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 $\exists\alpha.\tau$ to be indistinguishable by the rest of the program.

On se place dans le λ-calcul simplement et implicitement typé, étendu avec des constantes entières (type `int`), et muni d'une sémantique en appel par valeur. On rappelle la syntaxe des types τ, des termes a et des valeurs v, ainsi que les règles de typage :

We consider the implicitly-typed simply-typed λ-calculus extended with integer constants (type `int`) and equipped with a call-by-value semantics. We recall the syntax of types τ, terms a, and values v, as well as the typing rules.

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

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

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?

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 argument entier N.

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

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'utilisent pas les fonctions récursives ? qui utilisent 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?

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 ρ qui associe à chaque variable 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 function ρ that maps every type variable α to a (possibly empty or infinite) set of pairs of closed 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:

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

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.

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

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

(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

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

Question 6

Montrer que les relations $\approx$ et $\sim$ coincident sur les valeurs : si v et v' sont des valeurs,

Show that the relations $\approx$ et $\sim$ coincide over values: if v and v' are values,

$$\rho \vdash v \approx v' : \tau \iff \rho \vdash v \sim v' : \tau$$

Question 7

Soient a, a', b, b' des termes. Démontrer que $\approx$ est stable par l'inverse de la relation de réduction :

Let a, a', b, b' be terms. Prove that $\approx$ is closed under the inverse of the reduction relation:

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

Question 8

Soient a, a', b, b' des termes. Démontrer que $\approx$ est stable par application, c'est-à-dire :

Let a, a', b, b' be terms. Prove that $\approx$ is closed under applications, that is:

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

Question 9

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

$$\Gamma \vdash a : \tau \quad \text{avec/with } \mathsf{dom}(\Gamma) = \{x_1, \ldots, x_n\} \tag{\textbf{A}}$$

Soit θ and θ' des substitutions des variables x_i par des valeurs closes

Let θ and θ' be substitutions of closed values for the variables x_i

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

telles que

such that

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

Démontrer que

Prove that

$$\rho \vdash \theta a \approx \theta' a : \tau \tag{\textbf{C}}$$

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.

Question 10

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:

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

Question 11

Nous étendons le langage avec un destructeur unaire `succ` de type $\mathit{int} \to \mathit{int}$, et avec la δ-réduction

We now extend the language with a unary destructor `succ` of type $\mathit{int} \to \mathit{int}$ and δ-reduction

$$\texttt{succ}\ N \to N + 1$$

Montrer que le lemme fondamental des relations logiques est préservé. (Décrire et détailler uniquement le nouveau cas à ajouter dans la preuve de la question 9.)

Show that the fundamental lemma of logical relation is preserved. (Show and detail only the case that must be added to the proof of question 9.)

Théorèmes gratuits ! / Theorems for free!

Question 12

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

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

Question 13

Soit f un terme tel que

Let f be a term such that

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

Donner deux exemples de termes f qui ont ce type et se comportent différemment. Puis montrer que tout terme f de ce type se comporte comme l'un de ces deux exemples. Plus précisément, en utilisant le lemme fondamental, prouver que

Give two possible terms f that have this type. Then, show that any term f with that type behaves like one of these two examples. More precisely, using the fundamental lemma, prove that

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

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.

Question 14

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 $(v_i)_{i\in\mathbb{N}}$ des valeurs des itérés de g par :

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

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

Construire une interprétation ρ telle que

Construct an interpretation ρ such that

$$\rho \vdash f \ \mathsf{succ}\ 0 \approx f \ g \ v : \alpha$$

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

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

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 où 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:

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

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

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

Les contexte d'évaluation sont étendus de la 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:

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

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

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.

On note

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*:

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

Question 15

Montrer que le lemme fondamental est encore valide avec les types existentiels : détailler dans la preuve de la question 9 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: in the proof of question 9, write the new cases for the new language constructs).

Comme nous l'avons vu en cours, les types existentiels permettent de modéliser les types de données abstraits. Par exemple, on s'intéresse à un type abstrait α ressemblant aux booléens et muni d'une constante `true` de type α, d'une fonction `not` de type $\alpha \to \alpha$ représentant la négation, et d'une fonction `istrue` de type $\alpha \to$ `bool`. L'interface de ce type abstrait peut être décrite par le type existentiel

As mentioned in the lectures, existential types model abstract data types. For instance, consider an abstract data type α that looks like booleans and is equipped of a constant `true` of type α, of a function `not` with type $\alpha \to \alpha$ representing boolean negation, and of a function `istrue` of type $\alpha \to$ `bool`. The interface of this abstract data type can be described by the existential type

$$\tau_{abool} = \exists\alpha.(\alpha \times (\alpha \to \alpha) \times (\alpha \to \texttt{bool}))$$

Voici deux implémentations de ce type abstrait. L'une représente le type α par des booléens concrets, l'autre le représente par des entiers avec la convention "entier pair = `true`" et "entier impair = `false`".

Here are two implementations of this abstract type. The first implementation represents type α by concrete booleans; the other, by integers, with the convention that "even integer = `true`" and "odd integer = `false`".

$$\begin{aligned} abool_1 : \tau_{abool} &= \mathsf{pack}\ (\texttt{true}, \texttt{not}, \lambda x.x) \\ abool_2 : \tau_{abool} &= \mathsf{pack}\ (0, \texttt{succ}, \lambda x.(x \bmod 2 = 0)) \end{aligned}$$

Le principe d'indépendance vis-à-vis des représentations, énoncé par J. C. Reynolds en 1983, dit qu'il doit être possible de remplacer une implémentation d'un type abstrait par une autre sans changer le comportement d'aucun programme bien typé utilisant ce type abstrait. Ainsi, dans l'exemple ci-dessus, $abool_1$ et $abool_2$ sont indistinguables par tout programme bien typé, alors même que ces deux paquetages implémentent le type abstrait par des types concrets `bool` et `int` qui ne sont pas isomorphes.

The principle of representation independence, due to J.C. Reynolds in 1983, states that it must be possible to replace an implementation of an abstract data type by another implementation, without changing the behavior of any well-typed program that uses this abstract data type. For instance, in the example above, $abool_1$ and $abool_2$ are indistinguishable by any well-typed program, even though those two packages implement the abstract type by concrete types `bool` and `int` that are not isomorphic.

Question 16

Soit $\exists\alpha.\tau$ l'interface d'un type de données abstrait et $\mathsf{pack}\ a_1, \mathsf{pack}\ a_2$ deux implémentations de cette interface :

Let $\exists\alpha.\tau$ be the interface of an abstract data type and $\mathsf{pack}\ a_1, \mathsf{pack}\ a_2$ be two implementations of this interface:

$$\emptyset \vdash \mathsf{pack}\ a_1 : \exists\alpha.\tau \qquad \emptyset \vdash \mathsf{pack}\ a_2 : \exists\alpha.\tau$$

Soit p un programme "client" paramétré par une implémentation de notre type de données abstrait, c'est-à-dire un terme tel que

Let p be a "client program" parameterized over an implementation of our abstract data type, that is, a term such that

$$\emptyset \vdash p : \exists\alpha.\tau \to \texttt{int}$$

Donner une condition suffisante sur les termes a_1 et a_2 qui garantit que, pour tout programme client p, les termes $p\ (\mathsf{pack}\ a_1)$ et $p\ (\mathsf{pack}\ a_2)$ s'évaluent à l'identique :

Give a sufficient condition over the terms a_1 and a_2 that guarantees that for any client program p, the terms $p\ (\mathsf{pack}\ a_1)$ and $p\ (\mathsf{pack}\ a_2)$ evaluate identically:

$$\forall p, N, \quad p\ (\mathsf{pack}\ a_1) \xrightarrow{*} N \iff p\ (\mathsf{pack}\ a_2) \xrightarrow{*} N$$

Question 17

Est-ce que votre condition suffisante est vraie pour les deux implémentations $abool_1$ et $abool_2$ du type abstrait τ_{abool} ? On admettra que le lemme fondamental reste vrai si on étend les relations logiques aux types `bool` et triplets de la manière évidente :

Does your sufficient condition hold for the two implementation $abool_1$ et $abool_2$ of the abstract type τ_{abool}? You can admit that the fundamental lemma still holds if we extend logical relations at types `bool` and triples in the obvious way:

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

Types universels / Universal types

Question 18

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.