1 Second-Order Curry-Howard Correspondence

The well-known *Curry-Howard correspondence* describes a mapping between type theory and logic: propositions correspond to types and proofs correspond to programs. This correspondence is usually formulated only for *intuitionistic logics* (IL), in which the *law of excluded middle* (LEM) or equivalently the *law of double negation* (DNE) does not hold. Concretely, the following propositions are not provable in IL, thus by the correspondence there exists no programs that prove them:

- LEM: $\forall P.P \vee \neg P$
- DNE: $\forall P.\neg\neg P \to P$

This problem is about proving that intuitionistic logic with the law of excluded middle is equivalent to intuitionistic logic with the law of double negation, that is $IL + LEM = IL + DNE$.

Curry-Howard: Negation. In intuitionistic logic, $\neg P$ is the same as $P \to \bot$, where $\bot$ means absurdity. So the proposition $\neg\neg P$ is interpreted as $(\mathtt{P} \to \bot) \to \bot$. We assume absurdity $\bot$ corresponds to the type $\bot$ (or `Bot`) in types, which is not inhabited. The following program `explode` is provided:

$$\emptyset \vdash \mathtt{explode}{:}\forall \mathtt{P}.\bot \to \mathtt{P}$$

The program `explode` has the type $\mathtt{P}.\bot \to \mathtt{P}$ under the empty environment. Logically, it says that from absurdity any proposition can be derived, which corresponds to a well-known principle in logic.

Curry-Howard: Universal quantification and System F. A second-order proposition of form $\forall P.T$ corresponds to a type in System F with pairs and sums. It can be proved by a term that has that type under the empty environment. For example, the proposition $\forall P.P \to P$ translates to the type $\forall \mathtt{P}.\mathtt{P} \to \mathtt{P}$ and is proved by the program $\Lambda \mathtt{P}.\lambda \mathtt{x}{:}\mathtt{P}.\mathtt{x}$.

Task. Please prove the following propositions. For each of them, first translate them to a type in System F with pairs and sums. Then, provide a term that has that type under the empty environment (it can use `explode`). The last two propositions prove that $IL + LEM = IL + DNE$.

(1) $\forall P.P \to \neg\neg P$

Type: $\forall \mathtt{P}.\mathtt{P} \to ((\mathtt{P} \to \bot) \to \bot)$
Term: $\Lambda \mathtt{P}.\lambda \mathtt{x}{:}\mathtt{P}.\lambda \mathtt{f}{:}(\mathtt{P} \to \bot).\mathtt{f\ x}$

(2) $\forall P.\neg\neg(P \vee \neg P)$

Type: $\forall \mathtt{P}.((\mathtt{P} + (\mathtt{P} \to \bot)) \to \bot) \to \bot$

Term:
```
ΛP.λf:(P + (P → ⊥)) → ⊥.
  let a:P → ⊥ = λx:P.f (inl x) in
  let b:(P → ⊥) → ⊥ = λx:P → ⊥.f (inr x) in
  b a
```

(3) $(\forall P.P \vee \neg P) \to (\forall Q.\neg\neg Q \to Q)$

Type: $(\forall \mathtt{P}.\mathtt{P} + (\mathtt{P} \to \bot)) \to (\forall \mathtt{Q}.((\mathtt{Q} \to \bot) \to \bot) \to \mathtt{Q})$

Term: $\lambda \mathtt{x}{:}\forall \mathtt{P}.(\mathtt{P} + (\mathtt{P} \to \bot)).\Lambda \mathtt{Q}.\lambda \mathtt{f}{:}(\mathtt{Q} \to \bot) \to \bot.$

```
case (x [Q]) of
  inl q ⇒ q
  inr nq ⇒ explode [Q] (f nq)
```

(4) $(\forall P.\neg\neg P \to P) \to (\forall Q.Q \vee \neg Q)$

Type: $(\forall \mathtt{P}.((\mathtt{P} \to \bot) \to \bot) \to \mathtt{P}) \to (\forall \mathtt{Q}.\mathtt{Q} + (\mathtt{Q} \to \bot))$

Term: $\lambda \mathtt{x}{:}(\forall \mathtt{P}.((\mathtt{P} \to \bot) \to \bot) \to \mathtt{P}).\Lambda \mathtt{Q}.$

$\mathtt{x}\ [\mathtt{Q} + \mathtt{Q} \to \bot]\ (M\ [\mathtt{Q}])$

where M is the term in (2).

2 Featherweight Java with Field Assignment

The original Featherweight Java (FJ) proposal, as defined by Igarashi, Pierce and Wadler, presented a minimal calculus for Java's type system. In this question, we ask you to extend the calculus with a field assignment operation.

In your solution, we expect you to use a store (similar to the development you have seen for the STLC with references in TAPL, ch. 13, pp. 153–170). A field assignment should change the state of the corresponding object and return the object itself as a result. No new types or sequencing should be added. You need to ensure that progress and preservation still hold, but there is no need to prove it.

1. Adapt the syntax (terms and/or values) of Featherweight Java.
2. Define the new evaluations rule(s) and, if your calculus requires changes to the original (E-ProjNew), (E-InvkNew), (E-CastNew) or (E-Field) evaluation rules, give the updated versions.
3. Define the new typing rule(s) and, if your calculus requires changes in the original (T-Invk) or (T-New) typing rules, give the updated versions.

Think about what it means in terms of *values*. In FJ, values are `new` constructor calls with value arguments, because we can ignore the *identity* of objects. Is that still the case in your system? In other words, how do you deal with *aliasing*.

Solution:

Note that there are definitely several solutions. The design space is quite large. In particular, there are multiple choices of representation of the store, which then have consequences on everything else.

As the design space is quite large, we do not think that studying the following solution is very interesting. The *process* of discovering what is and is not possible is much more important than any particular solution.

1. Syntax

`t`	::=		**Terms**
	|		...
	|	`t.f = t`	assignment
	|	l	store location
`v`	::=		**Values**
	|	l	store location

2. Evaluation rules

A store μ is a map assigning to every location l a pair $\mu(l) = (\mathtt{C}, F)$ containing a class type $\mathtt{C}$ and partial map F from fields $\mathtt{f}$ of the class to values $F(\mathtt{f}) = \mathtt{v}$. The reduction judgment $\mathtt{t}_1 \mid \mu_1 \longrightarrow \mathtt{t}_2 \mid \mu_2$ relates a term $\mathtt{t}_1$ and its associated store μ_1 to an updated term $\mathtt{t}_2$ and associated store μ_2 resulting from a reduction step.

$$\frac{l \notin \mathrm{dom}(\mu) \quad \mathrm{fields}(\mathtt{C}) = \overline{\mathtt{C}}\ \overline{\mathtt{f}}}{\mathtt{new}\ \mathtt{C}(\overline{\mathtt{v}}) \mid \mu \longrightarrow l \mid (\mu, l \mapsto (\mathtt{C}, \overline{\mathtt{f}} \mapsto \overline{\mathtt{v}}))} \text{ (E-New)}$$

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

$$\frac{\mu(l) = (\mathtt{C}, F) \quad F(\mathtt{f}_i) = \mathtt{v}_i}{l.\mathtt{f}_i \mid \mu \longrightarrow \mathtt{v}_i \mid \mu} \text{ (E-Proj)}$$

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

$$\frac{\mu(l) = (\mathtt{C}, F) \quad \mathrm{fields}(\mathtt{C}) = \overline{\mathtt{C}}\ \overline{\mathtt{f}}}{l.\mathtt{f}_i = \mathtt{v} \mid \mu \longrightarrow l \mid [l \mapsto (\mathtt{C}, [\mathtt{f}_i \mapsto \mathtt{v}]F)]\mu} \text{ (E-Assign)}$$

$$\frac{\mathtt{t}_0 \mid \mu \longrightarrow \mathtt{t}_0' \mid \mu'}{\mathtt{t}_0.\mathtt{f}_i = \mathtt{t}_1 \mid \mu \longrightarrow \mathtt{t}_0'.\mathtt{f}_i = \mathtt{t}_1 \mid \mu'} \text{ (E-AssignC1)}$$

$$\frac{\mu(l) = (\mathtt{C}, F) \quad \mathtt{C} <: \mathtt{D}}{(\mathtt{D})l \mid \mu \longrightarrow l \mid \mu} \text{ (E-CastNew)}$$

$$\frac{\mathtt{t}_1 \mid \mu \longrightarrow \mathtt{t}_1' \mid \mu'}{\mathtt{v}.\mathtt{f}_i = \mathtt{t}_1 \mid \mu \longrightarrow \mathtt{v}.\mathtt{f} = \mathtt{t}_1' \mid \mu'} \text{ (E-AssignC2)}$$

3. Typing rules

The typing judgment $\Gamma \mid \Sigma \vdash \mathtt{t} : \mathtt{T}$ relates a term $\mathtt{t}$ to its type $\mathtt{T}$ in a typing context Γ and store typing Σ.

$$\frac{\Sigma(l) = \mathtt{T}}{\Gamma \mid \Sigma \vdash l : \mathtt{T}} \text{ (T-Store)}$$

$$\frac{\mathrm{fields}(\mathtt{C}) = \overline{\mathtt{D}}\ \overline{\mathtt{f}} \qquad \Gamma \mid \Sigma \vdash \overline{\mathtt{t}} : \overline{\mathtt{C}} \qquad \overline{\mathtt{C}} <: \overline{\mathtt{D}}}{\Gamma \mid \Sigma \vdash \mathtt{new}\ \mathtt{C}(\overline{\mathtt{t}}) : \mathtt{C}} \text{ (T-New)}$$

$$\frac{\Gamma \mid \Sigma \vdash \mathtt{t}_0 : \mathtt{T} \qquad \mathrm{fields}(\mathtt{C}) = \overline{\mathtt{D}}\ \overline{\mathtt{f}} \qquad \Gamma \mid \Sigma \vdash \mathtt{t}_1 : \mathtt{T}' \qquad \mathtt{T}' <: \mathtt{D}_i}{\Gamma \mid \Sigma \vdash \mathtt{t}_0.\mathtt{f}_i \ = \ \mathtt{t}_1 : \mathtt{T}} \text{ (T-Assign)}$$

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