CIS 500 — Software Foundations

Final Exam (and WPE-I exam)

Answer key

November 13, 2002

Inductive definitions

1. (20 points) Suppose we define a set of expressions called *widgets* as follows. (We'll use the metavariables X, Y, A, B, and C to range over widgets.)

$$\begin{array}{lcl} X & ::= & X \otimes X \\ & & X \square X \\ & & \star \end{array}$$

We can express the syntax of widgets equivalently in OCaml, as follows:

```
type widget = Cross of widget * widget
            | Box of widget * widget
            | Star
```

The following OCaml function computes a relation on widgets, written $X \multimap Y$ in mathematical notation and `lolli widX widY` in ascii. (Note: don't waste time figuring out what the $\multimap$ relation "means": it was invented just for the exam and is not intended to correspond to anything computationally natural.)

```
let rec lolli widX widY =
  widX = widY  ||
  match (widX, widY) with
      (Cross(widA, widB), Cross(widC, widD)) →
        (lolli widA widC) || (lolli widB widD)
    | (Box(widA, Star), Star) →
        lolli widA Star
    | (Box(widA, Cross(widB, widC)), _) →
        lolli (Cross(widA, widY)) (Box(widC, widB))
    | (_, Box(widB, widC)) →
        (lolli widB Star) && (lolli widC widX)
    | (_,_) →
         false
```

Recall that the OCaml operator && is "boolean and" and || is "or."

Give a set of inference rules for the $\multimap$ relation corresponding to the given OCaml implementation—i.e., such that $X \multimap Y$ iff evaluating `lolli widX widY` in OCaml yields `true` (where `widX` is an OCaml value of type `widget` representing the mathematical widget X and similarly `widY` represents Y). One of the required rules is given as an example; you write the rest.

$$\frac{A \multimap \star}{A \square \star \multimap \star}$$

Answer:

$$A \multimap A$$

$$\frac{A \multimap C}{A \otimes B \multimap C \otimes D}$$

$$\frac{B \multimap D}{A \otimes B \multimap C \otimes D}$$

$$\frac{A \otimes Y \multimap C \square B}{A \square (B \otimes C) \multimap Y}$$

$$\frac{B \multimap \star \qquad C \multimap X}{X \multimap B \square C}$$

Grading scheme:

- *0 points for omitted rule*
- *4 points for each rule with the following deductions:*
  - *-2 for each upside-down rule*
  - *-2 for wild card or blank where meta variable should be*
  - *-1 for incorrect variable other than wild card*
- *No deductions were made for the following*
  - *full credit (8/8) for folding to "or" rules together into 1 rule with an "or" separating the premises*
  - *attempting a rule for the last case in the OCaml match*
  - *No deduction for minor, but clear, notational deviations, including a different arrow character or a blank, or ordered pair notation instead of lollipop arrow*
  - *no deduction for omitting parens from 4th rule in key (a surprisingly rare error!)*

Untyped lambda-calculus

2. Recall that the *Church encoding* of the natural numbers in untyped lambda-calculus represents a number *n* as a function that applies a given “successor operation” *n* times to a given “starting value.”

   Church numerals:

```
0 = λs. λz. z
1 = λs. λz. (s z)
2 = λs. λz. s (s z)
3 = λs. λz. s (s (s z))
```

   In this problem, we explore a different encoding of natural numbers based on their *binary* representation. Example encodings in this new system are:

   Binary numerals:

```
0 = λone. λzero. λt. zero t
1 = λone. λzero. λt. one t
2 = λone. λzero. λt. one (zero t)
3 = λone. λzero. λt. one (one t)
4 = λone. λzero. λt. one (zero (zero t))
5 = λone. λzero. λt. one (zero (one t))
```

   As in the standard Church encoding, numbers are represented as functions that can be used to “count.” However, rather then applying a single given function *n* times to a starting value, the binary encoding is supplied with *two* functions plus a starting value. It applies its `one` argument to the starting value if the lowest-order bit of the binary representation of *n* is 1; otherwise (if the lowest order bit is 0) it applies its `zero` argument. To the result, it applies `one` if the next lowest bit in the binary representation of *n* is 1, otherwise `zero`; and so on up to the highest-order bit.

   Note that, leading zeros are considered insignificant in this encoding—i.e., both

```
fn one. fn zero. fn t.  zero (one t)
```

   and

```
fn one. fn zero. fn t. one t
```

   are valid encodings of the number 1. (This implies that the function passed for the `zero` argument should “have no effect” if it is the last one to be applied.)

   (a) (7 points) Write a function that multiplies a number in the binary encoding by 2. (Hint: in base 2, a multiplication by 2 is just a one-bit shift.)

   Answer: `λn. λone. λzero. λt. n one zero (zero t)`

   Grading scheme:

   - *2 points for correct args./shape of the function*
   - *3 points for using the term `(zero t)` in roughly the correct way*
   - *2 points for having everything else right*

   (b) (7 points) Recall that we can encode Boolean values in the untyped lambda calculus as follows:

```
tru = λt. λf. t
fls = λt. λf. f
```

Write a function that returns `tru` when given the binary encoding of `0` and `fls` if given the encoding of any non-zero number.

Answer: `λn. n (λx. fls) (λy. y) tru`

Grading scheme:

- *2 points for correct args./shape of the function*
- *1 point for replacing 'one' by something like "λx. fls"*
- *1 point for replacing 'zero' by something like "λx. x"*
- *1 point for replacing 't' by something like "tru"*
- *2 points for everything else right extra points for being close*

Simply typed lambda-calculus

The definition of the simply typed lambda-calculus is reproduced on pages 15 to 17.

3. (4 points)

For each of the following types, write a closed lambda-term that has that type *and that contains at least one application.* For example, for the type `(Nat→Nat)→Nat`, the following is a correct answer:

```
λf:Nat→Nat. f 1
```

On the other hand,

```
λf:Nat→Nat. 1
```

is not an acceptable answer (even though it has the correct type) because it does not contain an application.

(a) `((Nat→Unit)→Nat) → Nat`
Answer: *`λf:(Nat→Unit)→Nat. f (λx:Nat. unit)`*

(b) `(Unit→Nat) → ((Unit→Nat)→Nat) → Nat`
Answer: *`λf:Unit→Nat. λg:(Unit→Nat)→Nat. g f`*

Grading scheme: Two points for each part.

Subtyping

4. (20 points) In this question, you will be asked to fill in a proof of the preservation theorem for the simply typed lambda calculus with records and subtyping. The theorem is identical to the one in Chapter 15 of TAPL, but for the sake of variety we'll do proof by induction on evaluation derivations (in TAPL it went by induction on typing derivations).

   The following two lemmas will be used in the proof. (Note that the first four cases of the first lemma list inversion properties of the typing relation, while the last case concerns the subtype relation.)

   LEMMA [INVERSION]:

   (a) If $\Gamma \vdash \{\mathtt{k}_i\mathtt{=s}_i\ ^{i\in 1..m}\}\ \mathtt{:}\ \mathtt{T}$, then there are types $\mathtt{T}_1$ to $\mathtt{T}_m$ such that $\Gamma \vdash \mathtt{s}_i\ \mathtt{:}\ \mathtt{T}_i$ for each i and such that $\{\mathtt{k}_i\mathtt{:T}_i\ ^{i\in 1..m}\}\ \mathtt{<:}\ \mathtt{T}$.

   (b) If $\Gamma \vdash \mathtt{t.l}\ \mathtt{:}\ \mathtt{T}$, then $\Gamma \vdash \mathtt{t}\ \mathtt{:}\ \{\mathtt{l:T}\}$.

   (c) If $\Gamma \vdash \mathtt{t}_1\ \mathtt{t}_2\ \mathtt{:}\ \mathtt{T}$ then $\Gamma \vdash \mathtt{t}_1\ \mathtt{:}\ \mathtt{T}'\to\mathtt{T}$ and $\Gamma \vdash \mathtt{t}_2\ \mathtt{:}\ \mathtt{T}'$.

   (d) If $\Gamma \vdash \lambda\mathtt{x:S}_1\mathtt{.s}_2\ \mathtt{:}\ \mathtt{T}_1\to\mathtt{T}_2$, then $\mathtt{T}_1\ \mathtt{<:}\ \mathtt{S}_1$ and $\Gamma, \mathtt{x:S}_1 \vdash \mathtt{s}_2\ \mathtt{:}\ \mathtt{T}_2$.

   (e) If $\mathtt{S}\ \mathtt{<:}\ \{\mathtt{l}_j\mathtt{:T}_j\ ^{i\in 1..m}\}$, then $\mathtt{S} = \{\mathtt{k}_i\mathtt{:S}_i\ ^{i\in 1..n}\}$, where $\{\mathtt{l}_j\ ^{i\in 1..m}\} \subseteq \{\mathtt{k}_i\ ^{i\in 1..n}\}$ and $\mathtt{S}_i\ \mathtt{<:}\ \mathtt{T}_j$ for each common label $\mathtt{k}_i = \mathtt{l}_j$.

   LEMMA [SUBSTITUTION]: If $\Gamma, \mathtt{x:S} \vdash \mathtt{t}\ \mathtt{:}\ \mathtt{T}$ and $\Gamma \vdash \mathtt{s}\ \mathtt{:}\ \mathtt{S}$, then $\Gamma \vdash [\mathtt{x} \mapsto \mathtt{s}]\mathtt{t}\ \mathtt{:}\ \mathtt{T}$.

   For each case in the proof on the next page, write down the *skeleton* of the argument. A skeleton contains the same sequence of steps as the full argument, but omits all details. The rules for writing skeletons are as follows:

   - Steps of the form "By part (x) of the inversion lemma, we obtain..." in the full argument become "inversion(x)" in the skeleton.
   - Steps of the form "By the substitution lemma, we obtain..." become "substitution."
   - Steps of the form "By the induction hypothesis, we obtain..." become "IH."
   - Steps of the form "By typing rule T-XXX, we obtain..." become "T-XXX."
   - If the full argument doesn't use any of the lemmas or the induction hypothesis, then its skeleton is "Direct."

   For example, if the full argument is

   > *Case* E-APP1*:* $\quad \mathtt{t} = \mathtt{t}_1\ \mathtt{t}_2 \qquad \mathtt{t}' = \mathtt{t}_1'\ \mathtt{t}_2 \qquad \mathtt{t}_1 \longrightarrow \mathtt{t}_1'$
   >
   > By part (c) of the inversion lemma, we obtain $\Gamma \vdash \mathtt{t}_1\ \mathtt{:}\ \mathtt{T}'\to\mathtt{T}$ and $\Gamma \vdash \mathtt{t}_2\ \mathtt{:}\ \mathtt{T}'$. By the induction hypothesis, we obtain $\Gamma \vdash \mathtt{t}_1'\ \mathtt{:}\ \mathtt{T}'\to\mathtt{T}$. The required result ($\Gamma \vdash \mathtt{t}_1'\ \mathtt{t}_2\ \mathtt{:}\ \mathtt{T}$) now follows by T-APP.

   the skeleton is written:

   > *Case* E-APP1*:* $\quad \mathtt{t} = \mathtt{t}_1\ \mathtt{t}_2 \qquad \mathtt{t}' = \mathtt{t}_1'\ \mathtt{t}_2 \qquad \mathtt{t}_1 \longrightarrow \mathtt{t}_1'$
   >
   > Inversion(c), IH, T-APP.

THEOREM [PRESERVATION]: If $\mathtt{t} \longrightarrow \mathtt{t}'$ and $\Gamma \vdash \mathtt{t} : \mathtt{T}$, then $\Gamma \vdash \mathtt{t}' : \mathtt{T}$.

Proof: By induction on evaluation derivations, with a case analysis on the final rule used.

Case E-APP1*:* $\mathtt{t} = \mathtt{t}_1\ \mathtt{t}_2 \quad \mathtt{t}' = \mathtt{t}_1'\ \mathtt{t}_2 \quad \mathtt{t}_1 \longrightarrow \mathtt{t}_1'$

Inversion(c), IH, T-APP.

Case E-APP2*:* $\mathtt{t} = \mathtt{v}_1\ \mathtt{t}_2 \quad \mathtt{t}' = \mathtt{v}_1\ \mathtt{t}_2' \quad \mathtt{t}_2 \longrightarrow \mathtt{t}_2'$

Answer: Inversion(c), IH, T-APP*.*

Case E-APPABS*:* $\mathtt{t} = (\lambda\mathtt{x}{:}\mathtt{T}_{11}.\mathtt{t}_{12})\ \mathtt{v}_2 \quad \mathtt{t}' = [\mathtt{x} \mapsto \mathtt{v}_2]\mathtt{t}_{12}$

Answer: Inversion(c), inversion(d), T-SUB*, substitution.*

Case E-RCD*:* $\mathtt{t} = \{\mathtt{l}_i{=}\mathtt{v}_i\ ^{i\in 1..j-1}, \mathtt{l}_j{=}\mathtt{t}_j, \mathtt{l}_k{=}\mathtt{t}_k\ ^{k\in j+1..n}\}$
$\mathtt{t}' = \{\mathtt{l}_i{=}\mathtt{v}_i\ ^{i\in 1..j-1}, \mathtt{l}_j{=}\mathtt{t}_j', \mathtt{l}_k{=}\mathtt{t}_k\ ^{k\in j+1..n}\}$
$\mathtt{t}_j \longrightarrow \mathtt{t}_j'$

Answer: Inversion(a), IH, T-RCD, T-SUB*.*

Case E-PROJ*:* $\mathtt{t} = \mathtt{t}_1.\mathtt{l} \quad \mathtt{t}' = \mathtt{t}_1'.\mathtt{l} \quad \mathtt{t}_1 \longrightarrow \mathtt{t}_1'$

Answer: Inversion(b), IH, T-PROJ*.*

Case E-PROJRCD*:* $\mathtt{t} = \{\mathtt{l}_i{=}\mathtt{v}_i\ ^{i\in 1..n}\}.\mathtt{l}_j \quad \mathtt{t}' = \mathtt{v}_j$

Answer: Inversion(b), inversion(a), inversion(e), T-SUB*.*

Grading scheme: 4 points per case. -2 for missing or extra IH. -1 for missing rule. -1 for extraneous rule. Exceptions: instead of counting off 2 points for a wrong but similar rule (i.e. the right rule is missing and the wrong rule extraneous) only deducted 1 point. Also, only 1 point deducted for rules in transposed order.

5. (3 points) Suppose we changed part (a) of the inversion lemma in problem 4 to match the one given in chapter 15 of TAPL:

   If $\Gamma \vdash \{\mathtt{k}_a\mathtt{=}\mathtt{s}_a{}^{a\in 1..m}\} : \{\mathtt{l}_i\mathtt{:}\mathtt{T}_i{}^{i\in 1..n}\}$, then $\{\mathtt{l}_i{}^{i\in 1..n}\} \subseteq \{\mathtt{k}_a{}^{a\in 1..m}\}$ and $\Gamma \vdash \mathtt{s}_a : \mathtt{T}_i$ for each common label $\mathtt{k}_a = \mathtt{l}_i$.

   Which step would *fail* in the above proof of preservation? Briefly explain why.

   Answer: The E-Rcd *case would fail, for several reasons: First of all, the new case (a) of the inversion lemma would not apply at all, because the assumption on* $\mathtt{t}$ *is just that it has some type* $\mathtt{T}$*, not that* $\mathtt{T}$ *has the form of a record type. Even assuming that we* could *apply the new inversion lemma, we would not get much further since, in order to apply the IH and reconstruct the original type for* $\mathtt{t}'$*, we need to have typing derivations for all fields of the record; but this variant of the rule doesn't provide typing derivations for fields that do not appear in* $\{\mathtt{l}_i\mathtt{:}\mathtt{T}_i{}^{i\in 1..n}\}$.

   The other cases all still work (including in particular E-ProjRcd*).*

   Grading scheme: 1 point for mentioning E-Rcd*; 0 points if* E-Rcd *not mentioned. -1 for mentioning cases besides* E-Rcd*. One additional point for an incomplete but not wrong explanation. Full credit for explaining* any *of the difficulties listed in the sample answer above. (Strictly speaking, only the first difficulty is correct: the new inversion lemma simply cannot be applied. But we still gave full credit to answers that focused on the other difficulties.)*

6. (3 points) Another natural-looking variant of part (a) of the inversion lemma is this:

   If $\Gamma \vdash \{\mathtt{k}_i\mathtt{=}\mathtt{s}_i{}^{i\in 1..n}\} : \mathtt{T}$, then $\mathtt{T} = \{\mathtt{k}_i\mathtt{:}\mathtt{T}_i{}^{i\in 1..n}\}$ and $\Gamma \vdash \mathtt{s}_i : \mathtt{T}_i$ for each $i \in 1..n$.

   Sadly, however, this variant is false.

   Briefly explain why.

   Answer: Because it claims that any type for a record expression will have the same fields as the record (in the same order); but the subsumption rule allows us to "forget" fields in the type that the expression actually mentions and to permute the order and raise the types of the remaining fields.

   Grading scheme: This question was a bit hard to answer because the clear-cut part (the fact that the lemma is problematic because of being false) was already given. Full credit was awarded for any answer that suggested that the possibility of subsumption was not taken into account, or that gave a correct counter-example. 2 points for "sort of correct" but garbled answers. 1 point for very brief or garbled answers that mention the word "subtyping." No points for explaining that the proof of preservation doesn't go through (it does *go through, if we assume the lemma is true).*

Subtyping

7. Suppose we add new types A, B, C, D, and E to the simply typed lambda-calculus with subtyping, along with the following subtyping rules:

$$\mathtt{A} <: \mathtt{B}$$
$$\mathtt{B} <: \mathtt{C}$$
$$\mathtt{D} <: \mathtt{E}$$
$$\mathtt{A} <: \mathtt{E}$$

(a) (6 points) For each type S from the left-hand column and type T from the right-hand column such that S <: T, draw a line connecting S to T.

<u>Choices for S:</u>	<u>Choices for T:</u>
S_1 = {x:D}→{y:A}	T_1 = E
S_2 = {y:B}	T_2 = {x:D}
S_3 = {x:A,y:A}	T_3 = {x:E}→{y:B}
S_4 = D	T_4 = {}
S_5 = {x:B}	T_5 = {y:E}

Answer:

- S_1 *not a subtype of any* T_i
- $S_2 <: T_4$
- $S_3 <: T_4, T_5$
- $S_4 <: T_1$
- $S_5 <: T_4$

Grading scheme: One point off for each missing line; one point off for each incorrect line.

continued...

(b) (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`.

Give the join and meet of each pair of types, where they exist (otherwise write "undefined.")

i. `{x:E, y:C}` and `{x:B, y:B}`

join = `{x:Top,y:C}` meet = `{x:A,y:B}`

ii. `{x:B}` and `{y:B}`

join = `{}` meet = `{x:B, y:B}`

iii. `{x:A}` and `{x:D}`

join = `{x:E}` meet = undefined

Grading scheme: 1 point for each exactly correct answer. No partial credit.

(c) (3 points) How does the join of `{x:A}` and `{x:D}` change if we add the rule `D <: B` to the subtyping judgment?

Answer: It ceases to exist: in the enriched system, the common upper bounds of `{x:A}` and `{x:D}` are `{x:B}`, `{x:C}`, and `{x:E}`; but none of these is a subtype of all the others.

Grading scheme:

- *full credit for "becomes undefined"*
- *no points for giving a join*
- *1 point for saying something like "there are 2 possible joins"*
- *1/2 points deducted for saying "undefined" plus something incorrect*
- *full credit for "unchanged" on papers where 7Biii was (incorrectly) marked as undefined*
- *1 point for {} with a semi-reasonable explanation*

Imperative object encodings

This problem involves an encoding of simple objects in the simply typed lambda-calculus with subtyping, records, references, and fixed points. This calculus is defined for your reference on pages 15 to 21.

8. (20 points) Consider the five following class definitions, in the style of TAPL Chapter 18:

```
R = {};
O = {m:Unit→Unit, n:Unit→Unit, o:Unit→Unit};

classA = λr:R. λself: Unit→O. λ_:Unit.
           {m = λ_:Unit. (self unit).n unit,
            n = λ_:Unit. (self unit).m unit,
            o = λ_:Unit. unit };

classB = λr:R. λself: Unit→O. λ_:Unit.
           let super = classA r self unit in
           {m = λ_:Unit. unit,
            n = super.n,
            o = super.o };

classC = λr:R. λself: Unit→O. λ_:Unit.
           let super = classA r self unit in
           {m = super.m,
            n = λ_:Unit. super.n unit,
            o = super.o };

classD = λr:R. λself: Unit→O. λ_:Unit.
           let super = classA r self unit in
           {m = super.m,
            n = λ_:Unit. super.o unit,
            o = super.o };

classE = λr:R. λself: Unit→O. λ_:Unit.
           let super = classA r self unit in
           {m = (self unit).m,
            n = super.n,
            o = super.o };
```

Each of these classes produces objects with the same type, `O`, which offers three methods. Their state representations are all the trivial record type, `R`. The methods `m`, `n`, and `o` all take `Unit` arguments and return `Unit` results—that is, the only interesting thing about them is whether they diverge or converge. The class `classA` is the "root class" from which the other four are derived.

continued...

Given these definitions, does evaluation of the following expressions converge (yielding `unit`) or diverge? For each one, circle the word "converges" or "diverges," as appropriate. *Answers boxed below.*

`((fix (classA {})) unit).n unit`	converges	[✓ diverges]
`((fix (classA {})) unit).o unit`	[✓ converges]	diverges
`((fix (classB {})) unit).n unit`	[✓ converges]	diverges
`((fix (classB {})) unit).o unit`	[✓ converges]	diverges
`((fix (classC {})) unit).n unit`	converges	[✓ diverges]
`((fix (classC {})) unit).o unit`	[✓ converges]	diverges
`((fix (classD {})) unit).n unit`	[✓ converges]	diverges
`((fix (classD {})) unit).o unit`	[✓ converges]	diverges
`((fix (classE {})) unit).n unit`	converges	[✓ diverges]
`((fix (classE {})) unit).o unit`	converges	[✓ diverges]

Featherweight Java

The definition of Featherweight Java is reproduced on pages 22 to 24.

9. (a) (16 points)

Chapter 3 of TAPL introduced an alternative *big-step* style for presenting evaluation relations, in which each rule shows how to evaluate a term of a particular form "all the way" to a final value, rather than just showing how to make it take a single step. For example, in the lambda-calculus with booleans, the usual small step evaluation rules for `if`...

$$\texttt{if true then } t_2 \texttt{ else } t_3 \longrightarrow t_2$$

$$\texttt{if false then } t_2 \texttt{ else } t_3 \longrightarrow t_3$$

$$\frac{t_1 \longrightarrow t_1'}{\texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3 \longrightarrow \texttt{if } t_1' \texttt{ then } t_2 \texttt{ else } t_3}$$

correspond to the following big-step rules:

$$\frac{t_1 \Downarrow \texttt{true} \qquad t_2 \Downarrow v_2}{\texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3 \Downarrow v_2}$$

$$\frac{t_1 \Downarrow \texttt{false} \qquad t_3 \Downarrow v_3}{\texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3 \Downarrow v_3}$$

Write down a big step evaluation relation for Featherweight Java. (You'll need four inference rules.)

Answer:

$$\frac{t_0 \Downarrow \texttt{new } C(\overline{v}) \qquad C <: D}{(D)t_0 \Downarrow \texttt{new } C(\overline{v})}$$

$$\frac{t_0 \Downarrow \texttt{new } C(\overline{v}) \qquad \textit{fields}(C) = \overline{C}\ \overline{f}}{t_0.f_i \longrightarrow v_i}$$

$$\frac{t_0 \Downarrow \texttt{new } C(\overline{v}) \qquad \textit{mbody}(m, C) = (\overline{x}, s) \qquad \overline{t} \Downarrow \overline{u} \qquad [\overline{x} \mapsto \overline{u}, \texttt{this} \mapsto \texttt{new } C(\overline{v})]s \Downarrow v}{t_0.m(\overline{t}) \longrightarrow v}$$

$$\frac{\overline{t} \Downarrow \overline{v}}{\texttt{new } C(\overline{t}) \Downarrow \texttt{new } C(\overline{v})}$$

Grading scheme:

- *4 points per rule.*
- *For all rules:*
  - *-2 points for using a value instead of a term on the LHS.*
  - *-4 points for a loop in the operational semantics.*
  - *+/- points for being very close or very wrong.*
- *Individual rules: Cast:*
  - *-1 for no subtyping constraint.*
  - *-2 for no class mentioned in the new expression.*
  - *-1 for evaluating to wrong type.*
- *Prj.:*
  - *-2 for no class mentioned in the record evaluation.*

  - *-1 for no constraint that project field exists*
  - *-3 for not projecting from the record*
- *Meth:*
  - *-2 for no substitution at all*
  - *-3 for substitution in the conclusion*

(b) (5 points) State the preservation theorem for the big-step variant of FJ.

Answer: If $\Gamma \vdash$ `t : C` *and* `t` $\Downarrow$ `v`, *then* $\Gamma \vdash$ `v : D` *for some* `D <: C`.

Grading scheme:

- *No deduction for omitting* Γ
- *Either* `v` *or* `new C(v)` *accepted for the value*
- *-2 for omitting* `t` $\Downarrow$ `v`
- *-3 for neglecting the fact that the new type can be a* subtype *of old type*
- *hand-wavy english, correct but too vague = 1/5*
- *weird stuff that looks a tiny bit right = 1/5*

For reference: Simply typed lambda calculus with records (and `Nat`, `Bool`, and `Unit`)

Syntax

`t` ::=		*terms*
	`unit`	*constant* `unit`
	$\{\mathtt{l}_i\mathtt{=t}_i\ ^{i\in 1..n}\}$	*record*
	`t.l`	*projection*
	`x`	*variable*
	`λx:T.t`	*abstraction*
	`t t`	*application*
	`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*
	`unit`	*constant* `unit`
	$\{\mathtt{l}_i\mathtt{=v}_i\ ^{i\in 1..n}\}$	*record value*
	`λx:T.t`	*abstraction value*
	`true`	*true value*
	`false`	*false value*
	`nv`	*numeric value*
`T` ::=		*types*
	`Unit`	*unit type*
	$\{\mathtt{l}_i\mathtt{:T}_i\ ^{i\in 1..n}\}$	*type of records*
	`Bool`	*type of booleans*
	`Nat`	*type of natural numbers*
	`T→T`	*type of functions*
Γ ::=		*contexts*
	$\varnothing$	*empty context*
	Γ, `x:T`	*term variable binding*
`nv` ::=		*numeric values*
	`0`	*zero value*
	`succ nv`	*successor value*

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

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

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

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

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

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

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

$$\mathtt{if\ true\ then\ t_2\ else\ t_3} \longrightarrow \mathtt{t}_2 \qquad \text{(E-IfTrue)}$$

$$\mathtt{if\ false\ then\ t_2\ else\ t_3} \longrightarrow \mathtt{t}_3 \qquad \text{(E-IfFalse)}$$

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{if\ t_1\ then\ t_2\ else\ t_3} \longrightarrow \mathtt{if\ t_1'\ then\ t_2\ else\ t_3}} \qquad \text{(E-If)}$$

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

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

$$\mathtt{pred\ (succ\ nv_1)} \longrightarrow \mathtt{nv}_1 \qquad \text{(E-PredSucc)}$$

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

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

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

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

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

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

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

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

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

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

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{Bool} \qquad \Gamma \vdash \mathtt{t}_2 : \mathtt{T} \qquad \Gamma \vdash \mathtt{t}_3 : \mathtt{T}}{\Gamma \vdash \mathtt{if\ t_1\ then\ t_2\ else\ t_3} : \mathtt{T}} \qquad \text{(T-If)}$$

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

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{Nat}}{\Gamma \vdash \mathtt{succ}\ \mathtt{t}_1 : \mathtt{Nat}} \tag{T-Succ}$$

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{Nat}}{\Gamma \vdash \mathtt{pred}\ \mathtt{t}_1 : \mathtt{Nat}} \tag{T-Pred}$$

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{Nat}}{\Gamma \vdash \mathtt{iszero}\ \mathtt{t}_1 : \mathtt{Bool}} \tag{T-IsZero}$$

$$\frac{\mathtt{x}{:}\mathtt{T} \in \Gamma}{\Gamma \vdash \mathtt{x} : \mathtt{T}} \tag{T-Var}$$

$$\frac{\Gamma, \mathtt{x}{:}\mathtt{T}_1 \vdash \mathtt{t}_2 : \mathtt{T}_2}{\Gamma \vdash \lambda \mathtt{x}{:}\mathtt{T}_1.\mathtt{t}_2 : \mathtt{T}_1 \to \mathtt{T}_2} \tag{T-Abs}$$

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{T}_{11} \to \mathtt{T}_{12} \qquad \Gamma \vdash \mathtt{t}_2 : \mathtt{T}_{11}}{\Gamma \vdash \mathtt{t}_1\ \mathtt{t}_2 : \mathtt{T}_{12}} \tag{T-App}$$

For reference: Subtyping

New syntactic forms

$\mathtt{T}$	$::=$	$\ldots$	*types*
		$\mathtt{Top}$	*maximum type*

New subtyping rules $\boxed{\mathtt{S} <: \mathtt{T}}$

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

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

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

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

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

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

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

New typing rules $\boxed{\Gamma \vdash \mathtt{t} : \mathtt{T}}$

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

For reference: References, Fix, and `let`

New syntactic forms

`t`	::=	...	*terms*
		`ref t`	*reference creation*
		`!t`	*dereference*
		`t:=t`	*assignment*
		l	*store location*
		`let x=t in t`	*let binding*
		`fix t`	*fixed point of* `t`
`v`	::=	...	*values*
		l	*store location*
`T`	::=	...	*types*
		`Ref T`	*type of reference cells*
μ	::=	...	*stores*
		$\emptyset$	*empty store*
		$\mu, l = \mathtt{v}$	*location binding*
Σ	::=	...	*store typings*
		$\emptyset$	*empty store typing*
		$\Sigma, l{:}\mathtt{T}$	*location typing*

New evaluation rules $\boxed{\mathtt{t} \mid \mu \longrightarrow \mathtt{t}' \mid \mu'}$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$\mathtt{pred\ (succ\ nv}_1\mathtt{)} \mid \mu \longrightarrow \mathtt{nv}_1 \mid \mu \qquad \text{(E-PredSucc)}$$

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

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

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

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

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

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

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

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

New subtyping rules $\boxed{\mathtt{S} <: \mathtt{T}}$

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

New typing rules $\boxed{\Gamma \mid \Sigma \vdash \mathtt{t} : \mathtt{T}}$

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

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

$$\frac{\Gamma | \Sigma \vdash \mathtt{t}_1 : \mathsf{T}_{11} \to \mathsf{T}_{12} \qquad \Gamma | \Sigma \vdash \mathtt{t}_2 : \mathsf{T}_{11}}{\Gamma | \Sigma \vdash \mathtt{t}_1\ \mathtt{t}_2 : \mathsf{T}_{12}} \tag{T-App}$$

$$\Gamma | \Sigma \vdash \mathtt{unit} : \mathtt{Unit} \tag{T-Unit}$$

$$\frac{\Sigma(l) = \mathsf{T}_1}{\Gamma \mid \Sigma \vdash l : \mathtt{Ref}\ \mathsf{T}_1} \tag{T-Loc}$$

$$\frac{\Gamma \mid \Sigma \vdash \mathtt{t}_1 : \mathsf{T}_1}{\Gamma \mid \Sigma \vdash \mathtt{ref}\ \mathtt{t}_1 : \mathtt{Ref}\ \mathsf{T}_1} \tag{T-Ref}$$

$$\frac{\Gamma \mid \Sigma \vdash \mathtt{t}_1 : \mathtt{Ref}\ \mathsf{T}_{11}}{\Gamma \mid \Sigma \vdash\ !\mathtt{t}_1 : \mathsf{T}_{11}} \tag{T-Deref}$$

$$\frac{\Gamma \mid \Sigma \vdash \mathtt{t}_1 : \mathtt{Ref}\ \mathsf{T}_{11} \qquad \Gamma \mid \Sigma \vdash \mathtt{t}_2 : \mathsf{T}_{11}}{\Gamma \mid \Sigma \vdash \mathtt{t}_1 \mathtt{:=} \mathtt{t}_2 : \mathtt{Unit}} \tag{T-Assign}$$

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathsf{T}_1 \qquad \Gamma, \mathtt{x}{:}\mathsf{T}_1 \vdash \mathtt{t}_2 : \mathsf{T}_2}{\Gamma \vdash \mathtt{let}\ \mathtt{x}\mathtt{=}\mathtt{t}_1\ \mathtt{in}\ \mathtt{t}_2 : \mathsf{T}_2} \tag{T-Let}$$

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathsf{T}_1 \to \mathsf{T}_1}{\Gamma \vdash \mathtt{fix}\ \mathtt{t}_1 : \mathsf{T}_1} \tag{T-Fix}$$

For reference: FJ

Syntax

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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