{S}_1 \to \texttt{S}_2\ \texttt{<:}\ \texttt{T}_1 \to \texttt{T}_2} \qquad \text{(S-Arrow)}$$

$$\{\texttt{l}_i\texttt{:T}_i\ ^{i \in 1..n+k}\}\ \texttt{<:}\ \{\texttt{l}_i\texttt{:T}_i\ ^{i \in 1..n}\} \qquad \text{(S-RcdWidth)}$$

$$\frac{\text{for each } i \quad \texttt{S}_i\ \texttt{<:}\ \texttt{T}_i}{\{\texttt{l}_i\texttt{:S}_i\ ^{i \in 1..n}\}\ \texttt{<:}\ \{\texttt{l}_i\texttt{:T}_i\ ^{i \in 1..n}\}} \qquad \text{(S-RcdDepth)}$$

$$\frac{\{\texttt{k}_j\texttt{:S}_j\ ^{j \in 1..n}\} \text{ is a permutation of } \{\texttt{l}_i\texttt{:T}_i\ ^{i \in 1..n}\}}{\{\texttt{k}_j\texttt{:S}_j\ ^{j \in 1..n}\}\ \texttt{<:}\ \{\texttt{l}_i\texttt{:T}_i\ ^{i \in 1..n}\}} \qquad \text{(S-RcdPerm)}$$

$$\frac{\texttt{S}_1\ \texttt{<:}\ \texttt{T}_1 \qquad \texttt{T}_1\ \texttt{<:}\ \texttt{S}_1}{\texttt{Ref}\ \texttt{S}_1\ \texttt{<:}\ \texttt{Ref}\ \texttt{T}_1} \qquad \text{(S-Ref)}$$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Featherweight Java

Syntax

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

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

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

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

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

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

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

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

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

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

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

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

$$\frac{\begin{array}{c}CT(\mathtt{C}) = \mathtt{class\ C\ extends\ D}\ \{\overline{\mathtt{C}}\ \overline{\mathtt{f}};\ \mathtt{K}\ \overline{\mathtt{M}}\} \\ \mathtt{B\ m\ (}\overline{\mathtt{B}}\ \overline{\mathtt{x}}\mathtt{)\ \{return\ t;\}} \in \overline{\mathtt{M}}\end{array}}{mbody(\mathtt{m}, \mathtt{C}) = (\overline{\mathtt{x}}, \mathtt{t})}$$

$$\frac{\begin{array}{c}CT(\mathtt{C}) = \mathtt{class\ C\ extends\ D}\ \{\overline{\mathtt{C}}\ \overline{\mathtt{f}};\ \mathtt{K}\ \overline{\mathtt{M}}\} \\ \mathtt{m} \text{ is not defined in } \overline{\mathtt{M}}\end{array}}{mbody(\mathtt{m}, \mathtt{C}) = mbody(\mathtt{m}, \mathtt{D})}$$

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

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

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

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

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

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

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

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

$$\frac{\mathtt{t}_i \longrightarrow \mathtt{t}_i'}{\begin{array}{c}\mathtt{v}_0.\mathtt{m}(\overline{\mathtt{v}}, \mathtt{t}_i, \overline{\mathtt{t}}) \\ \longrightarrow \mathtt{v}_0.\mathtt{m}(\overline{\mathtt{v}}, \mathtt{t}_i', \overline{\mathtt{t}})\end{array}} \qquad \text{(E-Invk-Arg)}$$

$$\frac{\mathtt{t}_i \longrightarrow \mathtt{t}_i'}{\begin{array}{c}\mathtt{new\ C}(\overline{\mathtt{v}}, \mathtt{t}_i, \overline{\mathtt{t}}) \\ \longrightarrow \mathtt{new\ C}(\overline{\mathtt{v}}, \mathtt{t}_i', \overline{\mathtt{t}})\end{array}} \qquad \text{(E-New-Arg)}$$

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

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

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

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

$$\frac{\begin{array}{c}\Gamma \vdash \mathtt{t}_0 : \mathtt{C}_0 \\ mtype(\mathtt{m}, \mathtt{C}_0) = \overline{\mathtt{D}} \to \mathtt{C} \\ \Gamma \vdash \overline{\mathtt{t}} : \overline{\mathtt{C}} \qquad \overline{\mathtt{C}} <: \overline{\mathtt{D}}\end{array}}{\Gamma \vdash \mathtt{t}_0.\mathtt{m}(\overline{\mathtt{t}}) : \mathtt{C}} \qquad \text{(T-Invk)}$$

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

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

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

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

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

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

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

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