Paramétricité et indépendance vis-à-vis des représentations
Parametricity and representation independence

Examen partiel du cours MPRI 2-4 / Mid-term exam for MPRI 2-4 course

2011/11/29 — Durée / duration: 2h30

Le but de ce problème est de montrer que si l'on peut donner un type "suffisamment polymorphe" à une fonction, alors on peut deviner à partir du type quel est le comportement de la fonction. Par exemple, si $\emptyset \vdash a : \alpha \to \alpha$, alors a est forcément la fonction identité. De tels résultats s'appellent des propriétés de paramétricités, ou encore des "théorèmes gratuits" comme les appelle Ph. Wadler (1989). En présence d'abstraction de types (types $\exists\alpha.\tau$), le dual des propriétés de paramétricité est l'indépendance vis-à-vis des représentations : des conditions suffisantes pour que deux implémentations différentes du même type abstrait $\exists\alpha.\tau$ soient indistinguables par le reste du programme.

The goal of this exam is to show that if we can give a "polymorphic enough" type to a function, then we can guess the behavior of the function from its type. For instance, if $\emptyset \vdash a : \alpha \to \alpha$, then a must be the identity function. Such results are called parametricity properties or "theorems for free" as in Ph. Wadler (1989). In the presence of type abstraction (types $\exists\alpha.\tau$), the dual of parametricity properties is representation independence: sufficient conditions for two different implementations of the same abstract type $\exists\alpha.\tau$ are indistinguishable by the rest of the program.

On se place dans le λ-calcul simplement et implicitement typé, étendu avec des constantes entières de type abec une sémantique en appel par valeur. On rappelle la syntaxe des expressions, des valeurs et des termes :

We considered the implicitly-typed simply-typed λ-calcul extended with integer constants equipped with a call-by-value semantics. We reminds the syntax of types, terms, and values and the typing rules.

$$\tau ::= \alpha \mid \mathit{int} \mid \tau \to \tau \qquad a ::= x \mid N \mid a\ a \mid \lambda x.a \qquad v ::= N \mid \lambda x.a$$

$$\textsc{Int}\ \ \Gamma \vdash N : \mathit{int} \qquad \textsc{Var}\ \ \frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau} \qquad \textsc{Lam}\ \ \frac{\Gamma, x : \tau_0 \vdash a : \tau}{\Gamma \vdash \lambda x.a : \tau_0 \to \tau} \qquad \textsc{App}\ \ \frac{\Gamma \vdash a_1 : \tau_2 \to \tau_1 \qquad \Gamma \vdash a_2 : \tau_2}{\Gamma \vdash a_1\ a_2 : \tau_1}$$

Préambule / Warm-up

Dans ce préambule, on suppose que le calcul simplement typé est étendu avec une construction $\mu f.\lambda x.a$ pour les fonctions récursives.

In this warm-up section, we assume that the language is extended with a construct $\mu f.\lambda x.a$ for recursive functions.

Question 1

Soit a un terme tel que $\emptyset \vdash a : \alpha$. Montrer que, nécessairement, a diverge, c'est-à-dire que l'évaluation de a conduit forcément à une suite infinie de réductions $a \to a_1 \to a_2 \to \cdots$. Peut-on définir un tel terme a sans utiliser les fonctions récursives ?

Let a be a term such that $\emptyset \vdash a : \alpha$. Show that, necessarily, a diverges, that is, its evaluation must lead to an infinite sequence of reductions $a \to a_1 \to a_2 \to \cdots$. Can such a term a be defined without using recursive functions?

Answer: By way of contradiction, asume that the reduction of a terminates. By suject reduction and progress, it must end with a value v of type α. However, this is not possible, since values are either integers of type $\int$ or functions of arrow types $\tau \to \tau'$. Therefore a must diverge. In the absence of recursive functions, there is no such a because all reductions terminates. □

Question 2

Soit a un terme tel que $\emptyset \vdash a : \texttt{int} \to \alpha$. Montrer que a est une fonction qui ne termine jamais : $a\ N$ diverge pour tout entier N.

Let a be a term such that $\emptyset \vdash a : \texttt{int} \to \alpha$. Show that a is a nonterminating function: $a\ b$ diverges for any argument b.

Answer: Since, for any integer N, we have $\emptyset \vdash a\ N : \alpha$, the application $a\ N$ must diverge, as shown in the question 1.

□

Question 3

Donner un exemple d'un terme a tel que $\emptyset \vdash a : \alpha \to \alpha$ (on donnera seulement le terme, pas sa dérivation de typage). Donner un autre terme de ce type qui se comporte de la même façon. Voyez-vous d'autres termes du même type qui se comportent différemment, qui n'utilise pas les fonctions récursives ? qui utilise les fonctions récursives ?

Give an example of a term a such that $\emptyset \vdash a : \alpha \to \alpha$ (just give the term, without its typing derivation). Give another term of that type with the same behavior. Can you see another term of that type with a different behavior that does not use recursive functions? that uses recursive functions?

Answer: The identity function $a_0 = \lambda x.x$ is the obvious example. Another term with the same behavior is $\lambda x.a_0\ x$. As we shall see below, without recursive definitions, there is no other term of that type with the same behavior. If we allow recursive definitions, the are two other possible behaviors: a term that always loops, and a value that loops whenever applied. An example of the second kind is $a_2 = (\mu f.\lambda x.fx)$; an example of the first kind is $a_2\ a_0$.

□

Les relations logiques / Logical relations

Pour raisonner sur le comportement des termes en fonction de leurs types, nous allons utiliser une puissante construction sémantique appelée *relations logiques*.

To reason on the behaviors of terms as a function of their types, we will use a powerful semantic construction called *logical relations*.

On note *Val* l'ensemble des valeurs closes. Une interprétation des variables de types est une fonction finie ρ qui associe à certaines variables de types α un ensemble quelconque (possiblement vide ou infini) de paires de valeurs closes :

We write *Val* for the set of closed values. A type variable interpretation is a finite function ρ that maps some type variables α to (possibly infinite) sets of pairs of values:

$$\rho(\alpha) = \{(v_1, v_1'); (v_2, v_2'); \ldots\} \subseteq \mathit{Val} \times \mathit{Val}$$

On définit les deux relations suivantes :

We define the following two relations:

$\rho \vdash a \approx a' : \tau$: les termes a et a' sont reliés au type τ dans l'interprétation ρ
terms a and a' are related at type τ in interpretation ρ

$\rho \vdash v \sim v' : \tau$: les valeurs v et v' sont reliées au type τ dans l'interprétation ρ
values v and v' are related at type τ in interpretation ρ

Ces relations sont définies par les équivalences logiques suivantes :

These relations are defined by the following logical equivalences:

$$\begin{aligned}
\rho \vdash a \approx a' : \tau &\iff \exists v \in \mathit{Val}, \exists v' \in \mathit{Val},\ a \xrightarrow{*} v \wedge a' \xrightarrow{*} v' \wedge \rho \vdash v \sim v' : \tau \\
\rho \vdash v \sim v' : \texttt{int} &\iff v = v' = N \text{ for some integer constant } N \\
\rho \vdash v \sim v' : \alpha &\iff \alpha \in \mathrm{Dom}(\rho) \wedge (v, v') \in \rho(\alpha) \\
\rho \vdash v \sim v' : \tau_1 \to \tau_2 &\iff \forall w \in \mathit{Val}, \forall w' \in \mathit{Val},\ \rho \vdash w \sim w' : \tau_1 \Longrightarrow \rho \vdash v\ w \approx v'\ w' : \tau_2
\end{aligned}$$

Autrement dit, deux termes sont reliés si leurs évaluations terminent sur deux valeurs qui sont reliées. Deux valeurs de type `int` sont reliées si ce sont des constantes entières et si elles sont égales. Deux valeurs de type α sont reliées si elles figurent dans l'interprétation $\rho(\alpha)$ de la variable de type. Enfin, deux valeurs d'un type fonctionnel $\tau_1 \to \tau_2$ sont reliées si ce sont des fonctions qui envoient des arguments reliés au type τ_1 sur des résultats reliés au type τ_2.

In other words, two terms are related if their evaluations terminate on two values that are related. Two values of type `int` are related if they are integer constants and they are equal. Two values of type α are related if they belong to the interpretation $\rho(\alpha)$ of the type variable α. Finally, two values of a function type $\tau_1 \to \tau_2$ are related if they are functions that map related arguments (at type τ_1) to related results (at type τ_2).

Question 4

Expliquer (sans démonstration formelle) pourquoi les relations $\rho \vdash a \approx a' : \tau$ et $\rho \vdash v \sim v'$ sont mathématiquement bien définies par les équivalences logiques ci-dessus.

Explain (without a formal proof) why the two relations $\rho \vdash a \approx a' : \tau$ and $\rho \vdash v \approx v'$ are mathematically well-defined by the logical equivalences above.

Answer: The recursive defined is well-founded; the definition is in fact by induction on the size of types ($\rho \vdash (\cdot \approx \cdot) : \tau$ depends on $\rho \vdash (\cdot \approx \cdot) : \tau$ which itself depends on $\rho \vdash (\cdot \approx \cdot) : \tau'$ for τ' stictly smaller than τ).

□

Question 5

Dans cette question, on se donne les opérateurs arithmétiques usuels sur le type `int`. Est-ce que

In this question, we assume given the usual arithmetic operators over type `int`. Is it the case that

$$\emptyset \vdash (\lambda x.\ x + x) \approx (\lambda x.\ x \times 2) : \texttt{int} \to \texttt{int} \quad ?$$

(Répondre informellement ; il n'est pas nécessaire de faire une démonstration.) Même question pour

(Informal answer, please; no need for a formal proof.) Same question for

$$\emptyset \vdash (\lambda x.\ x) \approx (\lambda x.\ x + 1) : \texttt{int} \to \texttt{int} \quad ?$$

Answer: The first two functions are indistinguisable: they return the same integer $2n$ when applied to the same integer n. The last two functions differ, for instance when applied to integer 0, one returns 0 while the other one returns 1.

□

Question 6

Soient a, a', b, b' des termes. Démontrer que

Let a, a', b, b' be terms. Prove that

$$\rho \vdash a \approx a' : \tau \ \wedge \ a \xrightarrow{*} b \ \wedge \ a' \xrightarrow{*} b' \implies \rho \vdash b \approx b' : \tau$$

(≈ est close par réduction) *(≈ is closed by reduction)*

Answer: We first show tha ≈ is closed by one-step reduction on the left-hand side. Asume $\rho \vdash a \approx a' : \tau$ and $a \to b$ By definition of ≈ there exist values v and v' such that $a \xrightarrow{*} v$ and $a' \xrightarrow{*} v'$ (**1**) and $\rho \vdash v \sim v' : \tau$ (**2**). Since reduction is deterministic, we must actually have a sequence of the form $a \to b \xrightarrow{*} v$. In particular, $b \xrightarrow{*} v$, which together with (1) and (2) impiles $\rho \vdash b \approx a'$. The closure of ≈ by one-step reduction on the right-hand side follows by summetry of the problem. Its closure by arbitrary reduction on either side follows by an iteration of these elementary property. □

Question 7

Soient a, a', b, b' des termes. Démontrer que *Let a, a', b, b' be terms. Prove that*

$$\rho \vdash a \approx a' : \tau_2 \to \tau_1 \ \wedge \ \rho \vdash b \approx b' : \tau_2 \implies \rho \vdash a \ b \approx a' \ b' : \tau_1$$

(≈ est close par application) *(≈ is closed by application)*

Answer: Assume $\rho \vdash a \approx a' : \tau_2 \to \tau_1$ and $\rho \vdash b \approx b' : \tau_2$. By definition of ≈ there exist values v, v', w, and w' such that:

$$a \to v \ \wedge \ a' \to v' \ \wedge \ b \to w \ \wedge \ b' \to w' \ (\mathbf{1}) \qquad \rho \vdash v \sim v' : \tau_2 \to \tau_1 \ (\mathbf{2}) \qquad \rho \vdash w \sim w' : \tau_2 \ (\mathbf{3})$$

The definition of ∼ for function types applied to (2) and (3) implies $\rho \vdash v \ w \approx v' \ w' : \tau_1$ (**4**). This implies $\rho \vdash a \ b \approx a' \ b' : \tau_1$, as expected, since ∼ is closed by reduction (question 6) and the four reductions (1) implies $a \ b \to v \ w$ and $a' \ b' \to v' \ w'$. □

Question 8

Nous allons montrer le lemme fondamental des relations logiques*, dû à R. Statman (1985). Supposons*

We now show the fundamental lemma of logical relations*, due to R. Statman (1985). Assume*

$$\Gamma \vdash a : \tau \quad \textit{avec/with } \mathrm{Dom}(\Gamma) = \{x_1, \ldots, x_n\} \qquad (\boldsymbol{A})$$

Soit θ and θ' des substitutions *Let θ and θ' be substitutions*

$$\theta = [x_1 \mapsto v_1, \ldots, x_n \mapsto v_n] \qquad \theta' = [x_1 \mapsto v'_1, \ldots, x_n \mapsto v'_n]$$

telles que *such that*

$$\forall i \in= \{1, \ldots, n\}, \quad \rho \vdash \theta x_i \sim \theta' x_i : \Gamma(x_i) \qquad (\boldsymbol{B})$$

Démontrer que *Prove that*

$$\rho \vdash \theta a \approx \theta' a : \tau \qquad (\boldsymbol{C})$$

Autrement dit, pour tout terme bien typé a, si l'on remplace ses variables libres par deux jeux de valeurs reliées, on obtient deux termes reliés. Indication : on procédera par récurrence sur la dérivation de $\Gamma \vdash a : \tau$ et par cas sur a.

In other words, for every well-typed term a, if we replace its free variables by two sets of related values, we obtain two related terms. Hint: proceed by induction over the typing derivation of $\Gamma \vdash a : \tau$ and by case analysis over a.

Answer: We assume (1) and (2) and show (3) by induction on a.

Case a is x. By inversion of typing, x must be some x_i in $dom(\Gamma)$ and τ must be τ_i. Hence, the conclusion (3) is $\rho \vdash \theta x_i \approx \theta' x_i : \tau_i$, which immediately follows from (2).

Case a is N. By inversion of typing, τ must be int. Hence, the conclusion (3) is $\rho \vdash N \approx N : int$ which holds by definition of $\sim$.

Case a is $\lambda x.a_1$. We may assume, *w.l.o.g.*, that x does not appear free in Γ (**4**). By inversion of typing, τ is of the form $\tau_0 \to \tau_1$ with $\Gamma, x : \tau_0 \vdash a_1 : \tau_1$ (**5**). The conclusion (3) is $\rho \vdash \theta(\lambda x.a_1) \approx \theta'(\lambda x.a_1) : \tau_0 \to \tau_1$, *i.e.* $\rho \vdash \lambda x.\theta a_1 \approx \lambda x.\theta' a_1 : \tau_0 \to \tau_1$, thanks to (4). Since both terms are values, it suffices to show $\rho \vdash \lambda x.\theta a_1 \sim \lambda x.\theta' a_1 : \tau_0 \to \tau_1$.

Let v and v' such that $\rho \vdash v \sim v' : \tau_0$ (**6**), we show that $\rho \vdash (\lambda x.\theta a_1)\ v \approx (\lambda x.\theta' a_1)\ v' : \tau_1$ (**7**). Let θ_1 be $[x \mapsto v]\theta$ and θ'_1 be $[x \mapsto \theta]$. By induction hypothesis applied to (5), (6), (2), and (1), we have $\rho \vdash \theta_1 a_1 \approx \theta_1 a_1 : \tau_1$, *i.e.* $\rho \vdash [x \mapsto v](\theta a_1) \sim [x \mapsto v'](\theta' a_1) : \tau_1$. This implies (7), since $\approx$ is closed by reduction (question 6).

Case a is $a_1\ a_2$. By inversion of typing applied to (1), τ must be of the form $\tau_2 \to \tau_1$ and such that $\Gamma \vdash a_1 : \tau_2 \to \tau_1$ and $\Gamma \vdash a_2 : \tau_2$. Applying the induction hypothesis twice to a_1 and a_2, we obtain $\rho \vdash \theta a_1 \approx \theta' a_1 : \tau_2 \to \tau_1$ and $\rho \vdash \theta a_2 \approx \theta' a_2 : \tau_2$. This implies $\rho \vdash (\theta a_1)\ (\theta a_2) \approx (\theta' a_1)\ (\theta' a_2) : \tau$, since $\approx$ is closed by application (question 7), which is the conclusion (3). □

Question 9

Comme corollaire du lemme fondamental, donner une nouvelle preuve de normalisation forte pour le λ-calcul simplement typé :

As a corollary of the fundamental lemma, give a new proof of strong normalization for simply-typed λ-calculus:

$$\emptyset \vdash a : \tau \implies \exists v \in \mathit{Val},\ a \xrightarrow{*} v$$

Answer: Assume $\emptyset \vdash a : \tau$. The fundamental lemma implies $\emptyset \vdash a \approx a : \tau$, which by definition of $\approx$ implies in particular that there exists a value v such that $a \xrightarrow{*} v$.

□

Théorèmes gratuits ! / Theorems for free!

Question 10

Soit f un terme tel que

Let f be a term such that

$$\emptyset \vdash f : \alpha \to \alpha$$

En utilisant le lemme fondamental, montrer que f se comporte comme la fonction identité $\lambda x.x$:

Using the fundamental lemma, show that f behaves like the identity function $\lambda x.x$:

$$\forall v \in \mathit{Val},\ f\ v \xrightarrow{*} v$$

Indication : étant donné une valeur v, interpréter α par l'ensemble $\rho(\alpha) = \{(v, v)\}$.

Hint: for a given value v, interpret α by the set $\rho(\alpha) = \{(v, v)\}$.

□

Question 11

Soit f un terme tel que

Let f be a term such that

$$\emptyset \vdash f : \alpha \to \beta \to \alpha$$

En s'inspirant de la question précédente, montrer que f se comporte comme le combinateur $K = \lambda x.\lambda y.x$.

Reasoning as in the previous question, show that f behaves like the combinator $K = \lambda x.\lambda y.x$:

$$(\forall v_1, v_2 \in \mathit{Val},\ f\ v_1\ v_2 \xrightarrow{*} v_1)$$

□

Question 12

Soit f un terme tel que

Let f be a term such that

$$\emptyset \vdash f : \alpha \to \alpha \to \alpha$$

En remarquant que ce type est une instance du type de la question précédente, donner un terme possible pour f. En raisonnant par symmétrie, donner une autre terme possible pour f.

Noticing that this is an instance of the type of the previous question, give one possible term for f. Reasoning by symmetry, give another possible term for f.

Le terme f n'a en fait que deux comportement possibles. Décrire ceux-ci. Puis monter que ce sont bien les deux seuls comportements possibles.

The term f has actually only two possible behaviors. Describe them. Then show that these are indeed the only two possible behaviors for f.

Indication : étant donné deux valeurs v_1, v_2, interpréter α par l'ensemble $\rho(\alpha) = \{(1, v_1); (2, v_2)\}$, montrer que $\rho \vdash f\ 1\ 2 \approx f\ v_1\ v_2 : \alpha$, et conclure.

Hint: for two given values v_1, v_2, interpret α by the set $\rho(\alpha) = \{(1, v_1); (2, v_2)\}$, show that $\rho \vdash f\ 1\ 2 \approx f\ v_1\ v_2 : \alpha$, and conclude.

Answer: The two behaviors can be described by:

$$(\forall v_1, v_2 \in \mathit{Val},\ f\ v_1\ v_2 \xrightarrow{*} v_1) \quad \vee \quad (\forall v_1, v_2 \in \mathit{Val},\ f\ v_1\ v_2 \xrightarrow{*} v_2)$$

□

Question 13

Soit f un terme tel que

Let f be a term such that

$$\emptyset \vdash f : (\alpha \to \alpha) \to (\alpha \to \alpha)$$

Montrer que le seul comportement possible pour f est de composer son argument avec lui-même n fois, pour un n indépendant de son argument :

Show that the only possible behavior for f is to compose its functional argument with itself n times, for an n that is independent of the argument:

$$f \equiv \lambda g.\lambda x.\ x \ \vee\ f \equiv \lambda g.\lambda x.\ g\ x \ \vee\ f \equiv \lambda g.\lambda x.\ g(g\ x) \ \vee\ \ldots \ \vee\ f \equiv \lambda g.\lambda x.\ g^n(x) \ \vee\ \ldots$$

Plus formellement, pour une valeur $v : \tau$ et une fonction $g : \tau \to \tau$ données, on définit la suite $(v_i)_{i \in N}$ des valeurs des itérés de g par :

More formally: given a value $v : \tau$ and a function $g : \tau \to \tau$, define the sequence $(v_i)_{i \in N}$ of the value of the iterates of g by:

$$v_0 = v \qquad \forall i,\quad g\ v_i \xrightarrow{*} v_{i+1}$$

Construire une interprétation ρ telle que

Construct an interpretation ρ such that

$$\rho \vdash f\ (\lambda x.\ x + 1)\ 0 \approx f\ g\ v : \alpha$$

En déduire qu'il existe un entier n, indépendant de g et v, tel que

Deduce that there exists an integer n, independent of g and v, such that

$$f\ g\ v \xrightarrow{*} v_n$$

□

Question 14

Nous étendons le langage avec un destructeur unaire succ of type $int \to int$ and δ-reduction :

We now extend the language with the unary descructor succ of type $int \to int$ and δ-reduction:

$$succ\ N \to N + 1$$

Montrer que le lemme fondamental des relations logiques est préservé. (Décrire et détailler uniquement le nouveau cas à ajouter dans la preuve précédente.)

Show that the fundamental lemma of logical relation is preserved. (Show and detail only the case that must be added to the previous proof.)

Answer: A new case of expression appears in the case for constants:

Case a is succ. By inversion of typing, the type τ must be $int \to \int$. The conclusion (3) becomes $\rho \vdash succ \approx succ : int \to int$, *i.e.* $\rho \vdash succ \sim succ : int \to int$. For any value v and v' such that $\rho \vdash v \sim v' : int$, we must show that $\rho \vdash succ\ v \approx succ\ v' : int$ (**4**). By definition of $\sim$, v and v' must be a same integer N. Thus $succ\ v$ and $succv'$ reduces to the same integer $N + 1$, which are always related. Hence, (4) holds.

□

Indépendance vis-à-vis des représentations / Representation Independence

Nous étendons maintenant le langage avec des types existentiels primitifs. Afin de changer le minimum à la présentation précédente, Nous gardons le style implicitement typé et nous choisissons une présentation Γ n'introduit pas les variables de types. Le langage est étendu comme suit :

We now extend the language with primitive existential types. In order to change as litle as possible to the previous setting, we keep the implicitly typed style and we choose a presentation where type variables are not introduced in Γ. The language is extended as follows:

$$\tau ::= \ldots \mid \exists\alpha.\tau \qquad a ::= \ldots \mid \mathsf{pack}\ a \mid \mathsf{let}\ x = \mathsf{unpack}\ a\ \mathsf{in}\ a \qquad v ::= \ldots \mid \mathsf{pack}\ v$$

$$\frac{\Gamma \vdash a : [\alpha \mapsto \tau_0]\tau}{\Gamma \vdash \mathsf{pack}\ a : \exists\alpha.\tau} \qquad \frac{\Gamma \vdash a_0 : \exists\alpha.\tau_0 \qquad \Gamma, x : \tau_0 \vdash a : \tau \qquad \alpha \notin ftv(\Gamma, \tau)}{\Gamma \vdash \mathsf{let}\ x = \mathsf{unpack}\ a_0\ \mathsf{in}\ a : \tau}$$

$$\mathsf{let}\ x = \mathsf{unpack}\ (\mathsf{pack}\ v)\ \mathsf{in}\ a \to [x \mapsto v]a$$

Les contexte d'évaluation sont étendus de façon habituelle.

Evaluation contexts are extended as usual.

Nous étendons la définition des relations logiques pour les valeurs de type existentiel comme suit :

We now extend the logical relation for the case of existential types as follows:

$$\rho \vdash v \sim v' : \exists\alpha.\tau \iff \exists w, w' \in Val,\ \exists R \subseteq Val \times Val,\ \wedge \begin{cases} v = \mathsf{pack}\ w \wedge v' = \mathsf{pack}\ w' \\ \rho, (\alpha \mapsto R) \vdash w \sim w' : \tau \end{cases}$$

Autrement dit, deux valeurs empaquetées sont reliées au type $\exists\alpha.\tau$ si et seulement si leurs valeurs déparquetées sont reliées au type τ dans une extension de l'interprétation en α pour une relation R quelconque.

On note

In other words, two packed values are related at type $\exists\alpha.\tau$ if and only the unpacked values are related at type τ in an extension of the interpretation on α with some relation R.

We write

$$\mathcal{R}_\rho(\tau) = \{(v, v') \mid \rho \vdash v \sim v' : \tau\}$$

On admettra le *lemme de substitution* suivant :

We shall admit the following *substitution lemma*:

$$\rho, \alpha \mapsto \mathcal{R}_\rho(\tau_0) \vdash a \approx a' : \tau \quad \Longleftrightarrow \quad \rho \vdash a \approx a' : [\alpha \mapsto \tau_0]\tau$$

Question 15

Montrer que le lemme fondamental est encore valide avec les types existentiels : détailler dans la preuve de la question 8 les nouveaux cas correspondants aux nouvelles constructions du langage.

The statement of the fundamental lemma is unchanged. Show that the proof of the lemma can be extended: detail in the proof of question 8 the new cases case for to the new language constructs).

Answer: We reuse the notation of question 8. The hypotheses are (1) and (2). The conclusion (3) is shown by structural induction on the term a. We just need to two new cases:

Case a is $\mathsf{pack}\ a'$. By inversion of typing applied to (1), τ is of the form $\exists\alpha.\tau'$ and $\Gamma \vdash a' : [\alpha \mapsto \tau_0]\tau'$ for some type τ_0'. The induction hypothesis applied to a' implies that $\rho \vdash \theta a' \approx \theta' a' : [\alpha \mapsto \tau_0]\tau'$. the *substitution lemma* implies that $\rho, \alpha \mapsto \mathcal{R}_\rho(\tau_0) \vdash \theta a' \approx \theta' a' : \tau'$. Therefore, there exist values v and v' such that $\theta a' \xrightarrow{*} v$ and $\theta' a' \to v'$ (**1**) and $\rho, \alpha \mapsto \mathcal{R}_\rho(\tau_0) \vdash v \sim v' : \tau'$. This last relation implies $\rho \vdash \mathsf{pack}\ v \sim \mathsf{pack}\ v' : \exists\alpha.\tau'$ (**2**) by definition of $\sim$ for existential types. The two reduction sequences (1) imply $\mathsf{pack}\ (\theta a') \xrightarrow{*} \mathsf{pack}\ v$ and $\mathsf{pack}\ (\theta' a') \xrightarrow{*} \mathsf{pack}\ v'$, *i.e.* $\theta(\mathsf{pack}\ a) \xrightarrow{*} \mathsf{pack}\ v$ and $\theta'(\mathsf{pack}\ a) \xrightarrow{*} \mathsf{pack}\ v'$, since substitution commutes with $\mathsf{pack}\ \cdot$. Combined with (2), this implies the conclusion (3).

Case a is $\mathsf{let}\ x = \mathsf{unpack}\ a_1\ \mathsf{in}\ a_2$. We may assume *w.l.o.g.* that $x \notin dom(\Gamma)$. By inversion of typing applied to (1), we have $\Gamma \vdash a_1 : \exists\alpha.\tau_1$ and $\Gamma, x; \tau_1 \vdash a_2 : \tau$ where $\alpha \notin ftv(\Gamma, \tau)$. By induction hypothesis applied to a_1 we have $\rho \vdash \theta a_1 \approx \theta' a_1 : \exists\alpha.\tau_1$ By definition of $\approx$, there exist v_1 and v_1' such that $\theta a_1 \xrightarrow{*} \mathsf{pack}\ v_1$ and $\theta' a_1 \xrightarrow{*} \mathsf{pack}\ v_1'$; and a relation R such that $\rho, \alpha \mapsto R \vdash v_1 \approx v_1 : \tau_1$.

Let θ_1 be $[\alpha \mapsto v_1]\theta$ and θ_1' be $[\alpha \mapsto v_1']\theta'$. By preservation of typing, we have $\Gamma \vdash v_1 : \tau_1$ and $\Gamma \vdash v_1' : \tau_1$. Therefore we can apply the induction hypothesis to a_2 with θ_1 and θ_1'. Thus gives $\rho \vdash \theta_1 a_2 \approx \theta_1' a_2 : \tau$, *i.e.*, $\rho \vdash [x \mapsto v_1](\theta a_2) \approx [x \mapsto v_1'](\theta' a_2) : \tau$ (**3**). Observe that θa is $\mathsf{let}\ x = \mathsf{unpack}\ \theta a_1\ \mathsf{in}\ \theta a_2$ and reduces to $[x \mapsto v_1](\theta a_2)$; and similarly when replacing θ by θ'. This and (3) proves the conclusion (3), since $\approx$ is is closed by reduction (question 6). □

Question 16

Dans cet exercise, on suppsoe que le langage est étendu avec des booléens (avec les constructeurs true et false et le destructeur not) et une fonction even qui à un entier n retourne un booléen indiquant la parité de n comme un booléen.

Le language est aussi étendu avec des triplets. Les relations logiques sur les triplets sont définies par :

In this exercise, we asume that the language is extended with booleans (constructors true and false and the destructor not) and with a destructor even that given an integer n returns the parity of n as a boolean.

The language is also extended with triples. Logical relations on triplets are defined by:

$$\begin{aligned}\rho \vdash v \sim v' : \tau_1 \times \tau_2 \times \tau_3 \iff\ & \exists v_1, v_2, v_3, v_1', v_2', v_3'. \\ & v = (v_1, v_2, v_3) \wedge v' = (v_1', v_2', v_3) \wedge\ \forall i \in \{1,2,3\},\ \rho \vdash v_i \sim v_i' : \tau_i\end{aligned}$$

On admet que les propriétés sont préservées par ces extensions.

Soit les définitions suivantes :

We admit that the previous properties are preserved by these extensions.

Condider the following definitions:

$$v_1 = (0, succ, even) \qquad v_2 = (true, not, \lambda x.x) \qquad \tau_0 = \exists\alpha.(\alpha \times (\alpha \to \alpha) \times (\alpha \to bool))$$

On voudrait s'arrurer que les deux paquets **pack** v_1 *et* **pack** v_2 *sont indistinguables au type* τ_0*. Formellement, on veut montrer* $\emptyset \vdash v_1 \sim v_2 : \tau_0$*.*

Montrer qu'il suffit de constuire une interprétation ρ *qui relie les deux implémentations à un certain type. Construire cette interprétation. Vérfier qu'elle relie bien les deux implémentations.*

We wish to show that the two packages **pack** v_1 *and* **pack** v_2 *are indistinguable at type* τ_0*. Formally, we must show* $\emptyset \vdash v_1 \sim v_2 : \tau_0$*.*

Show that it suffices to build an appropriate interpretation ρ *that relates both implementations at a certain type. Build it. Verify that it correctly relates the two implementations.*

□

Question 17 (Universal types)

Nous étendons maintenant le langage avec des types polymorphes de première class $\forall\alpha.\tau$ *comme dans le Système F. Proposer une extension de la définition des relations logiques pour le cas des types polymorphes.*

We now extend the language with first-class polymorphic types $\forall\alpha.\tau$ *as in System F. Propose an extension of the definition of logical relations for the case of polymorphic types.*

□

Listes / Lists

Nous repartons maintenant du λ-calcul simplement typé, sans polymorphisme ni types existentiels. Nous ajoutons à ce langage le type τ `list` des listes dont les éléments sont de type τ. Le langage des termes est étendu avec les constructeurs de liste `Nil` et `Cons`, ainsi qu'un mécanisme de récursion primitive sur les listes. Nous omettons les détails de ce langage : l'important est qu'il permet d'écrire beaucoup de fonctions sur les listes tout en préservant la propriété de normalisation forte.

On note

We start again from simply-typed λ-calculus, without polymorphism nor existential types. To this language, we add the type τ `list` of lists whose elements are of type τ. The language of terms is extended with the two list constructors `Nil` and `Cons`, and with primitive recursion over lists. We omit the details of this extended language: what matters is that it can express many functions over lists, while preserving strong normalization.

We write

$$[v_1; \ldots; v_n] = \texttt{Cons}(v_1, \ldots, \texttt{Cons}(v_n, \texttt{Nil}))$$

la liste de longueur n composée des valeurs $v_1, \ldots, v_n$. Une telle liste est elle-même une valeur. On étend la définition des relations logiques aux types τ `list` comme suit :

the list of length n containing the values $v_1, \ldots, v_n$. Such a list is itself a value. We extend the definition of logical relations to types τ `list` as follows:

$$\begin{aligned}\rho \vdash v \sim v' : \tau\ \texttt{list} \iff\ & \exists n, v_1, \ldots, v_n, v'_1, \ldots, v'_n, \\ & v = [v_1; \ldots; v_n] \ \wedge\ v' = [v'_1; \ldots; v'_n] \ \wedge\ \forall i \in [1, n],\ \rho \vdash v_i \sim v'_i : \tau\end{aligned}$$

Autrement dit, deux valeurs sont reliées au type τ `list` si et seulement si ce sont deux listes de même longueur dont les éléments sont deux-à-deux reliés au type τ. On admettra que le lemme fondamental des relations logiques reste vrai pour ce langage étendu.

In other words, two values are related at type τ `list` if and only if they are lists of the same length whose elements are pairwise related at type τ. We admit that the fundamental lemma still holds in this extended language.

Question 18

En utilisant le lemme fondamental, montrer qu'il n'existe pas de terme f tel que

Using the fundamental lemma, show that there does not exist any term f such that

$$\emptyset \vdash f : \alpha \ \texttt{list} \to \alpha$$

Answer: Assume, by way of contradiction, that $\emptyset \vdash f : \alpha \ \texttt{list} \to \alpha$ for some term f. We interpret α by the empty set of pairs of values: $\rho(\alpha) = \emptyset$. Under this interpretation, it is the case that $\rho \vdash [\,] \sim [\,] : \alpha \ \texttt{list}$. Using question 7 and the fundamental lemma, it follows that $\rho \vdash f\ [\,] \approx f\ [\,] : \alpha$. This implies that $f\ [\,]$ evaluates to some value v such that $(v, v) \in \rho(\alpha) = \emptyset$. This is impossible. □

Question 19

Soit f un terme tel que

Let f be a term such that

$$\emptyset \vdash f : \alpha \ \texttt{list} \to \texttt{int}$$

Montrer que le résultat de f ne dépend que de la longueur de sa liste argument, et non pas des valeurs qu'elle contient :

Show that the result of f depends only on the length of its argument list, and not on the values contained in this list:

$$\forall n, v_1, \ldots, v_n, v'_1, \ldots, v'_n, \quad f\ [v_1; \ldots; v_n] = f\ [v'_1; \ldots; v'_n]$$

Answer: We interpret α as

$$\rho(\alpha) = \{(v_1, v'_1); \ldots; (v_n, v'_n)\}$$

We have $\rho \vdash [v_1; \ldots; v_n] \sim [v'_1; \ldots; v'_n] : \alpha \ \texttt{list}$ by definition of $\sim$ over list types. By the usual argument, it follows that $\rho \vdash f\ [v_1; \ldots; v_n] \approx f\ [v'_1; \ldots; v'_n] : \texttt{int}$. Since being related at type `int` means having the same integer value, the result follows. □

Question 20

Soit f un terme tel que

Let f be a term such that

$$\emptyset \vdash f : \alpha \ \texttt{list} \to \alpha \ \texttt{list}$$

Montrer que tout élément de la liste résultat de f est un élément de sa liste argument :

Show that any element of the result list of f is an element of its argument list:

$$f\ [v_1; \ldots; v_n] \xrightarrow{*} [v'_1; \ldots; v'_k] \implies \{v'_1, \ldots, v'_k\} \subseteq \{v_1, \ldots, v_n\}$$

Soit $[v_1; \ldots; v_n]$ une liste de n valeurs closes. Montrer que :

Let $[v_1; \ldots; v_n]$ be a list of n closed values. Show that:

$$\exists k, i_1, \ldots, i_k. \quad f\ [1; \ldots; n] \xrightarrow{*} [i_1; \ldots; i_k] \ \wedge \ f\ [v_1; \ldots; v_n] \xrightarrow{*} [v_{i_1}; \ldots; v_{i_k}]$$

Comment caractériseriez-vous, informellement, les comportements possibles de la fonction f ?

How would you characterize informally the possible behaviors of function f?

Answer: For the first question, we interpret α as $\rho(\alpha) = \{(v_1, v_1); \ldots; (v_n; v_n)\}$. We have $\rho \vdash [v_1; \ldots; v_n] \sim [v_1; \ldots; v_n] : \alpha$ `list`, from which it follows that $\rho \vdash f\ [v_1; \ldots; v_n] \approx f\ [v_1; \ldots; v_n] : \alpha$ `list` and therefore that $\rho \vdash [v'_1; \ldots; v'_k] \sim [v'_1; \ldots; v'_k] : \alpha$ `list`. This means that for all $i \in [1, k]$, $(v'_i, v'_i) \in \rho(\alpha)$, implying that there exists a $j \in [1, n]$ such that $v'_i = v_j$.

For the second question, we interpret α as $\rho(\alpha) = \{(1, v_1); \ldots, (n, v_n)\}$. We have $\rho \vdash [1; \ldots; n] \sim [v_1; \ldots; v_n] : \alpha$ `list`, from which it follows that $\rho \vdash f\ [1; \ldots; n] \approx f\ [v_1; \ldots; v_n] : \alpha$ `list`.

Let u be the value of $f\ [1; \ldots; n]$ and w be the value of $f\ [v_1; \ldots; v_n]$. We know that $\rho \vdash u \sim w : \alpha$ `list`. Therefore, u and w are lists of the same length, whose elements are pairwise related by $\rho(\alpha)$.

By construction of $\rho(\alpha)$, we see that the elements of u are integers between 1 and n. We can therefore write $u = [i_1; \ldots; i_k]$. It follows that w is a list of k values, the first being related to i_1 by $\rho(\alpha)$, the second to i_2, etc. Therefore, $w = [v_{i_1}; \ldots; v_{i_k}]$.

Note that $k, i_1, \ldots, i_k$ are determined by $f\ [1; \ldots; n]$ and therefore depend only on n but not on the exact values of the v_i. It follows that the only possible behavior for f is to pick some elements from its argument list, possibly dropping some, duplicating some and dropping some, in a way that depends only on the length of its argument list. □