Answers to the different parts are mostly independent of one another, provided properties stated in the preceding questions are admitted. Answers will also be judged by their clarity and concision.

Our goal is to study information flow in functional program that manipulate data having different security levels. For example, a program can manipulate both secret data (confidential clear text, secret cryptographic keys) and public data (encrypted text, public cryptographic keys). In this case, we would like to make sure that no secret input data flows to a public output, which would "leak" the secret.

To this end, we successively introduce and study: a language where data can be explicitly tagged with security labels (§2); a translation of this language to a more standard ML-like language, making explicit the tracking of security labels (§3); and the use of a type system to statically analyze information flows (§4).

1 Preliminaries / Preliminaires

We consider the call-by-value untyped λ-calculus parameterized over a collection of constants $c \in C$. Its syntax and semantics are defined as follows:

\[
\begin{align*}
a & ::= x | c | \lambda x. a | a a \\
v & ::= x | \lambda x. a | c v_1 \ldots v_n \\
E & ::= [] | a | v [ ]
\end{align*}
\]

\[
\begin{align*}
(\lambda x. a) v & \rightarrow [x \mapsto v]a \\
a_1 \rightarrow a_2 & \text{ if } a_1 \rightarrow_{\delta} a_2 \\
E[a_1] & \rightarrow E[a_2] & \text{ if } a_1 \rightarrow a_2 \quad (\text{Context})
\end{align*}
\]
Constants come with an arity and are split into constructors and destructors. Values $c_1 \ldots c_n$ are partial application of constants or full applications of constructors. The semantics is parameterized by a reduction relation $\rightarrow_\delta$. You may write $\rightarrow_\delta z$ the closure of the head reduction rule $z$ by rule (Context).

We introduce a let-construct $\text{let } x = a_1 \text{ in } a_2$ as syntactic sugar for $(\lambda x. a_2) a_1$.

**Question 1**

How should the definition of the language be modified if let-binding were treated as a primitive construct instead of syntactic sugar (no justification is needed)?

**Question 2**

We extend the language with integer constants. Are they constructors or destructors? What are their arities?

The language defined in this section is called the core language. In the following, we assume given a countable set of labels $\mathcal{L}$. In the examples, we use two labels: $S$ standing for “secret” and $P$ standing for “public”. The metavariable $\ell$ ranges over arbitrary labels.

**Question 3**

Consider the following three terms: Soient les trois termes suivants :

\[
\begin{align*}
  a_A &= (\lambda x. \lambda y. x) (S_\# 42) (P_\# 0) \\
  a_B &= (\lambda x. \lambda y. x) (P_\# 0) (S_\# 42) \\
  a_C &= (S_\# (\lambda x. \lambda y. x)) (P_\# 0) (P_\# 1)
\end{align*}
\]

How do they evaluate?

2  

2 A calculus with dependencies / Un calcul avec dépendances

The language defined in this section is called the source language. It extends the core language with a new construct to label expressions:

\[
\begin{align*}
  a &::= \ldots | \ell_\# a \\
  v &::= \ldots | \ell_\# v \\
  E &::= \ldots | \ell_\# [ ]
\end{align*}
\]

Notice that a node need not be labeled or may be labeled several times, as in $\ell_1\# (\ell_2\# a)$, which we simply write $\ell_1\# \ell_2\# a$.

**Question 3**

Consider the following three terms:

\[
\begin{align*}
  a_A &= (\lambda x. \lambda y. x) (S_\# 42) (P_\# 0) \\
  a_B &= (\lambda x. \lambda y. x) (P_\# 0) (S_\# 42) \\
  a_C &= (S_\# (\lambda x. \lambda y. x)) (P_\# 0) (P_\# 1)
\end{align*}
\]

How do they evaluate?
We introduce a new reduction rule to allow the extrusion of labels in the middle of redexes:

\[(\ell^\#_v_1)_2 \rightarrow (\ell^\#_v_1)_2\] (lift)

**Question 4**

Give the (longest) sequence of reductions starting with \(a_C\), indicating the head-reduction rules that are used.

**Question 5**

In \(a_A\) and \(a_B\), we replace the secret data 42 by another secret data 43. How does this affect the results of the evaluations of \(a_A\) and \(a_B\)? What can you say about the flows of secret data in these examples?

**Question 6**

We introduce boolean constants and a primitive construct if \(a_0\) then \(a_1\) else \(a_2\) with its usual lazy-evaluation reduction rules:

\[
\text{if true then } a_1 \text{ else } a_2 \rightarrow a_1 \quad \text{if false then } a_1 \text{ else } a_2 \rightarrow a_2
\]

What other rule should be added to permit evaluation in the presence of labels?

Can you explain this rule, technically and intuitively?

### 3 Labels as values / Les étiquettes comme valeurs

The source language defined in the previous section introduced labeling, which is not a standard construct in programming languages. In this section, we introduce a target language in which we will simulate the source language.

The target language is an extension of the core language with pairs and labels as values, but equipped with full reduction. That is, we allow reduction in any context, i.e. evaluation contexts become:

\[
E ::= [] | a | [a] | \lambda x. [a]
\]

Pairs are standard: we write \((a_1, a_2)\) for pairs, but we see them as an the application of an infix binary constructor \((\cdot, \cdot)\). We introduce two unary destructors \(\pi_1\) and \(\pi_2\) for the projections, with the two following reduction rules:

Les paires sont traitées de la façon usuelle : nous notons \((a_1, a_2)\) pour les paires, mais nous les voyons comme l’application d’un constructeur binaire infixé \((\cdot, \cdot)\). Nous introduisons deux destructeurs unaires \(\pi_1\) et \(\pi_2\) pour les projections, avec les règles de réduction suivantes :
We assume that labels form an upper semi-lattice with a smallest element \( \perp \). Thus two labels \( \ell_1 \) and \( \ell_2 \) always have a least upper bound, written \( \ell_1 \lor \ell_2 \). Moreover, \( \ell \lor \perp = \perp \lor \ell = \ell \). We add labels \( \ell \) to the target language as nullary constants and we introduce a binary construct \( \cdot @ \cdot \) to compute the join of two labels. That is, we add the following reduction rule:

\[
\ell_1 @ \ell_2 \rightarrow \ell_1 \lor \ell_2 \quad (\text{join})
\]

Notice that \( \cdot @ \cdot \) is a construct of the language, while \( \ell_1 \lor \ell_2 \) is just a mathematical notation and stands for the result of the join.

Besides, we add two extra reduction rules to perform joins of labels modulo associativity and neutral element:

\[
(a_1 @ a_2) @ a_3 \rightarrow a_1 @ (a_2 @ a_3) \quad (\text{assoc}) \quad \perp @ a \rightarrow a \quad (\text{neutral})
\]

We now define a translation of the source language into the target language. An expression \( a \) is translated into a pair \( \langle a_1, a_2 \rangle \) whose first component \( a_1 \) collects the labels attached to \( a \) and whose second component \( a_2 \) computes the (unlabeled) value of \( a \). We write \( \langle a \rangle \) for the translation of \( a \). We define \( \langle a \rangle = \langle \langle a \rangle^1, \langle a \rangle^2 \rangle \) where \( \langle a \rangle^1 \) and \( \langle a \rangle^2 \) are defined recursively as follows:

\[
\begin{align*}
\langle k \rangle^1 &= \perp \\
\langle x \rangle^1 &= \pi_1 x \\
\langle \lambda x. \ a \rangle^1 &= \perp \\
\langle a_1 a_2 \rangle^1 &= \langle a_1 \rangle^1 \ @ \pi_1 (\langle a_2 \rangle) \\
\langle \ell \# a \rangle^1 &= \ell \ @ \langle a \rangle^1
\end{align*}
\]

Notice that the translation approximates sets of labels by their least upper bound, i.e. using \( @ \) to join them instead of collecting all of them in a sequence. The smallest label \( \perp \) is used in the target expression when no label is carried by the source expression.

**Question 7**

How are let-bindings translated, when viewed as a derived form of the source language? Can you propose a nicer translation if we treat let-bindings as a primitive form in the source language?

\[
\begin{align*}
\langle \text{let } x = a_1 \text{ in } a_2 \rangle^1 &= ??? \\
\langle \text{let } x = a_1 \text{ in } a_2 \rangle^2 &= ???
\end{align*}
\]
Question 8

Now suppose that we add booleans and conditionals to both the source language and the target language. Propose translation rules for conditionals:

\[ (\text{if } a_1 \text{ then } a_2 \text{ else } a_3)^1 = ???? \quad (\text{if } a_1 \text{ then } a_2 \text{ else } a_3)^2 = ???? \]

Question 9

Show that every reduction in the source language is simulated by zero, one or several reductions of the translations in the target language:

\[ a_1 \rightarrow a_2 \implies (\langle a_1 \rangle) \rightarrow^* (\langle a_2 \rangle) \]

Give the proof for the two head reductions \( \beta \) and \( \text{lift} \); the result is trivial for reductions under a context. You can assume without proof the following property of substitutions (which holds thanks to full reduction):

\[ \langle a_1 \rangle[x \leftarrow \langle a_2 \rangle] \rightarrow^* \langle a_1[x \leftarrow a_2] \rangle \]

We write \( N \) for integers and \( M \) for (possibly) labeled integers of the form \( \ell_1^\sharp \cdots \ell_p^\sharp N \) where \( p \geq 0 \).

Question 10

Consider a source term \( a \) that evaluates to a labeled integer:

\[ a \rightarrow^* \ell_1^\sharp \cdots \ell_p^\sharp N \]

Show that the translation \( \langle a \rangle \) reduces to a value. Express this value in terms of \( N \) and of the labels \( \ell_1, \ldots, \ell_p \).

4 Typing the target calculus / Typage du calcul cible

In this section we assume that the target language is equipped with the type system of ML where expressions are implicitly typed.

We write \( \tau \) for types, \( \sigma \) for type schemes and \( \Gamma \) for typing environments. For the record, the typing rules are:

\[ \text{On note } N \text{ les entiers et } M \text{ les entiers (possiblement) étiquetés de la forme } \ell_1^\sharp \cdots \ell_p^\sharp N \text{ où } p \geq 0. \]
\[
\begin{array}{c|c|c|c|c}
\text{VAR} & \text{CONST} & \text{Abs} & \text{App} \\
\hline
x : \sigma \in \Gamma & c : \sigma & \Gamma, x : \tau_0 \vdash a : \tau & \Gamma \vdash a_2 : \tau_2, \Gamma \vdash a_1 : \tau_2 \to \tau_1 \\
\hline
\Gamma \vdash x : \sigma & \Gamma \vdash c : \sigma & \Gamma \vdash \lambda x. a : \tau_0 \to \tau & \Gamma \vdash a_1 a_2 : \tau_1 \\
\hline
\text{Let} & \text{Gen} & \text{Inst} \\
\hline
\Gamma \vdash a_1 : \sigma_1 & \Gamma, x : \sigma_1 \vdash a_1 : \sigma_2 & \alpha \vdash a : \sigma & \Gamma, \alpha \vdash a : \sigma \\
\hline
\Gamma \vdash \text{let } x = a_1 \text{ in } a_2 : \sigma_2 & \Gamma \vdash a : \forall \alpha. \sigma & \Gamma \vdash a : [\alpha \to \tau] \sigma \\
\end{array}
\]

The types of the constants are:

- \( N : \text{int} \)
- \( (\cdot, \cdot) : \forall \alpha_1. \forall \alpha_2. \alpha_1 \to \alpha_2 \to \alpha_1 \times \alpha_2 \)
- \( \pi_i : \forall \alpha_1. \forall \alpha_2. \alpha_1 \times \alpha_2 \to \alpha_i \)

We restrict the set of labels \( \mathcal{L} \) to the two labels \( S \) and \( P \), with \( P \) being smaller than \( S \) in the lattice of labels. Hence, the smallest label \( \bot \) is \( P \).

We introduce a new unary type constructor \( \# \) and two nullary type constructors \( \text{Pre} \) and \( \text{Abs} \). We assign the following type schemes to labels and join:

- \( S : \#\text{Pre} \)
- \( P : \forall \alpha. \#\alpha \)
- \( \odot : \forall \alpha. \#\alpha \to \#\alpha \to \#\alpha \)

**Question 11**

Can you explain why we give \( P \) a polymorphic type and not just \( P : \#\text{Abs} \)?

**Question 12 (Subject reduction for labels)**

Show that the head-reduction (join) and (assoc) preserves well-typedness.

We now assume that subject reduction holds in the target language.

**Question 13**

Show that if an expression \( a \) of the source language reduces to a labeled integer \( M \), and if \( \emptyset \vdash [\{a\}] : \text{Abs} \times \text{int} \), then \( \bot \) is the only possible label for \( M \).

**Question 14**

Building on the previous results, outline a static analysis for information flow. The analysis takes a closed term of the source language as input. It returns "secure" if it can guarantee that the term evaluates to a public integer, and "possibly insecure" otherwise.
Question 15

We now want to track integrity of data in addition to confidentiality. To this end, we use four labels instead of two: P meaning "public and untainted", S meaning "secret and untainted", T meaning "public and tainted", and ST = S ∨ T meaning "secret and tainted". How would you adapt the typing of labels accordingly?

5 Non-interference

In this section, we are going to prove that the tracking of labels performed by the source language correctly captures the flow of information through the program. More precisely, we show that if the value of a program does not carry a given label ℓ, then this value does not depend on the parts of the program labeled ℓ. This semantic property is called non-interference.

We extend the source language with a destructor constant Ω (pronounced “black hole”) of arity 1 that stands for an unknown expression. This constant has the following reduction rule:

Ω v → Ω (δΩ)

In other words, applying an unknown expression to a value produces an unknown expression, while an unknown expression is itself a value.

In this section we assume that L is composed of just the two labels P and S as in section 4.

We define the erasure [a] of a source term a where all subexpressions labeled S are replaced by Ω:

[x] = x
[λx.a] = λx.[a]
[a1 a2] = [a1] [a2]
[S♯a] = Ω
[P♯a] = P♯[a]

Note that [a] = a if and only if a contains no secret, i.e. no subexpressions labeled S. Also, [a1] = [a2] if and only if a1 and a2 differ only over secrets (i.e. subexpressions labeled S), but are otherwise identical.

Question 16

Show that reduction commutes with erasure, in the following sense:

a1 → a2 =⇒ [a1] →+S [a2]
Question 17

Consider two expressions $a_1$ and $a_2$ that differ only on labeled subexpressions and that evaluate to public values $v_1$ and $v_2$, respectively.

$$[a_1] = [a_2] \quad a_1 \rightarrow^* v_1 \quad a_2 \rightarrow^* v_2 \quad [v_1] = v_1 \quad [v_2] = v_2$$

Show that $a_1$ and $a_2$ evaluate to the same value, i.e. $v_1 = v_2$.

Question 18

Consider two expressions $a_1$ and $a_2$ that differ only on subexpressions labeled $S$, and that evaluate to labeled integers. Further assume the following typings for the translations of $a_1$ and $a_2$:

$$[a_1] = [a_2] \quad a_1 \rightarrow^* M_1 \quad a_2 \rightarrow^* M_2 \quad \Gamma \vdash ([a_1]) : \#Abs \times \text{int} \quad \Gamma \vdash ([a_2]) : \#Abs \times \text{int}$$

Conclude that the non-interference property holds: $a_1$ and $a_2$ evaluate to the same value.

6 Solutions

Question 1

$$a ::= \ldots | \text{let } x = a_1 \text{ in } a_1 \quad E ::= \ldots | \text{let } x = [ ] \text{ in } a \quad \text{let } x = v \text{ in } a \rightarrow_{\beta_{let}} [x \mapsto v]a$$

Question 2

Constructors of arity 0.

Question 3

$a_A$ reduces to the value $S\ (#\ 42)$. $a_B$ reduces to the value $P\ (#\ 0)$. $a_C$ is stuck because it cannot be reduced (rule $\beta$ does not apply), but it is not a value, since it is an application.

Question 4

$$a_C \rightarrow_{u_\theta} (S\ (#((\lambda x. \lambda y. x) (P\ (#0)))) (P\ (#1)))$$

$$\rightarrow_\beta (S\ (#(\lambda y. P\ (#0))) (P\ (#1)))$$

$$\rightarrow_{u_\theta} (\lambda y. P\ (#0)) (P\ (#1))$$

$$\rightarrow_\beta S\ (# P\ (#0)) (P\ (#1))$$

$$\rightarrow_\beta S\ (# P\ (#0))$$
Question 5

$a_A$ now reduces to the value $S^\sharp 43$, instead of $S^\sharp 42$ as before. However, $a_B$ still reduces to the value $P^\sharp 0$. The first case is a trivial example of a leak: the outcome of the program depends on the value of the secret. The reduction rules correctly track this leak, since the result is labeled $S$. In contrast, in the second case, the result is not labeled $S$ and is insensitive to the value of the secret: no leak takes place.

Question 6

if $\ell^\sharp v$ then $a_1$ else $a_2$ $\rightarrow$ $\ell^\sharp$ (if $v$ then $a_1$ else $a_2$)

Question 6 (continued)

Technically, the rule lifts labels outside of the conditional redex. Intuitively, even though the boolean expression $v$ does not appear in the result, the rule correctly says that the result of the evaluation depends on the boolean expression $v$ (tagged $\ell$) by forcing the tag $\ell$ to appear in the result.

For example, consider

if $a$ then false else true

This expression computes the negation of the boolean expression $a$. Its result carries as much information as $a$. (Indeed, by applying negation a second time, we can recover the value of $a$.) Therefore, if $a$ is a secret, the result of the conditional must also be a secret, even though its two arms false and true are public constants. So, any $S$ label carried by the value of $a$ must also be carried by the value of the conditional.

Question 7

$(\text{let } x = a_1 \text{ in } a_2)^1 = ((\lambda x. a_2) a_1)^1 = \bot \pi_1 ((\lambda x. (a_2))(a_1))$

$(\text{let } x = a_1 \text{ in } a_2)^2 = ((\lambda x. a_2) a_1)^2 = \pi_2 ((\lambda x. (a_2))(a_1))$

For a nicer translation, we can avoid the pointless join with $\bot$ and use a let-binding in the target language:

$(\text{let } x = a_1 \text{ in } a_2)^1 = (\text{let } x = (a_1) \text{ in } (a_2)^1)$

$(\text{let } x = a_1 \text{ in } a_2)^2 = (\text{let } x = (a_1) \text{ in } (a_2)^2)$

Question 8

$(\text{if } a_1 \text{ then } a_2 \text{ else } a_3)^1 = (a_1)^1 \oplus (\text{if } (a_1)^2 \text{ then } (a_2)^1 \text{ else } (a_3)^1)$

$(\text{if } a_1 \text{ then } a_2 \text{ else } a_3)^2 = \text{if } (a_1)^2 \text{ then } (a_2)^2 \text{ else } (a_3)^2$
Question 9

Consider first the head-reduction $\beta$. We reduce the two components of the translation separately:

$$\llbracket(\lambda x. a) v\rrbracket^1 = \bot \oslash_{\pi_1} (\llbracket(\lambda x. \{a\}) (\{v\})\rrbracket)$$

$$\rightarrow_{neutral} \pi_1 (\llbracket(\lambda x. \{a\}) (\{v\})\rrbracket)$$

$$\rightarrow_{\beta} \pi_1 (\{a\}[x \leftarrow \{v\}])$$

$$\rightarrow_{*} \pi_1 (\{a[x \leftarrow v]\})$$

$$\rightarrow_{\pi} \{a[x \leftarrow v]\}^1$$

$$\llbracket(\lambda x. a) v\rrbracket^2 = \pi_2 (\llbracket(\lambda x. \{a\}) (\{v\})\rrbracket)$$

$$\rightarrow_{\beta} \pi_2 (\{a\}[x \leftarrow \{v\}])$$

$$\rightarrow_{*} \pi_2 (\{a[x \leftarrow v]\})$$

$$\rightarrow_{\pi} \{a[x \leftarrow v]\}^2$$

(Notice that the last $\pi$ step in each case also uses full reduction.) The expected result follows:

$$\llbracket(\lambda x. a) v\rrbracket \rightarrow^* \{a[x \leftarrow v]\}$$

We now turn to the head-reduction $lift$, again considering the two components of the translation separately:

$$\llbracket(\ell_2^\sharp v_1) v_2\rrbracket^1 = \ell \oslash_{\pi_1} (\llbracket v_1 \rrbracket^1 \oslash_{\pi_1} (\llbracket v_2 \rrbracket))$$

$$\rightarrow_{assoc} \ell \oslash_{\pi_1} (\llbracket v_1 \rrbracket^1 \oslash_{\pi_1} (\llbracket v_2 \rrbracket))$$

$$\llbracket(\ell_2^\sharp v_1) v_2\rrbracket^2 = \pi_2 (\llbracket v_1 \rrbracket^2 \oslash v_2)$$

$$\llbracket(\ell_2^\sharp v_1) v_2\rrbracket = \{\ell_2^\sharp (v_1 v_2)\}^2$$

The expected result follows:

$$\llbracket(\ell_2^\sharp v_1) v_2\rrbracket \rightarrow^* \{\ell_2^\sharp (v_1 v_2)\}$$

Question 10

Using the previous question, we have a reduction sequence

$$\llbracket a \rrbracket \rightarrow^* \llbracket \ell_1^\sharp \cdots \ell_p^\sharp N \rrbracket = \ell_1 \oslash \cdots \oslash_{\pi} \ell_p \oslash \bot, N$$

The second component $N$ is already a value. The first component further reduces to $\ell_1 \vee \cdots \vee \ell_p$. Hence, $\llbracket a \rrbracket$ reduces to $\ell_1 \vee \cdots \vee \ell_p, N$, which is a value.

Question 11

This way $P$ can be given both type $\sharp Pre$ and $\sharp Abs$. The former allows to type $P \oslash S$. The latter allows to distinguish $P$ from $S$. In particular if $\Gamma \vdash \ell : \sharp Abs$, then $\ell$ can only be $P$.

Question 12

Case (join): Assume $\ell_1 \oslash \ell_2 \rightarrow \ell_1 \lor \ell_2$ and $\Gamma \vdash \ell_1 \oslash \ell_2 : \tau$ (1) We must show that $\Gamma \vdash \ell_1 \lor \ell_2 : \tau$ (2).

By inversion of typing for the judgment (1), $\tau$ must be of the form $\sharp a_0$ and moreover, $\Gamma \vdash \ell_1 : \sharp \tau$ (3) and $\Gamma \vdash \ell_2 : \sharp \tau$ (4). Notice that $\ell_1 \lor \ell_2$ is equal to either $\ell_1$ or $\ell_2$. Hence, the conclusion (2) follows from one of the hypotheses (3) or (4).

Case (assoc): Assume $\ell_1 \oslash (\ell_2 \oslash \ell_3) \rightarrow \ell_1 \oslash (\ell_2 \oslash \ell_3) : \tau$ and $\Gamma \vdash (\ell_1 \oslash (\ell_2 \oslash \ell_3) : \tau$. We must show that $\Gamma \vdash \ell_1 \oslash (\ell_2 \oslash \ell_3) : \tau$ (1). By inversion of typing for the judgment (2), $\tau$ must be of the form $\sharp a_i$ and moreover, $\Gamma \vdash \ell_i : \sharp \tau$ for $i \in \{1, 2, 3\}$. Hence, the conclusion follows by the type of the constant $\cdot \oslash \cdot$ and two application rules.
Question 13
Assume \( a \rightarrow^* M \) where \( M = \ell_1 \cdots \ell_p \). By Question 10, we know that \( \langle a \rangle \rightarrow^* \langle \ell, n \rangle \). By subject reduction, we have \( \emptyset \vdash \langle \ell, n \rangle : |\text{Abs} \times \text{int} \). By inversion of typing, we have \( \emptyset \vdash \ell : |\text{Abs} \) (1). By observation (classification of values) \( \ell \) can only be \( \bot \). Thus, the \( \ell_i \)'s must all be \( \bot \).

Question 14
Let \( a \) be the input source term.

1. Translate \( a \) to the target language, obtaining \( \langle a \rangle \).
2. Infer the most general type scheme \( \sigma \) of \( \langle a \rangle \) in the empty environment.
3. If \( |\text{Abs} \times \text{int} \) is an instance of \( \sigma \), then return "secure". Otherwise, return "possibly insecure".

Question 15
We turn \( \sharp \) into a binary type constructor and assign the following types:

\[
\begin{align*}
P : \forall \alpha_1. \forall \alpha_2. \sharp(\alpha_1, \alpha_2) & \quad S : \forall \alpha_2. \sharp(\text{Pre}, \alpha_2) & \quad T : \forall \alpha_1. \sharp(\alpha_1, \text{Pre}) & \quad ST : \sharp(\text{Pre}, \text{Pre}) \\
\odot : \forall \alpha_1. \forall \alpha_2. \sharp(\alpha_1, \alpha_2) & \rightarrow \sharp(\alpha_1, \alpha_2) & \rightarrow \sharp(\alpha_1, \alpha_2)
\end{align*}
\]

Question 16
We show the result by induction on the proof of \( a_1 \rightarrow a_2 \).

Case head reductions \( \beta \) and \( \text{lift} \): We do a case analysis on the label for \( \text{lift} \):

\[
\begin{align*}
\[(\lambda x. a) \ v] = (\lambda x. \ [a]) \ [v] & \rightarrow_\beta \ [a][x \leftarrow [v] = [a[x \leftarrow v]] \\
\[(S \sharp v_1) v_2] = \Omega \ [v_2] & \rightarrow_{\delta_\text{lift}} \Omega = [S_{\sharp} (v_1 v_2)] \\
\[(P_{\sharp} v_1) v_2] = (P_{\sharp} [v_1]) \ [v_2] & \rightarrow_{\text{lift}} P_{\sharp} ([v_1] [v_2]) = [P_{\sharp} (v_1 v_2)]
\end{align*}
\]

We used two trivial properties: 1- erasure commutes with substitutions, and 2- the erasure of a value is a value.

Case context: We have \( a_1 \rightarrow a_2 \) (1). We show that \( E[a_1] \rightarrow E[a_2] \) (2) by induction and case analysis on \( E \). If \( E \) is of the form \( S_{\sharp} [\ ] \), then \( E[a_1] = \Omega = E[a_2] \) and zero reductions take place after stripping. Otherwise, \( E[a_{1_i}] = E[[a_{1_i}]] \) for \( i \) in \( \{1, 2\} \). By induction hypothesis applied to (1), we have \( a_{1_i} \rightarrow^* a_{2_i} \). Then, the conclusion (2) follows by rule context.

Question 17
Using the result of the previous question, we have the following reduction sequences after erasure:

\[
\begin{align*}
a_1 \rightarrow^* [v_1] & \quad [a_2] \rightarrow^* [v_2]
\end{align*}
\]

Reductions in the source language are deterministic: for any \( a_1 \), there exists at most one \( a_2 \) such that \( a_1 \rightarrow a_2 \). Moreover, values do not reduce. Hence, for any \( a \) there exists at most one value \( v \) such that \( a \rightarrow^* v \). Since \( [a_{1_i}] = [a_{2_i}] \), it follows that \( [v_1] = [v_2] \). The result follows from the hypotheses \( [v_1] = v_1 \) and \( [v_2] = v_2 \).

Question 18
By Question 13, we know that the \( M_i \)'s may only contain \( P\)-labels. Thus, we have \( [v_i] = v_i \) for \( i \) in \( \{1, 2\} \). Then, Question 17 tells us that they are actually the same value.