CIS 500 — Software Foundations
Final Exam

December 20, 2006

Name or WPE-I Id: ____________________

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

Instructions

- This is a closed-book exam.
- You have 120 minutes to answer all of the questions. The entire exam is worth 120 points.
- Questions vary significantly in difficulty, and the point value of a given question is not always exactly proportional to its difficulty. Do not spend too much time on any one question.
- Partial credit will be given. All correct answers are short. The back side of each page and the companion handout may be used as scratch paper.
- Good luck!

Inductively Defined Relations

Define the syntactic categories of *blobs* (written x) and *counts* (written y) as follows:

$$\begin{array}{lcl} x & ::= & \sharp \\ & & \flat \\ & & \natural \\ & & x \cdot x \\ \\ y & ::= & 0 \\ & & +y \\ & & -y \end{array}$$

That is, a blob is a tree whose leaves are labeled $\sharp$, $\natural$, or $\flat$; a count is a sequence of $+$s and $-$s ending in 0.

Now define the relation "accumulating x onto y yields y'," written $x \curvearrowright y \;\triangleright\; y'$, as the least three-place relation closed under the following rules:

$$\sharp \curvearrowright y \;\triangleright\; +y \qquad \text{(Sharp)}$$

$$\flat \curvearrowright y \;\triangleright\; -y \qquad \text{(Flat)}$$

$$\natural \curvearrowright y \;\triangleright\; y \qquad \text{(Natural)}$$

$$\frac{x_1 \curvearrowright y \;\triangleright\; y'' \qquad x_2 \curvearrowright y'' \;\triangleright\; y'}{x_1 \cdot x_2 \curvearrowright y \;\triangleright\; y'} \qquad \text{(Dot)}$$

$$\frac{x_2 \cdot x_1 \curvearrowright y \;\triangleright\; y'}{x_1 \cdot x_2 \curvearrowright y \;\triangleright\; y'} \qquad \text{(Swap)}$$

Notice that the result of accumulating x onto y always has y itself as a suffix, and that it additionally includes one $+$ for every $\sharp$ in x and one $-$ for every $\flat$ in x. The middle component of the relation, y, is analogous to the "accumulator parameter" sometimes used by tail-recursive OCaml functions. The Swap rule introduces some flexibility in the order of $+$s and $-$s in y', relative to the positions of $\sharp$s and $\flat$s in x.

1. (6 points) Are the following statements derivable? (Write YES or NO for each.)

   (a) $\sharp \cdot (\natural \cdot \natural) \curvearrowright 0 \;\; \triangleright \;\; +0$

   (b) $\sharp \cdot (\sharp \cdot \flat) \curvearrowright 0 \;\; \triangleright \;\; + - +0$

   (c) $(\sharp \cdot \natural) \cdot (\flat \cdot \natural) \curvearrowright +0 \;\; \triangleright \;\; + + -0$

2. (20 points) Write a careful inductive proof of the following fact. Make sure to explictly mention every step in the proof (use of an assumption, use of the induction hypothesis, use of one of the inference rules, etc.).

Fact: For every x there is some y' such that $x \curvearrowright 0 \ \triangleright \ y'$

Untyped Lambda-Calculus

The following problem concerns the untyped lambda-calculus. This system is summarized on page 1 of the companion handout.

3. (6 points) Recall the definitions of observational and behavioral equivalence from the lecture notes:

- Two terms `s` and `t` are *observationally equivalent* iff either both are normalizable (i.e., they reach a normal form after a finite number of evaluation steps) or both are divergent.
- Terms `s` and `t` are *behaviorally equivalent* iff, for every finite sequence of values $\texttt{v}_1, \texttt{v}_2, \ldots, \texttt{v}_n$ (including the empty sequence), the applications

$$\texttt{s}\ \texttt{v}_1\ \texttt{v}_2\ \ldots\ \texttt{v}_n$$

and

$$\texttt{t}\ \texttt{v}_1\ \texttt{v}_2\ \ldots\ \texttt{v}_n$$

are observationally equivalent.

For each of the following pairs of terms, write *YES* if the terms are behaviorally equivalent and *NO* if they are not.

(a) $(\lambda\texttt{x}.\ \lambda\texttt{y}.\ \texttt{x}\ (\lambda\texttt{z}.\ \texttt{z})\ \texttt{y})$
and $(\lambda\texttt{x}.\ \lambda\texttt{y}.\ (\lambda\texttt{z}.\ \texttt{z})\ \texttt{x}\ \texttt{y})$

(b) $(\lambda\texttt{s}.\ \lambda\texttt{z}.\ \texttt{s}\ (\texttt{s}\ \texttt{z}))$
and $(\lambda\texttt{n}.\ \lambda\texttt{s}.\ \lambda\texttt{z}.\ \texttt{s}\ (\texttt{n}\ \texttt{s}\ \texttt{z}))\ (\lambda\texttt{s}.\ \lambda\texttt{z}.\ \texttt{s}\ \texttt{z})$

(c) $(\lambda\texttt{x}.\ \texttt{x}\ \texttt{x})\ (\lambda\texttt{x}.\ \texttt{x}\ \texttt{x})$
and $\texttt{Z}\ (\lambda\texttt{g}.\ \lambda\texttt{h}.\ \texttt{h})\ (\lambda\texttt{z}.\ \texttt{z})$
where $\texttt{Z} = (\lambda\texttt{f}.\ \lambda\texttt{y}.\ (\lambda\texttt{x}.\ \texttt{f}\ (\lambda\texttt{y}.\ \texttt{x}\ \texttt{x}\ \texttt{y}))\ (\lambda\texttt{x}.\ \texttt{f}\ (\lambda\texttt{y}.\ \texttt{x}\ \texttt{x}\ \texttt{y}))\ \texttt{y})$, as in lecture notes

Subtyping

The following problems concern the simply typed lambda-calculus with subtyping (and records, variants, and references). This system is summarized on page 2 of the companion handout.

4. (10 points) Circle T or F for each of the following statements.

   (a) There is an infinite descending chain of distinct types in the subtype relation—that is, an infinite sequence of types $\mathtt{S}_0$, $\mathtt{S}_1$, etc., such that all the $\mathtt{S}_i$'s are different and each $\mathtt{S}_{i+1}$ is a subtype of $\mathtt{S}_i$.

   T F

   (b) There is an infinite ascending chain of distinct types in the subtype relation—that is, an infinite sequence of types $\mathtt{S}_0$, $\mathtt{S}_1$, etc., such that all the $\mathtt{S}_i$'s are different and each $\mathtt{S}_{i+1}$ is a supertype of $\mathtt{S}_i$.

   T F

   (c) There exists a type that is a subtype of every other type.

   T F

   (d) There exists a record type that is a subtype of every other record type.

   T F

   (e) There exists a variant type that is a subtype of every other variant type.

   T F

5. (15 points) The standard subtyping rule for references is:

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

Suppose we drop the first premise so that `Ref` becomes a covariant type constructor:

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

Indicate whether each of the following properties remains true (write "TRUE") or becomes false (write "FALSE"), and briefly explain why.

(a) *Progress*: Suppose `t` is a closed, well-typed term (that is, $\emptyset|\Sigma \vdash \mathtt{t} : \mathtt{T}$ for some `T` and Σ). Then either `t` is a value or else, for any store μ such that $\emptyset|\Sigma \vdash \mu$, there is some term $\mathtt{t}'$ and store μ' with $\mathtt{t}|\mu \longrightarrow \mathtt{t}'|\mu'$.

(b) *Preservation*: If

$\Gamma|\Sigma \vdash \mathtt{t} : \mathtt{T}$
$\Gamma|\Sigma \vdash \mu$
$\mathtt{t}|\mu \longrightarrow \mathtt{t}'|\mu'$

then, for some $\Sigma' \supseteq \Sigma$,

$\Gamma|\Sigma' \vdash \mathtt{t}' : \mathtt{T}$
$\Gamma|\Sigma' \vdash \mu'$.

(c) *Existence of joins*: For every pair of types `S` and `T` there is some type `J` such that `S` and `T` are both subtypes of `J` and such that, for any other type `U`, if `S` and `T` are both subtypes of `U`, then `J` is a subtype of `U`.

Object Encodings in Lambda-Calculus

The questions in this section are based the following small class hierarchy encoded in lambda-calculus. (Note that this encoding is in the simpler style of section 18.11 of TAPL; it does not incorporate the refinements for improved efficiency discussed at the very end of the chapter, in 18.12.)

```
/* A couple of miscellaneous helper functions -- "not" on booleans... */
not = λb:Bool. if b then false else true;
/* and a comparison function for numbers: */
leq =
  fix (λf:Nat→Nat→Bool.
         λm:Nat. λn:Nat.
           if iszero m then true
           else if iszero n then false
           else f (pred m) (pred n));

/* The interface type of "pair objects": */
Pair = {set1:Nat→Unit, set2:Nat→Unit, lessoreq:Unit→Bool, greater:Unit→Bool};

/* The internal representation of "pair objects": */
PairRep = {x1: Ref Nat, x2:Ref Nat};

/* A class of "abstract pair objects."  Note that the lessoreq and
   greater methods call each other recursively. */
absPairClass =
  λr:PairRep.
  λself: Unit→Pair.
    λ_:Unit.
      {set1 = λi:Nat.  r.x1:=i,
       set2 = λi:Nat.  r.x2:=i,
       lessoreq = λ_:Unit. not ((self unit).greater unit),
       greater = λ_:Unit. not ((self unit).lessoreq unit)};

/* A function that creates a new abstract pair object: */
newAbsPair =
  λ_:Unit. let r = {x1=ref 0, x2=ref 0} in
              fix (absPairClass r) unit;

/* A subclass that overrides the lessoreq method: */
pairClass =
  λr:PairRep.
  λself: Unit→Pair.
    λ_:Unit.
      let super = absPairClass r self unit in
      {set1 = super.set1,
       set2 = super.set2,
       lessoreq = λ_:Unit. leq (!(r.x1)) (!(r.x2)),
       greater = super.greater};

/* A function that creates a new pair object: */
newPair =
  λ_:Unit. let r = {x1=ref 0, x2=ref 0} in
              fix (pairClass r) unit;
```

6. (6 points) Circle T or F for each of the following statements.

(a) The expression `newAbsPair unit` diverges.

T F

(b) The expression `(newAbsPair unit).set1 5` diverges.

T F

(c) The expression `(newAbsPair unit).greater unit` diverges.

T F

(d) The expression `newPair unit` diverges.

T F

(e) The expression `(newPair unit).set1 5` yields `unit`.

T F

(f) The expression `(newPair unit).greater unit` yields `false`.

T F

7. (16 points) Write another class `myPairClass` that uses `pairClass` as its superclass and that adds one more method, called `setSmaller`, that calls the `lessoreq` method to determine which field is smaller and then calls either the `set1` or the `set2` method to update the value of this field. (Your new method should not use `:=`, `!`, or numeric comparison directly.) You do not need to write the `newMyPair` function—just the class.

```
MyPair = {set1:Nat→Unit, set2:Nat→Unit,
          lessoreq:Unit→Bool, greater:Unit→Bool,
          setSmaller:Nat→Unit};

myPairClass =
```

Featherweight Java with Exceptions

The problems in this section deal with an extension of FJ with exceptions. The definition of the original FJ is given for reference on page 6 of the companion handout.

The full syntax of terms in the extended language, including two new syntactic forms for raising and handling `error`s, is:

`t ::=`		
	`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*
	`error`	*run-time error*
	`try t with t`	*trap errors*

(Note that we are adding the simplest form of exceptions here—exceptions are just the term `error`, with no additional value carried along.)

The typing rules for `error` and `try...with...` are standard:

$$\Gamma \vdash \texttt{error : C} \qquad \text{(T-Error)}$$

$$\frac{\Gamma \vdash \texttt{t}_1 \texttt{ : C} \qquad \Gamma \vdash \texttt{t}_2 \texttt{ : C}}{\Gamma \vdash \texttt{try t}_1 \texttt{ with t}_2 \texttt{ : C}} \qquad \text{(T-Try)}$$

Having added exceptions to the system, we no longer need to define failing casts as stuck terms (as in original FJ); instead, we can make a failing cast raise an exception:

$$\frac{\texttt{C} \not<: \texttt{D}}{\texttt{(D)(new C(}\overline{\texttt{v}}\texttt{))} \longrightarrow \texttt{error}} \qquad \text{(E-BadCast)}$$

The other new evaluation rules follow the same pattern as in $\lambda_\rightarrow$ with exceptions: `error` "percolates up" through the other term constructors, aborting their evaluation as it goes. For example:

$$\texttt{error.m(}\overline{\texttt{u}}\texttt{)} \longrightarrow \texttt{error} \qquad \text{(E-InvkError)}$$

$$\texttt{(new C(}\overline{\texttt{v}}\texttt{)).m(u}_1\texttt{...u}_{i-1}\texttt{,error,t}_{i+1}\texttt{...t}_n\texttt{)} \longrightarrow \texttt{error} \qquad \text{(E-InvkErrorArg)}$$

$$\texttt{try error with t}_2 \longrightarrow \texttt{t}_2 \qquad \text{(E-TryError)}$$

8. (10 points) What other evaluation rules do we need to add to complete the definition?

9. (6 points) State an appropriate progress theorem for the extended language. (Do not prove it.)

10. (18 points) The statement of the preservation theorem for FJ with exceptions is exactly the same as for ordinary FJ:

Theorem: If $\Gamma \vdash \mathtt{t} : \mathtt{C}$ and $\mathtt{t} \longrightarrow \mathtt{t}'$, then $\Gamma \vdash \mathtt{t}' : \mathtt{C}'$ for some $\mathtt{C}' <: \mathtt{C}$.

Fill in the blanks in the following proof of this theorem. Make sure to explictly mention every step required in the proof (use of an assumption, use of the induction hypothesis, use of a typing or evaluation rule, etc.).

Proof: By induction on a derivation of $\mathtt{t} \longrightarrow \mathtt{t}'$, with a case analysis on the final rule. (Just three of the cases are given here; we are eliding several others.)

Case E-BadCast: $\quad \mathtt{t} = \mathtt{(D)(new\ B(\overline{v}))} \quad \mathtt{t}' = \mathtt{error} \quad \mathtt{B} \not<: \mathtt{D}$

Case E-TryError: $\quad \mathtt{t} = \mathtt{try\ error\ with\ t_2} \quad \mathtt{t}' = \mathtt{t_2}$

Case E-CastNew: $\quad \mathtt{t} = \mathtt{(D)(new\ C_0(\overline{v}))} \quad \mathtt{C_0} <: \mathtt{D} \quad \mathtt{t}' = \mathtt{new\ C_0(\overline{v})}$

Polymorphism

The following problem concerns the polymorphic lambda-calculus (with a primitive `fix` construct and booleans). This system is summarized on page 9 of the companion handout.

11. (7 points) Suppose (following the example in Chapter 23 of TAPL and in the lecture notes) that our language is also equipped with a type constructor `List` and the following term constructors for the usual list manipulation primitives.

```
nil   : ∀ X. List X
cons  : ∀ X. X → List X → List X
isnil : ∀ X. List X → Bool
head  : ∀ X. List X → X
tail  : ∀ X. List X → List X
```

Complete the following definition of a `mapfilter` function on lists by filling in the missing type parameters (`mapfilter` is an "all in one" combination of `map` and `filter`—it filters a list using the boolean function `test` and applies `f` to each element of the resulting list). All necessary type parameters are indicated with blanks, which you are to fill in.

```
mapfilter = λX. λY. λtest:X → Bool. λf:X → Y

 (fix (λrec:List X → List Y.

           λxs:List X.

               if isnil  [_____]  xs then

                 nil  [_____]

               else if test (head  [_____]  xs) then

                 cons  [_____]  (f (head  [_____]  xs)) (rec (tail  [_____]  xs))

               else

                 rec (tail  [_____]  xs)))
```

Companion handout

Full definitions of the systems used in the exam

Untyped Lambda-calculus

Syntax

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

Evaluation

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

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

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

Simply-typed lambda calculus with subtyping (and records and variants)

Syntax

$\mathtt{t}$ ::=		*terms*
	$\mathtt{x}$	*variable*
	$\lambda\mathtt{x}:\mathtt{T}.\mathtt{t}$	*abstraction*
	$\mathtt{t}\ \mathtt{t}$	*application*
	$\{\mathtt{l}_i=\mathtt{t}_i\ ^{i\in 1..n}\}$	*record*
	$\mathtt{t.l}$	*projection*
	unit	*constant* unit
	ref t	*reference creation*
	!t	*dereference*
	t:=t	*assignment*
	l	*store location*
	<l=t> (no as)	*tagging*
	case t of $\mathtt{<l}_i\mathtt{=x}_i\mathtt{>}\Rightarrow\mathtt{t}_i\ ^{i\in 1..n}$	*case*
$\mathtt{v}$ ::=		*values*
	$\lambda\mathtt{x}:\mathtt{T}.\mathtt{t}$	*abstraction value*
	$\{\mathtt{l}_i=\mathtt{v}_i\ ^{i\in 1..n}\}$	*record value*
	unit	*constant* ***unit***
	l	*store location*
$\mathtt{T}$ ::=		*types*
	$\{\mathtt{l}_i:\mathtt{T}_i\ ^{i\in 1..n}\}$	*type of records*
	Top	*maximum type*
	$\mathtt{T}\rightarrow\mathtt{T}$	*type of functions*
	Unit	*unit type*
	Ref T	*type of reference cells*
	$\mathtt{<l}_i:\mathtt{T}_i\ ^{i\in 1..n}\mathtt{>}$	*type of variants*
Γ ::=		*type environments*
	$\emptyset$	*empty type env.*
μ ::=		*stores*
	$\emptyset$	*empty store*
	$\mu, l = \mathtt{v}$	*location binding*
Σ ::=		*store typings*
	$\emptyset$	*empty store typing*
	$\Sigma, l:\mathtt{T}$	*location typing*

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

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

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

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

$$\{\mathtt{l}_i\mathtt{=}\mathtt{v}_i\ ^{i\in 1..n}\}.\mathtt{l}_j|\mu \longrightarrow \mathtt{v}_j|\mu \tag{E-ProjRcd}$$

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

$$\frac{\mathtt{t}_1|\mu \longrightarrow \mathtt{t}_1'|\mu'}{\mathtt{ref}\ \mathtt{t}_1|\mu \longrightarrow \mathtt{ref}\ \mathtt{t}_1'|\mu'} \tag{E-Ref}$$

$$\frac{\mu(l) = \mathtt{v}}{\mathtt{!}l|\mu \longrightarrow \mathtt{v}|\mu} \tag{E-DerefLoc}$$

$$\frac{\mathtt{t}_1|\mu \longrightarrow \mathtt{t}_1'|\mu'}{\mathtt{!}\mathtt{t}_1|\mu \longrightarrow \mathtt{!}\mathtt{t}_1'|\mu'} \tag{E-Deref}$$

$$l\mathtt{:=}\mathtt{v}_2|\mu \longrightarrow \mathtt{unit}|[l \mapsto \mathtt{v}_2]\mu \tag{E-Assign}$$

$$\frac{\mathtt{t}_1|\mu \longrightarrow \mathtt{t}_1'|\mu'}{\mathtt{t}_1\mathtt{:=}\mathtt{t}_2|\mu \longrightarrow \mathtt{t}_1'\mathtt{:=}\mathtt{t}_2|\mu'} \tag{E-Assign1}$$

$$\frac{\mathtt{t}_2|\mu \longrightarrow \mathtt{t}_2'|\mu'}{\mathtt{v}_1\mathtt{:=}\mathtt{t}_2|\mu \longrightarrow \mathtt{v}_1\mathtt{:=}\mathtt{t}_2'|\mu'} \tag{E-Assign2}$$

$$\mathtt{case}\ (\mathtt{<}\mathtt{l}_j\mathtt{=}\mathtt{v}_j\mathtt{>}\ \mathtt{as}\ \mathtt{T})\ \mathtt{of}\ \mathtt{<}\mathtt{l}_i\mathtt{=}\mathtt{x}_i\mathtt{>}\Rightarrow\mathtt{t}_i\ ^{i\in 1..n}|\mu \longrightarrow [\mathtt{x}_j \mapsto \mathtt{v}_j]\mathtt{t}_j|\mu \tag{E-CaseVariant}$$

$$\frac{\mathtt{t}_0|\mu \longrightarrow \mathtt{t}_0'|\mu'}{\mathtt{case}\ \mathtt{t}_0\ \mathtt{of}\ \mathtt{<}\mathtt{l}_i\mathtt{=}\mathtt{x}_i\mathtt{>}\Rightarrow\mathtt{t}_i\ ^{i\in 1..n}|\mu \longrightarrow \mathtt{case}\ \mathtt{t}_0'\ \mathtt{of}\ \mathtt{<}\mathtt{l}_i\mathtt{=}\mathtt{x}_i\mathtt{>}\Rightarrow\mathtt{t}_i\ ^{i\in 1..n}|\mu'} \tag{E-Case}$$

$$\frac{\mathtt{t}_i|\mu \longrightarrow \mathtt{t}_i'|\mu'}{\mathtt{<}\mathtt{l}_i\mathtt{=}\mathtt{t}_i\mathtt{>}\ \mathtt{as}\ \mathtt{T}|\mu \longrightarrow \mathtt{<}\mathtt{l}_i\mathtt{=}\mathtt{t}_i'\mathtt{>}\ \mathtt{as}\ \mathtt{T}|\mu'} \tag{E-Variant}$$

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

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

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

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

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

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

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

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

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

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

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

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

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{T}_1}{\Gamma \vdash \mathtt{<l}_1\mathtt{=t}_1\mathtt{>} : \mathtt{<l}_1\mathtt{:T}_1\mathtt{>}} \tag{T-Variant}$$

$$\frac{\begin{array}{c}\Gamma \vdash \mathtt{t}_0 : \mathtt{<l}_i\mathtt{:T}_i\ {}^{i\in 1..n}\mathtt{>} \\ \text{for each } i \quad \Gamma, \mathtt{x}_i\mathtt{:T}_i \vdash \mathtt{t}_i : \mathtt{T}\end{array}}{\Gamma \vdash \mathtt{case}\ \mathtt{t}_0\ \mathtt{of}\ \mathtt{<l}_i\mathtt{=x}_i\mathtt{>}\Rightarrow\mathtt{t}_i\ {}^{i\in 1..n} : \mathtt{T}} \tag{T-Case}$$

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

$$\mathtt{S} <: \mathtt{S} \tag{S-Refl}$$

$$\frac{\mathtt{S} <: \mathtt{U} \qquad \mathtt{U} <: \mathtt{T}}{\mathtt{S} <: \mathtt{T}} \tag{S-Trans}$$

$$\mathtt{S} <: \mathtt{Top} \tag{S-Top}$$

$$\frac{\mathtt{T}_1 <: \mathtt{S}_1 \qquad \mathtt{S}_2 <: \mathtt{T}_2}{\mathtt{S}_1 \rightarrow \mathtt{S}_2 <: \mathtt{T}_1 \rightarrow \mathtt{T}_2} \tag{S-Arrow}$$

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

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

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

$$\frac{\mathtt{T}_1 <: \mathtt{S}_1 \qquad \mathtt{S}_1 <: \mathtt{T}_1}{\mathtt{Ref}\ \mathtt{S}_1 <: \mathtt{Ref}\ \mathtt{T}_1} \tag{S-Ref}$$

$$\mathtt{<l}_i\mathtt{:T}_i\ {}^{i\in 1..n}\mathtt{>} \quad <: \quad \mathtt{<l}_i\mathtt{:T}_i\ {}^{i\in 1..n+k}\mathtt{>} \tag{S-VariantWidth}$$

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

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

Featherweight Java

Syntax

$\texttt{CL}$ ::=		*class declarations*
	$\texttt{class C extends C \{}\overline{\texttt{C}}\ \overline{\texttt{f}}\texttt{; K }\overline{\texttt{M}}\texttt{\}}$	
$\texttt{K}$ ::=		*constructor declarations*
	$\texttt{C(}\overline{\texttt{C}}\ \overline{\texttt{f}}\texttt{) \{super(}\overline{\texttt{f}}\texttt{); this.}\overline{\texttt{f}}\texttt{=}\overline{\texttt{f}}\texttt{;\}}$	
$\texttt{M}$ ::=		*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*
	$\texttt{t.m(}\overline{\texttt{t}}\texttt{)}$	*method invocation*
	$\texttt{new C(}\overline{\texttt{t}}\texttt{)}$	*object creation*
	$\texttt{(C) t}$	*cast*
$\texttt{v}$ ::=		*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 $\boxed{\mathit{fields}(\texttt{C}) = \overline{\texttt{C}}\ \overline{\texttt{f}}}$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Polymorphic Lambda-Calculus (with $\texttt{fix}$ and booleans)

Syntax

$\texttt{t}$ ::=		*terms*
	$\texttt{x}$	*variable*
	$\lambda\texttt{x:T.t}$	*abstraction*
	$\texttt{t t}$	*application*
	$\texttt{let x=t in t}$	*let binding*
	$\texttt{fix t}$	*fixed point of* $\texttt{t}$
	$\texttt{true}$	*constant true*
	$\texttt{false}$	*constant false*
	$\texttt{if t then t else t}$	*conditional*
	$\lambda\texttt{X.t}$	*type abstraction*
	$\texttt{t [T]}$	*type application*
$\texttt{v}$ ::=		*values*
	$\lambda\texttt{x:T.t}$	*abstraction value*
	$\texttt{true}$	*true value*
	$\texttt{false}$	*false value*
	$\lambda\texttt{X.t}$	*type abstraction value*
$\texttt{T}$ ::=		*types*
	$\texttt{Bool}$	*type of booleans*
	$\texttt{X}$	*type variable*
	$\texttt{T}\to\texttt{T}$	*type of functions*
	$\forall\texttt{X.T}$	*universal type*
Γ ::=		*type environments*
	$\emptyset$	*empty type env.*
	$\Gamma, \texttt{X}$	*type variable binding*

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

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

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

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

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

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

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

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

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

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

$$(\lambda \mathtt{X}.\mathtt{t}_{12})\ [\mathtt{T}_2] \longrightarrow [\mathtt{X} \mapsto \mathtt{T}_2]\mathtt{t}_{12} \qquad \text{(E-TappTabs)}$$

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

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

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

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

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

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

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

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

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

$$\frac{\Gamma, \mathtt{X} \vdash \mathtt{t}_2 : \mathtt{T}_2}{\Gamma \vdash \lambda \mathtt{X}.\mathtt{t}_2 : \forall \mathtt{X}.\mathtt{T}_2} \qquad \text{(T-TAbs)}$$

$$\frac{\Gamma \vdash \mathtt{t}_1 : \forall \mathtt{X}.\mathtt{T}_{12}}{\Gamma \vdash \mathtt{t}_1\ [\mathtt{T}_2] : [\mathtt{X} \mapsto \mathtt{T}_2]\mathtt{T}_{12}} \qquad \text{(T-TApp)}$$