CIS 500 — Software Foundations

Final Exam
Answer key
December 17, 2003

Inductive Definitions

1. (4 points)

   Alfred Steeple, a promising young meta-computer scientist, is designing a new programming language based around "frobs". The set of frobs is defined by the following grammar:

$$\begin{array}{rcl} F & ::= & \square \\ & | & F \circ F \\ & | & F \diamond F \\ & | & \star F \end{array}$$

   The most important property of frobs is a three-place relation called "frobnostication." This relation, written $(F_1 \mid F_2 \mid F_3)$ and pronounced "F_1, F_2, and F_3 frobnosticate," is the least relation closed under the following rules:

$$\frac{}{(\square \mid \square \mid \star\square)} \qquad \frac{(F_1 \mid F_2 \mid \star F_3)}{(F_1 \mid F_1 \circ F_2 \mid F_3)} \qquad \frac{(F_2 \mid F_1 \mid \star F_3)}{(F_1 \mid F_1 \circ F_2 \mid F_3)}$$

$$\frac{(F_2 \mid F_1 \mid F_2 \diamond F_1)}{(F_1 \mid F_2 \mid F_1 \diamond F_2)} \qquad \frac{(F_1 \mid F_1 \mid \star F_2) \qquad (F_2 \mid F_2 \mid \star F_1)}{(F_1 \mid F_1 \mid \star\star F_2)}$$

   (Note: Do not waste time trying to understand what frobnostication "means"—Alfred is the only person who understands this.)

   For each of the following triples F_1, F_2, and F_3 of frobs, write "yes" if $(F_1 \mid F_2 \mid F_3)$ is in the frobnostication relation and "no" if it is not.

   (a) $(\square \mid \star\square \mid \star\star\square)$ *Answer: no*

   (b) $(\square \mid \square \mid \square \diamond \square)$ *Answer: no*

   Grading scheme: Binary.

2. (10 points)

Alfred's graduate student, Igor, wrote an OCaml definition for a datatype of frobs.

```
type frob = Box
          | Circ of frob * frob
          | Diamond of frob * frob
          | Star of frob
```

Unfortunately, Igor had some trouble understanding Alfred's inference rules when he implemented some OCaml code to mechanically check frobnostication. Cross out his mistakes and write in appropriate corrections. There is exactly one mistake in each clause.

Reminder: In OCaml, the boolean expression `x && y` evaluates to `true` if both `x` and `y` evaluate to `true`; the expression `x || y` evaluates to `true` if either `x` or `y` does.

```
let rec fnos f1 f2 f3 =
  match (f1, f2, f3) with

    |   (Box, Box, Star Box) → false

    |   (f1, Circ(f1',f2), f3) when f1 = f1' →
          (fnos f1 f2 (Star f3)) && (fnos f2 f1 (Star f3))


    |   (f1, f2, Diamond(f1',f2')) when f1 = f1' && f2 = f2' →
          fnos f2 f1 (Star (f2,f1))


    |   (f1, f1', Star(Star f2)) when f1 = f1' →
          (fnos f1 f1' (Star f2)) || (fnos f2 f2 (Star f1))


    |   (_, _, _) → true
```

Answer:

```
let rec fnos f1 f2 f3 =
  match (f1, f2, f3) with
    |   (Box, Box, Star Box) → true
    |   (f1, Circ(f1',f2), f3) when f1 = f1' →
          (fnos f1 f2 (Star f3)) || (fnos f2 f1 (Star f3))
    |   (f1, f2, Diamond(f1',f2')) when f1 = f1' && f2 = f2' →
          fnos f2 f1 (Diamond (f2,f1))
    |   (f1, f1', Star(Star f2)) when f1 = f1' →
          (fnos f1 f1' (Star f2)) && (fnos f2 f2 (Star f1))
    |   (_, _, _) → false
```

Grading scheme: Binary.

Untyped Lambda Calculus

Recall the definitions of the Church numerals and the basic operations over them from Chapter 5 of TAPL:

```
c0 = λs. λz. z;
c1 = λs. λz. s z;
c2 = λs. λz. s (s z);
c2 = λs. λz. s (s (s z));

scc = λn. λs. λz. s (n s z);
iszro = λm. m (λx. fls) tru;

tru = λt. λf. t
fls = λt. λf. f
and = λb. λc. b c fls;
not = λb. b fls tru
```

Here is the Church encoding of lists from the solution to homework 4:

```
nil = λc. λn. n;
cons = λh. λt. λc. λn. c h (t c n);
head = λl. l (λh.λt.h) fls;
isnil = λl. l (λh.λt.fls) tru;
```

3. (6 points) Fill in the blanks in the following definition to yield a lambda term `makezeros` that, when applied to a church numeral c_n, returns a list containing the element c_0 repeated `n` times. For example, `makezeros` c_3 should behave like `cons` c_0 `(cons` c_0 `(cons` c_0 `nil))`.

   Your answer should use only the combinators defined above (plus applications and variables). Explicit λ-abstractions are not allowed.

   Answer:

   ```
   makezeros = λn. n (cons c0) (nil)
   ```

   Grading scheme: 4 points for `cons` c_0*, 2 points for* `nil`*. Partial credits given for almost correct answers.*

4. (8 points) Fill in the blanks in the following definition to yield a lambda term `allzeros` that, when applied to a list of numbers, returns `tru` if all the numbers in the list are zeros, and otherwise returns `fls`. For example,

   `allzeros (cons` c_2 `(cons` c_0 `(cons` c_1 `nil)))`

   should evaluate to `fls`, and

   `allzeros (cons` c_0 `(cons` c_0 `nil)))`

   should evaluate to `tru`.

   Again, your answer should use only the combinators defined at the top of the page (plus applications and variables). Explicit λ-abstractions are not allowed.

   Answer: `allzeros = λl. l (λa. λb. and (iszro a) b) tru`

   Grading scheme: 6 points for `and (iszro a) b`*, 2 points for* `tru`*. Partial credits given for almost correct answers.*

Simply typed lambda-calculus

The following questions refer to the simply typed lambda-calculus (with base types `Bool` *and* `Nat`*). The syntax, typing, and evaluation rules for this system are given on page 1 of the companion handout.*

5. (9 points) Recall that the *erasure* of a simply typed lambda-term is the untyped term found by deleting all the type annotations on λ-abstractions.

   For each of the following untyped lambda-terms, write down a simply typed term whose erasure is the given term. For example, if the given term were

   ```
   λx. λy. y x
   ```

   then one possible correct answer would be:

   ```
   λx:Bool. λy:Bool→Bool. y x
   ```

   If there is no simply typed term that erases to the given untyped term, write "none."

   (a) `λx. λy. y` *Answer:* `λx:Bool. λy:Bool. y`

   (b) `λx. (λy. y x) x` *Answer: none*

   (c) `λf. λx. f x x` *Answer:* `λf:Bool→Bool→Bool. λx:Bool. f x x`

   Grading scheme: Binary.

6. (8 points) A "zero" of a function t_1 is an input n such that $t_1\ n = 0$. Suppose we want a new language feature that allows us to easily search for a zero of a given function. This can be done by adding a new operator `findzero`, whose behavior can be described informally as follows:

   - if $t_1\ n$ reduces to 0, then `findzero` $t_1\ n$ eventually reduces (perhaps after many steps) to n;
   - if $t_1\ n$ reduces to a numeric value that is different from 0, then `findzero` $t_1\ n$ reduces to `findzero` t_1 `(succ` n`)`.

   Formalize `findzero` by filling in the blanks in the following typing and evaluation rules. Your evaluation rules should enforce a left-to-right, call-by-value reduction strategy, and should preserve the properties of determinism and uniqueness of normal forms.

$$\frac{\Gamma \vdash t_1 : \underline{\texttt{Nat}\to\texttt{Nat}} \qquad \Gamma \vdash t_2 : \underline{\texttt{Nat}}}{\Gamma \vdash \texttt{findzero}\ t_1\ t_2 : \underline{\texttt{Nat}}} \qquad \text{(T-FINDZERO)}$$

$$\frac{t_1 \longrightarrow t_1'}{\texttt{findzero}\ \underline{t_1\ t_2} \longrightarrow \texttt{findzero}\ \underline{t_1'\ t_2}} \qquad \text{(E-FINDZERO1)}$$

$$\frac{\underline{t_2} \longrightarrow \underline{t_2'}}{\texttt{findzero}\ \underline{v_1\ t_2} \longrightarrow \texttt{findzero}\ \underline{v_1\ t_2'}} \qquad \text{(E-FINDZERO2)}$$

$$\texttt{findzero}\ v_1\ nv_2 \longrightarrow \texttt{if iszero (}v_1\ nv_2\texttt{) then }\underline{nv_2}\texttt{ else }\underline{\texttt{findzero}\ v_1\ \texttt{(succ}\ nv_2\texttt{)}} \qquad \text{(E-FINDZEROV)}$$

   Grading scheme: Two points for each rule.

7. (4 points) Can `findzero` be defined as a derived form (i.e., can we "desugar" each `findzero` $t_1\ t_2$ into some ordinary term, with the same typing and evaluation behavior, in the pure simply typed lambda-calculus with booleans and numbers)? If so, give its definition. If not, briefly explain why not.

   Answer:
   No. If `f` *has no zeros at all (e.g.,* `f` *is* `λx:Nat. succ x`*), then, according to the informal description,* `findzero f 0` *must diverge. But we know that all well-typed terms in the simply typed lambda-calculus are terminating. So there is no way of translating* `findzero` *into a well-typed term of this language.*

   Grading scheme: -2 for the wrong answer. -1 for imperfect justification or implementation, -2 for wrong justification or implementation without recursion.

Simply typed lambda-calculus with subtyping

The following questions refer to the pure simply typed lambda-calculus with subtyping (with just `Top`*—no records). The syntax, typing, and evaluation rules for this system are given on page 3 of the companion handout. The rules defining algorithmic the subtyping and typing relations for this system are given on page 4.*

8. (21 points) Recall the inversion lemma for the subtype relation in the simply typed lambda-calculus with subtyping:

   > LEMMA (SUBTYPING INVERSION): If $S <: T_1 \to T_2$, then S has the form $S_1 \to S_2$, with $T_1 <: S_1$ and $S_2 <: T_2$.

   The minimal typing theorem for this system can be stated as follows:

   > THEOREM: If $\Gamma \vdash t : T$, then $\Gamma \vdash\!\triangleright\ t : S$ for some $S <: T$.

   Give a detailed proof of the minimal typing theorem.

   (Note that the system we are considering in this problem, defined on page 3 of the handout, includes just `Top` and $\to$; your proof need not deal with other type constructors such as records. If needed, you may refer [without proof] to the subtyping inversion lemma stated above. Also, you may assume that the declarative and algorithmic subtype relations coincide—i.e., $S <: T$ iff $\vdash\!\triangleright\ S <: T$, for all S and T).

   Answer: By induction on the given (declarative) typing derivation, with a case analysis on the final rule used in the derivation.

   Case T-VAR*:* $t = x \qquad \Gamma(x) = T$

   Immediate, by TA-VAR.

   Case T-ABS*:* $t = \lambda x{:}T_1 . t_2 \qquad \Gamma, x{:}T_1 \vdash t_2 : T_2 \qquad T = T_1 \to T_2$

   By the induction hypothesis, $\Gamma, x{:}T_1 \vdash\!\triangleright\ t_2 : S_2$*, for some* $S_2 <: T_2$*. By* TA-ABS, $\Gamma \vdash\!\triangleright\ t : T_1 \to S_2$*. By* S-ARROW, $T_1 \to S_2 <: T_1 \to T_2$*, as required.*

   Case T-APP*:* $t = t_1\ t_2 \qquad \Gamma \vdash t_1 : T_{11} \to T_{12} \qquad \Gamma \vdash t_2 : T_{11} \qquad T = T_{12}$

   By the induction hypothesis, $\Gamma \vdash\!\triangleright\ t_1 : S_1$ *for some* $S_1 <: T_{11} \to T_{12}$ *and* $\Gamma \vdash\!\triangleright\ t_2 : S_2$ *for some* $S_2 <: T_{11}$*. By the subtyping inversion lemma,* S_1 *must have the form* $S_{11} \to S_{12}$*, for some* S_{11} *and* S_{12} *with* $T_{11} <: S_{11}$ *and* $S_{12} <: T_{12}$*. By transitivity,* $S_2 <: S_{11}$*. By the completeness of algorithmic subtyping (stated above),* $\vdash\!\triangleright\ S_2 <: S_{11}$*. Now, by* TA-APP, $\Gamma \vdash\!\triangleright\ t_1\ t_2 : S_{12}$*, which finishes this case since we already have* $S_{12} <: T_{12}$*.*

   Case T-SUB*:* $t : U \qquad U <: T$

   By the induction hypothesis, $\Gamma \vdash\!\triangleright\ t : S$*, with* $S <: U$*. By* S-TRANS, $S <: T$*.*

   Grading scheme:

   - *-8 for omitting T-APP case.*
   - *-3 for omitting T-VAR case.*
   - *-5 for omitting T-SUB case.*
   - *-1 for not stating that the proof was by induction.*
   - *-5 for a skeleton style proof.*
   - *-1 to -2 for minor "thinkos".*
   - *-3 to -4 for serious localized confusions*
   - *-2 for missing the use of S-TRANS in T-SUB case.*
   - *-3 for using induction in the T-VAR case.*

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

The following questions refer to the simply typed lambda-calculus with subtyping, records, and references (and base types `Nat`, `Bool`, *and* `Unit`*). The syntax, typing, and evaluation rules for this system are given on page 5 of the companion handout.*

9. (3 points)

   (a) Is `{x:Top}` a subtype of `{x:Top→Top}`? *Answer: no*

   (b) Is `{x:Ref Top}` a subtype of `{x:Top}`? *Answer: yes*

   (c) Is `Ref Top` a subtype of `Ref (Ref Top)`? *Answer: no*

   Grading scheme: Binary.

10. (6 points) Recall the definitions of joins and meets in the subtype relation:

    - A type `J` is called a *join* of a pair of types `S` and `T` if `S<:J, T<:J`, and, for all types `U`, if `S<:U` and `T<:U`, then `J<:U`.
    - A type `M` is a *meet* of `S` and `T` if `M<:S, M<:T`, and, for all types `L`, if `L<:S` and `L<:T`, then `L<:M`.

    Write down a join and meet for each pair of types, where they exist (otherwise write "undefined").

    (a)
    ```
    S = { x:{a:Nat, b:Nat} }
    T = { x:{b:Top}, d:Top }
    ```

    Answer:
    join = `{x:{b:Top}}`
    meet = `{x:{a:Nat,b:Nat},d:Top}`

    (b)
    ```
    S = {x:Ref Top}
    T = {x:Ref Nat, y:Ref Top}
    ```

    Answer:
    join = `{x:Top}`
    meet = `undefined`

    (c)
    ```
    S = (Top→Bool) → (Top→Bool)
    T = (Nat→Nat) → (Nat→Top)
    ```

    Answer:
    join = `Top`
    meet = `(Nat→Top)→(Top→Bool)`

    Grading scheme: Binary.

11. (6 points) Suppose we add a new axiom

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

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 fails. For example, the term `{} {}` *is well typed, but stuck.*

Grading scheme: -6 for Yes. -3 for wrong or irrelevant justification.

12. (6 points) Suppose, instead, that we add the subtyping axiom

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

to the original system. Does the progress theorem remain true? 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 it is not).*

Grading scheme: -6 for Yes. -3 for wrong or irrelevant justification.

Featherweight Java

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

Consider the following FJ class definitions (where class `A` is defined elsewhere).

```
class Fst extends Object {
  A f1; A f2;
  Fst(A f1, A f2) { super(); this.f1 = f1; this.f2 = f2; }
  A choose() { return this.f1; }
  A chooseagain() { return this.choose(); }
}

class Snd extends Fst {
  Snd(A f1, A f2) { super(f1, f2); }
  A choose() { return this.f2; }
}

class Thd extends Snd {
  Thd(A f1, A f2) { super(f1, f2); }
  A choose() { return this.chooseagain(); }
}
```

13. (3 points) Suppose that `v` and `w` are values belonging to class `A`. Write down the final results of evaluating the following terms.

    (a) `(new Fst(v,w)).chooseagain()` *Answer:* `v`

    (b) `(new Snd(v,w)).chooseagain()` *Answer:* `w`

    (c) `(new Thd(v,w)).chooseagain()` *Answer: diverges*

    Grading scheme: Binary

14. (11 points) Fill in the blanks in the following encoding of the classes `Fst`, `Snd`, and `Thd` into the simply-typed lambda-calculus with subtyping (and records, references, etc, as shown on page 5 of the companion handout), in the style of Section 18.11 of TAPL.

```
Rep = { f1: Ref A, f2: Ref A };
Fst = Snd = Thd = { choose: Unit→A, chooseagain: Unit→A };
```

Answer:

```
fstClass = λr:Rep. λself: Unit→Fst. λ_:Unit.
             { choose = λ_:Unit. !(r.f1),
               chooseagain = λ_:Unit. (self unit).choose unit };

sndClass = λr:Rep. λself: Unit→Snd. λ_:Unit.
             let super = fstClass r self unit in
             { choose = λ_:Unit. !(r.f2),
               chooseagain = super.chooseagain };

thdClass = λr:Rep. λself: Unit→Thd. λ_:Unit.
             let super = sndClass r self unit in
             { choose = λ_:Unit. (self unit).chooseagain unit,
               chooseagain = super.chooseagain };

newFst = λf1val:A. λf2val:A.
           let r = {f1 = ref f1val, f2 = ref f2val} in
           fix (fstClass r) unit;
```

Grading scheme: One point for each hole.

15. (15 points)

The substitution lemma for FJ is stated as follows:

> LEMMA: If $\Gamma, \overline{x} : \overline{B} \vdash t : C$ and $\Gamma \vdash \overline{s} : \overline{A}$, where $\overline{A} <: \overline{B}$, then $\Gamma \vdash [\overline{x} \mapsto \overline{s}]t : E$ for some $E <: C$.

Fill in the blanks in the following proof of this lemma.

Proof: By induction on the derivation of $\Gamma, \overline{x} : \overline{B} \vdash t : C$.

Case T-VAR*:* $t = x \qquad x : C \in \Gamma$

If $x \notin \overline{x}$, then the result is trivial since $[\overline{x} \mapsto \overline{s}]x = \underline{x}$. On the other hand, if $x = x_i$ and $C = B_i$, then, since $[\overline{x} \mapsto \overline{s}]x = \underline{s_i}$, letting $E = \underline{A_i}$ finishes the case.

Case T-FIELD*:* $t = t_0.f_i \qquad \Gamma, \overline{x} : \overline{B} \vdash t_0 : C_0 \qquad \mathit{fields}(C_0) = \overline{C}\ \overline{f} \qquad C = C_i$

By the induction hypothesis, there is some E_0 such that $\Gamma \vdash [\overline{x} \mapsto \overline{s}]t_0 : E_0$ and $E_0 <: C_0$. It is easy to check that $\mathit{fields}(E_0) = (\mathit{fields}(C_0), \overline{E}\ \overline{g})$ for some $\overline{E}\ \overline{g}$. Therefore, by T-FIELD, $\Gamma \vdash ([\overline{x} \mapsto \overline{s}]t_0).f_i : C_i$.

Case T-INVK*:* $t = t_0.m(\overline{t}) \qquad \Gamma, \overline{x} : \overline{B} \vdash t_0 : C_0 \qquad \mathit{mtype}(m, C_0) = \overline{D} \to C$
$\Gamma, \overline{x} : \overline{B} \vdash \overline{t} : \overline{C} \qquad \overline{C} <: \overline{D}$

By the induction hypothesis, there are some E_0 and $\overline{E}$ such that:

$$\Gamma \vdash [\overline{x} \mapsto \overline{s}]t_0 : E_0 \qquad E_0 <: C_0 \qquad \Gamma \vdash [\overline{x} \mapsto \overline{s}]\overline{t} : \overline{E} \qquad \underline{\overline{E} <: \overline{C}}.$$

It is easy to check that $\mathit{mtype}(m, E_0) = \overline{D} \to C$. Moreover, $\overline{E} <: \overline{D}$ by the transitivity of $<:$. Therefore, by T-INVK, $\Gamma \vdash \underline{[\overline{x} \mapsto \overline{s}]t_0.m([\overline{x} \mapsto \overline{s}]\overline{t})} : C$.

Case T-NEW*:* $t = \mathsf{new}\, C(\overline{t}) \qquad \mathit{fields}(C) = \overline{C}\ \overline{f} \qquad \Gamma, \overline{x} : \overline{B} \vdash \overline{t} : \overline{D} \qquad \overline{D} <: \overline{C}$

By the induction hypothesis, $\Gamma \vdash [\overline{x} \mapsto \overline{s}]\overline{t} : \overline{E}$ for some $\overline{E}$ with $\overline{E} <: \overline{D}$. We have $\overline{E} <: \overline{C}$ by the transitivity of $<:$. Therefore, by T-NEW, $\Gamma \vdash \mathsf{new}\, C([\overline{x} \mapsto \overline{s}]\overline{t}) : C$.

Case T-UCAST*:* $t = (C)t_0 \qquad \Gamma, \overline{x} : \overline{B} \vdash t_0 : D \qquad D <: C$

By the induction hypothesis, there is some E such that $\Gamma \vdash [\overline{x} \mapsto \overline{s}]t_0 : E$ and $E <: D$. We obtain $E <: C$ by the transitivity of $<:$, which yields $\Gamma \vdash \underline{(C)([\overline{x} \mapsto \overline{s}]t_0)} : C$ by T-UCAST.

Case T-DCAST*:* $t = (C)t_0 \qquad \Gamma, \overline{x} : \overline{B} \vdash t_0 : D \qquad C <: D \qquad C \neq D$

By the induction hypothesis, there is some E such that $\Gamma \vdash [\overline{x} \mapsto \overline{s}]t_0 : E$ and $E <: D$. If $E <: C$ or $C <: E$, then $\Gamma \vdash (C)([\overline{x} \mapsto \overline{s}]t_0) : C$ by T-UCAST or T-DCAST, respectively. On the other hand, if both $C \not<: E$ and $E \not<: C$, then $\Gamma \vdash (C)([\overline{x} \mapsto \overline{s}]t_0) : C$ (with a *stupid warning*) by T-SCAST.

Case T-SCAST*:* $t = \underline{(C)t_0} \qquad \Gamma, \overline{x} : \overline{B} \vdash t_0 : D \qquad C \not<: D \qquad D \not<: C$

By the induction hypothesis, there is some E such that $\Gamma \vdash [\overline{x} \mapsto \overline{s}]t_0 : E$ and $E <: D$. It is easy to show, by induction on the subtype relation, that if $E <: C$ and $\underline{E <: D}$, then either $C <: D$ or $D <: C$; since we know that neither of these is the case, we may conclude that $\underline{E \not<: C}$. Moreover, if it were the case that $C <: E$, then we would have (by transitivity) that $C <: D$, which we know is not the case; so we may conclude that $C \not<: E$. Thus, $\Gamma \vdash (C)([\overline{x} \mapsto \overline{s}]t_0) : C$ (with a *stupid warning*), by T-SCAST. □

Grading scheme: Binary.

Companion handout

Full definitions of the systems used in the exam

Simply typed lambda calculus (with `Bool` and `Nat`)

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*
		`x`	*variable*
		`λx:T.t`	*abstraction*
		`t t`	*application*
v	::=		*values*
		`true`	*true value*
		`false`	*false value*
		`nv`	*numeric value*
		`λx:T.t`	*abstraction value*
nv	::=		*numeric values*
		`0`	*zero value*
		`succ nv`	*successor value*
T	::=		*types*
		`Bool`	*type of booleans*
		`Nat`	*type of natural numbers*
		`T→T`	*type of functions*
Γ	::=		*contexts*
		$\emptyset$	*empty context*
		Γ, `x:T`	*term variable binding*

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Typing $\boxed{\Gamma \vdash \texttt{t} : \texttt{T}}$

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

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

$$\frac{\Gamma \vdash \texttt{t}_1 : \texttt{Bool} \quad \Gamma \vdash \texttt{t}_2 : \texttt{T} \quad \Gamma \vdash \texttt{t}_3 : \texttt{T}}{\Gamma \vdash \texttt{if}\ \texttt{t}_1\ \texttt{then}\ \texttt{t}_2\ \texttt{else}\ \texttt{t}_3 : \texttt{T}} \qquad \text{(T-If)}$$

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

$$\frac{\Gamma \vdash \texttt{t}_1 : \texttt{Nat}}{\Gamma \vdash \texttt{succ}\ \texttt{t}_1 : \texttt{Nat}} \qquad \text{(T-Succ)}$$

$$\frac{\Gamma \vdash \texttt{t}_1 : \texttt{Nat}}{\Gamma \vdash \texttt{pred}\ \texttt{t}_1 : \texttt{Nat}} \qquad \text{(T-Pred)}$$

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

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

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

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

Pure simply typed lambda calculus with subtyping (no records)

Syntax

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

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

$$\frac{t_1 \longrightarrow t_1'}{t_1\ t_2 \longrightarrow t_1'\ t_2} \quad \text{(E-App1)}$$

$$\frac{t_2 \longrightarrow t_2'}{v_1\ t_2 \longrightarrow v_1\ t_2'} \quad \text{(E-App2)}$$

$$(\lambda x{:}T_{11}.t_{12})\ v_2 \longrightarrow [x \mapsto v_2]t_{12} \quad \text{(E-AppAbs)}$$

Subtyping $\boxed{S <: T}$

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

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

$$S <: \mathsf{Top} \quad \text{(S-Top)}$$

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

Typing $\boxed{\Gamma \vdash t : T}$

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

$$\frac{\Gamma, x{:}T_1 \vdash t_2 : T_2}{\Gamma \vdash \lambda x{:}T_1.t_2 : T_1{\to}T_2} \quad \text{(T-Abs)}$$

$$\frac{\Gamma \vdash t_1 : T_{11}{\to}T_{12} \qquad \Gamma \vdash t_2 : T_{11}}{\Gamma \vdash t_1\ t_2 : T_{12}} \quad \text{(T-App)}$$

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

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

Algorithmic subtyping $\boxed{\vdash_\blacktriangleright S <: T}$

$$\vdash_\blacktriangleright S <: \mathsf{Top} \qquad \text{(SA-Top)}$$

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

Algorithmic typing $\boxed{\Gamma \vdash_\blacktriangleright t : T}$

$$\frac{x : T \in \Gamma}{\Gamma \vdash_\blacktriangleright x : T} \qquad \text{(TA-Var)}$$

$$\frac{\Gamma, x : T_1 \vdash_\blacktriangleright t_2 : T_2}{\Gamma \vdash_\blacktriangleright \lambda x : T_1 . t_2 : T_1 \to T_2} \qquad \text{(TA-Abs)}$$

$$\frac{\Gamma \vdash_\blacktriangleright t_1 : T_1 \qquad T_1 = T_{11} \to T_{12} \qquad \Gamma \vdash_\blacktriangleright t_2 : T_2 \qquad \vdash_\blacktriangleright T_2 <: T_{11}}{\Gamma \vdash_\blacktriangleright t_1\ t_2 : T_{12}} \qquad \text{(TA-App)}$$

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*

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

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

$$\frac{t_1 \mid \mu \longrightarrow t_1' \mid \mu'}{t_1\ t_2 \mid \mu \longrightarrow t_1'\ t_2 \mid \mu'} \quad \text{(E-App1)}$$

$$\frac{t_2 \mid \mu \longrightarrow t_2' \mid \mu'}{v_1\ t_2 \mid \mu \longrightarrow v_1\ t_2' \mid \mu'} \quad \text{(E-App2)}$$

$$(\lambda x{:}T_{11}.t_{12})\ v_2 \mid \mu \longrightarrow [x \mapsto v_2]t_{12} \mid \mu \quad \text{(E-AppAbs)}$$

$$\{l_i{=}v_{i\ i\in 1..n}\}.l_j \mid \mu \longrightarrow v_j \mid \mu \quad \text{(E-ProjRcd)}$$

$$\frac{t_1 \mid \mu \longrightarrow t_1' \mid \mu'}{t_1.l \mid \mu \longrightarrow t_1'.l \mid \mu'} \quad \text{(E-Proj)}$$

$$\frac{t_j \mid \mu \longrightarrow t_j' \mid \mu'}{\begin{array}{l}\{l_i{=}v_{i\ i\in 1..j-1}, l_j{=}t_j, l_k{=}t_{k\ k\in j+1..n}\} \mid \mu \\ \longrightarrow \{l_i{=}v_{i\ i\in 1..j-1}, l_j{=}t_j', l_k{=}t_{k\ k\in j+1..n}\} \mid \mu'\end{array}} \quad \text{(E-Rcd)}$$

$$\frac{l \notin dom(\mu)}{\text{ref } v_1 \mid \mu \longrightarrow l \mid (\mu, l \mapsto v_1)} \quad \text{(E-RefV)}$$

$$\frac{t_1 \mid \mu \longrightarrow t_1' \mid \mu'}{\text{ref } t_1 \mid \mu \longrightarrow \text{ref } t_1' \mid \mu'} \quad \text{(E-Ref)}$$

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

$$\frac{t_1 \mid \mu \longrightarrow t_1' \mid \mu'}{!t_1 \mid \mu \longrightarrow !t_1' \mid \mu'} \quad \text{(E-Deref)}$$

$$l{:=}v_2 \mid \mu \longrightarrow \text{unit} \mid [l \mapsto v_2]\mu \quad \text{(E-Assign)}$$

$$\frac{t_1 \mid \mu \longrightarrow t_1' \mid \mu'}{t_1{:=}t_2 \mid \mu \longrightarrow t_1'{:=}t_2 \mid \mu'} \quad \text{(E-Assign1)}$$

$$\frac{t_2 \mid \mu \longrightarrow t_2' \mid \mu'}{v_1{:=}t_2 \mid \mu \longrightarrow v_1{:=}t_2' \mid \mu'} \quad \text{(E-Assign2)}$$

$$\text{if true then } t_2 \text{ else } t_3 \mid \mu \longrightarrow t_2 \mid \mu \quad \text{(E-IfTrue)}$$

$$\text{if false then } t_2 \text{ else } t_3 \mid \mu \longrightarrow t_3 \mid \mu \quad \text{(E-IfFalse)}$$

$$\frac{t_1 \mid \mu \longrightarrow t_1' \mid \mu'}{\text{if } t_1 \text{ then } t_2 \text{ else } t_3 \mid \mu \longrightarrow \text{if } t_1' \text{ then } t_2 \text{ else } t_3 \mid \mu'} \quad \text{(E-If)}$$

$$\frac{t_1 \mid \mu \longrightarrow t_1' \mid \mu'}{\mathtt{succ}\ t_1 \mid \mu \longrightarrow \mathtt{succ}\ t_1' \mid \mu'} \quad \text{(E-Succ)}$$

$$\mathtt{pred}\ \mathtt{0} \mid \mu \longrightarrow \mathtt{0} \mid \mu \quad \text{(E-PredZero)}$$

$$\mathtt{pred}\ (\mathtt{succ}\ nv_1) \mid \mu \longrightarrow nv_1 \mid \mu \quad \text{(E-PredSucc)}$$

$$\frac{t_1 \mid \mu \longrightarrow t_1' \mid \mu'}{\mathtt{pred}\ t_1 \mid \mu \longrightarrow \mathtt{pred}\ t_1' \mid \mu} \quad \text{(E-Pred)}$$

$$\mathtt{iszero}\ \mathtt{0} \mid \mu \longrightarrow \mathtt{true} \mid \mu \quad \text{(E-IszeroZero)}$$

$$\mathtt{iszero}\ (\mathtt{succ}\ nv_1) \mid \mu \longrightarrow \mathtt{false} \mid \mu \quad \text{(E-IszeroSucc)}$$

$$\frac{t_1 \mid \mu \longrightarrow t_1' \mid \mu'}{\mathtt{iszero}\ t_1 \mid \mu \longrightarrow \mathtt{iszero}\ t_1' \mid \mu'} \quad \text{(E-IsZero)}$$

$$\mathtt{let}\ x{=}v_1\ \mathtt{in}\ t_2 \mid \mu \longrightarrow [x \mapsto v_1]t_2 \mid \mu \quad \text{(E-LetV)}$$

$$\frac{t_1 \mid \mu \longrightarrow t_1' \mid \mu'}{\mathtt{let}\ x{=}t_1\ \mathtt{in}\ t_2 \mid \mu \longrightarrow \mathtt{let}\ x{=}t_1'\ \mathtt{in}\ t_2 \mid \mu'} \quad \text{(E-Let)}$$

$$\mathtt{fix}\ (\lambda x{:}T_1.\,t_2) \mid \mu \longrightarrow [x \mapsto (\mathtt{fix}\ (\lambda x{:}T_1.\,t_2))]t_2 \mid \mu \quad \text{(E-FixBeta)}$$

$$\frac{t_1 \mid \mu \longrightarrow t_1' \mid \mu'}{\mathtt{fix}\ t_1 \mid \mu \longrightarrow \mathtt{fix}\ t_1' \mid \mu} \quad \text{(E-Fix)}$$

Subtyping $\boxed{S <: T}$

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

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

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

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

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

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

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

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

$$\frac{\text{for each } i \quad \Gamma \mid \Sigma \vdash t_i : T_i}{\Gamma \mid \Sigma \vdash \{l_i = t_i \ ^{i \in 1..n}\} : \{l_i : T_i \ ^{i \in 1..n}\}} \quad \text{(T-RCD)}$$

$$\frac{\Gamma \mid \Sigma \vdash t_1 : \{l_i : T_i \ ^{i \in 1..n}\}}{\Gamma \mid \Sigma \vdash t_1 . l_j : T_j} \quad \text{(T-PROJ)}$$

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

$$\frac{\Gamma, x : T_1 \mid \Sigma \vdash t_2 : T_2}{\Gamma \mid \Sigma \vdash \lambda x : T_1 . t_2 : T_1 \to T_2} \quad \text{(T-ABS)}$$

$$\frac{\Gamma \mid \Sigma \vdash t_1 : T_{11} \to T_{12} \qquad \Gamma \mid \Sigma \vdash t_2 : T_{11}}{\Gamma \mid \Sigma \vdash t_1 \ t_2 : T_{12}} \quad \text{(T-APP)}$$

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

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

$$\frac{\Sigma(l) = T_1}{\Gamma \mid \Sigma \vdash l : \mathtt{Ref}\ T_1} \quad \text{(T-LOC)}$$

$$\frac{\Gamma \mid \Sigma \vdash t_1 : T_1}{\Gamma \mid \Sigma \vdash \mathtt{ref}\ t_1 : \mathtt{Ref}\ T_1} \quad \text{(T-REF)}$$

$$\frac{\Gamma \mid \Sigma \vdash t_1 : \mathtt{Ref}\ T_{11}}{\Gamma \mid \Sigma \vdash !t_1 : T_{11}} \quad \text{(T-DEREF)}$$

$$\frac{\Gamma \mid \Sigma \vdash t_1 : \mathtt{Ref}\ T_{11} \qquad \Gamma \mid \Sigma \vdash t_2 : T_{11}}{\Gamma \mid \Sigma \vdash t_1 := t_2 : \mathtt{Unit}} \quad \text{(T-ASSIGN)}$$

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

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

$$\frac{\Gamma \mid \Sigma \vdash t_1 : \mathtt{Bool} \qquad \Gamma \mid \Sigma \vdash t_2 : T \qquad \Gamma \mid \Sigma \vdash t_3 : T}{\Gamma \mid \Sigma \vdash \mathtt{if}\ t_1\ \mathtt{then}\ t_2\ \mathtt{else}\ t_3 : T} \quad \text{(T-IF)}$$

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

$$\frac{\Gamma \mid \Sigma \vdash t_1 : \mathtt{Nat}}{\Gamma \mid \Sigma \vdash \mathtt{succ}\ t_1 : \mathtt{Nat}} \quad \text{(T-SUCC)}$$

$$\frac{\Gamma \mid \Sigma \vdash t_1 : \mathtt{Nat}}{\Gamma \mid \Sigma \vdash \mathtt{pred}\ t_1 : \mathtt{Nat}} \quad \text{(T-PRED)}$$

$$\frac{\Gamma \mid \Sigma \vdash t_1 : \mathtt{Nat}}{\Gamma \mid \Sigma \vdash \mathtt{iszero}\ t_1 : \mathtt{Bool}} \quad \text{(T-ISZERO)}$$

$$\frac{\Gamma \mid \Sigma \vdash t_1 : T_1 \qquad \Gamma, x : T_1 \mid \Sigma \vdash t_2 : T_2}{\Gamma \mid \Sigma \vdash \mathtt{let}\ x = t_1\ \mathtt{in}\ t_2 : T_2} \quad \text{(T-LET)}$$

$$\frac{\Gamma \mid \Sigma \vdash t_1 : T_1 \to T_1}{\Gamma \mid \Sigma \vdash \mathtt{fix}\ t_1 : T_1} \quad \text{(T-FIX)}$$

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*
	$\texttt{t.m}(\overline{\texttt{t}})$	*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 $\boxed{\mathit{fields}(\texttt{C}) = \overline{\texttt{C}}\ \overline{\texttt{f}}}$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$\frac{\mathit{fields}(\mathtt{C}) = \overline{\mathtt{D}}\ \overline{\mathtt{f}} \qquad \Gamma \vdash \overline{\mathtt{t}} : \overline{\mathtt{C}} \qquad \overline{\mathtt{C}} <: \overline{\mathtt{D}}}{\Gamma \vdash \mathtt{new}\ \mathtt{C}(\overline{\mathtt{t}}) : \mathtt{C}} \quad \text{(T-NEW)}$$

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

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

$$\frac{\Gamma \vdash \mathtt{t}_0 : \mathtt{D} \qquad \mathtt{C} \not<: \mathtt{D} \qquad \mathtt{D} \not<: \mathtt{C} \qquad \textit{stupid warning}}{\Gamma \vdash (\mathtt{C})\mathtt{t}_0 : \mathtt{C}} \quad \text{(T-SCAST)}$$

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

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

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

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