Call-by name / Appel par nom
Extensible records / Enregistrements extensibles

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

2017/12/01 — Duration / durée: 3h00

Parts 1 and 2 are independent of one another.

Answers are judged by their correctness, but also by their clarity, conciseness, and precision. You don't have to justify answers when not asked.

Math displays are one-column and spread over the whole page width.

Les parties 1 et 2 sont indépendantes l'une de l'autre.

Les réponses sont jugées d'après leur correction, mais aussi d'après leur clarté, leur concision et leur précision. Vous n'avez pas besoin de justifier vos réponses lorsque ce n'est pas demandé.

Les formules mathématiques sont en simple colonne et s'étendent sur toute la largeur de la page.

1 Encoding call-by-name into call-by-value / Codage de l'appel par nom en appel par valeur

In this part, we study a program transformation whose purpose is to simulate a call-by-name evaluation strategy in a call-by-value calculus.

We work with the pure untyped λ-calculus.

Dans cette partie, nous étudions une transformation de programmes dont le but est de simuler une stratégie d'évaluation en appel par nom dans un calcul en appel par valeur.

Nous travaillons dans le λ-calcul pur non typé.

$$t, u ::= x \mid \lambda x.t \mid t \mathbin{@} t \qquad\qquad v ::= \lambda x.t$$

This calculus is equipped with a small-step operational semantics for the call-by-value strategy, defined as follows:

Ce calcul est muni d'une sémantique opérationnelle à petits pas pour la stratégie d'appel par valeur, définie par les règles suivantes :

$$\textsc{BetaV}\ \frac{}{(\lambda x.t) \mathbin{@} v \longrightarrow_{\text{cbv}} t[v/x]} \qquad \textsc{AppL}\ \frac{t \longrightarrow_{\text{cbv}} t'}{t \mathbin{@} u \longrightarrow_{\text{cbv}} t' \mathbin{@} u} \qquad \textsc{AppVR}\ \frac{u \longrightarrow_{\text{cbv}} u'}{v \mathbin{@} u \longrightarrow_{\text{cbv}} v \mathbin{@} u'}$$

The call-by-name strategy is defined as follows:

La stratégie d'appel par nom est définie comme suit :

$$\textsc{Beta}\ \frac{}{(\lambda x.t) \mathbin{@} u \longrightarrow_{\text{cbn}} t[u/x]} \qquad \textsc{AppL}\ \frac{t \longrightarrow_{\text{cbn}} t'}{t \mathbin{@} u \longrightarrow_{\text{cbn}} t' \mathbin{@} u}$$

We are interested in the following program transformation:

Nous nous intéressons à la transformation de programmes suivante :

$$\begin{aligned} [\![x]\!] &= !\,x \\ [\![\lambda x.t]\!] &= \lambda x.[\![t]\!] \\ [\![t \mathbin{@} u]\!] &= [\![t]\!] \mathbin{@} (\delta[\![u]\!]) \end{aligned}$$

The notations δt ("delay t") and $!\,t$ ("force t") are defined as follows:

$$\begin{array}{lcll} \delta t & = & \lambda x.t & \text{where } x \notin fv(t) \\ !\,t & = & t \mathbin{@} () & \text{where } () \text{ is an arbitrary closed value} \end{array}$$

As much as possible, please use these notations instead of expanding them and using the underlying terms.

Les notations δt ("retarder t") et $!\,t$ ("exiger t") sont définies ainsi :

Partout où cela est possible, efforcez-vous d'employer ces notations plutôt que de les expanser et de revenir aux termes sous-jacents.

Question 1

Indicate which reduction sequence, in call-by-value, arises out of the term $!\,(\delta t)$. Justify your answer simply by indicating which reduction rule(s) applies (apply) and why.

Indiquez quelle séquence de réductions, en appel par valeur, est issue du terme $!\,(\delta t)$. Justifiez votre réponse en indiquant simplement quelle(s) règle(s) de réduction s'applique(nt) et pourquoi.

□

Question 2

Consider the term $t = (\lambda x.x) \mathbin{@} (\lambda x.x)$. **a)** How does t reduce in call-by-name? **b)** What is its image $[\![t]\!]$ through the transformation? **c)** How does $[\![t]\!]$ reduce in call-by-value? Justify by indicating which reduction rules apply and why.

On considère le terme $t = (\lambda x.x) \mathbin{@} (\lambda x.x)$. **a)** Comment t se réduit-il en appel par nom ? **b)** Quelle est son image $[\![t]\!]$ à travers la traduction ? **c)** Comment $[\![t]\!]$ se réduit-il en appel par valeur ? Justifiez en indiquant quelles règles de réduction s'appliquent et pourquoi.

□

Question 3

Consider the term $t = (\lambda x.\lambda y.x) \mathbin{@} (\lambda x.x)$. **a)** How does t reduce in call-by-name? **b)** What is its image $[\![t]\!]$ through the transformation? **c)** How does $[\![t]\!]$ reduce in call-by-value? Justify by indicating which reduction rules apply and why.

On considère le terme $t = (\lambda x.\lambda y.x) @ (\lambda x.x)$. **a)** Comment t se réduit-il en appel par nom ? **b)** Quelle est son image $[\![t]\!]$ à travers la traduction ? **c)** Comment $[\![t]\!]$ se réduit-il en appel par valeur ? Justifiez en indiquant quelles règles de réduction s'appliquent et pourquoi.

□

Question 4

We conjecture that $t \longrightarrow_{\text{cbn}} u$ implies $[\![t]\!] \longrightarrow^{\star}_{\text{cbv}} [\![u]\!]$. Based on your answers to the previous questions, prove or disprove this conjecture.

On conjecture que $t \longrightarrow_{\text{cbn}} u$ implique $[\![t]\!] \longrightarrow^{\star}_{\text{cbv}} [\![u]\!]$. À l'aide de vos réponses aux questions précédentes, démontrez ou réfutez cette conjecture.

□

We introduce the notion of parallel call-by-value reduction, written $\Rightarrow_{\text{cbv}}$, defined by the following rules:

$$\textsc{ParaBetaV}\ \frac{t_1 \Rightarrow_{\text{cbv}} t_2 \qquad v_1 \Rightarrow_{\text{cbv}} v_2}{(\lambda x.t_1) \mathbin{@} v_1 \Rightarrow_{\text{cbv}} t_2[v_2/x]} \qquad \textsc{ParaVar}\ \frac{}{x \Rightarrow_{\text{cbv}} x}$$

$$\textsc{ParaLam}\ \frac{t_1 \Rightarrow_{\text{cbv}} t_2}{\lambda x.t_1 \Rightarrow_{\text{cbv}} \lambda x.t_2} \qquad \textsc{ParaApp}\ \frac{t_1 \Rightarrow_{\text{cbv}} t_2 \qquad u_1 \Rightarrow_{\text{cbv}} u_2}{t_1 \mathbin{@} u_1 \Rightarrow_{\text{cbv}} t_2 \mathbin{@} u_2}$$

The correctness of the program transformation $[\![\cdot]\!]$ is stated as a Simulation lemma, as follows:

On introduit la notion de réduction parallèle en appel par valeur, notée $\Rightarrow_{\text{cbv}}$, et définie par les règles suivantes :

La correction de la transformation de programmes $[\![\cdot]\!]$ s'énonce alors comme un lemme de Simulation qui prend la forme suivante :

Simulation: $t \longrightarrow_{\text{cbn}} u$ implies $[\![t]\!] \Rightarrow^{\star}_{\text{cbv}} [\![u]\!]$.

Question 5

Prove the Simulation lemma. A precise proof is expected, where each reasoning step is (briefly) justified. A certain Substitution property is required: propose its statement without proving it.

Démontrez le lemme de Simulation. On demande une preuve précise, où chaque étape de raisonnement est (brièvement) justifiée. Une certaine propriété de Substitution est nécessaire : proposez-en l'énoncé sans le démontrer.

□

In the following, a substitution is a total function of variables to terms, which almost everywhere is the identity.
The metavariables σ and θ denote substitutions. By convention, σ denotes a substitution that maps variables to "source" terms, whereas θ denotes a substitution that maps variables to "transformed" terms.
A relation $\mathcal{R}$ between two substitutions θ and σ is defined as follows:

Dans la suite, une substitution est une fonction totale des variables vers les termes, qui est presque partout l'identité. On note $t[\sigma]$ l'application d'une substitution σ à un terme t.
Les métavariables σ et θ dénotent des substitutions. Par convention, σ dénote une substitution qui à chaque variable associe un terme "source", tandis que θ dénote une substitution qui à chaque variable associe un terme "transformé".
On définit une relation $\mathcal{R}$ entre deux substitutions θ et σ comme suit :

$$\theta \,\mathcal{R}\, \sigma \qquad \Longleftrightarrow \qquad \forall x.\, !\,(\theta(x)) \Rightarrow_{\text{cbv}} [\![\sigma(x)]\!]$$

This definition allows stating a very general Substitution lemma:

L'intérêt de cette définition est de permettre l'énoncé d'un lemme de Substitution très général :

Substitution: $\theta \,\mathcal{R}\, \sigma$ implies $[\![t]\!][\theta] \Rightarrow_{\text{cbv}} [\![t[\sigma]]\!]$.

Question 6

Prove this Substitution lemma. A precise proof is expected, where each reasoning step is (briefly) justified.

Démontrez ce lemme de Substitution. On demande une preuve précise, où chaque étape de raisonnement est (brièvement) justifiée.

□

Question 7

Does the Substitution property that you proposed at Question 5 follow from the above Substitution lemma? Prove it.

La propriété de Substitution que vous avez proposée lors de la Question 5 découle-t-elle du lemme de Substitution ci-dessus ? Prouvez-le.

□

2 Extensible records / Enregistrements extensibles

The goal of this exercise is to add *extensible records* to System F. "Extensible" means that we can write a function that adds any field to any existing record, regardless of its other fields.

Le but de cet exercice est d'ajouter des *enregistrements extensibles* au Système F. "Extensible" signifie que nous pouvons écrire une fonction qui ajoute un champ à n'importe quel enregistrement, quels que soient ses autres champs.

2.1 Records with default values / Enregistrements avec valeurs par défaut

We assume given a countable collection of labels $\ell \in \mathcal{L}$. Records with default fields are defined on every field. Standard records with a finite number of fields will be addressed in the next section.

Nous supposons donnée une collection dénombrable d'étiquettes $\ell \in \mathcal{L}$. Les enregistrements avec champs par défaut sont des enregistrements définis sur tous les champs. Les enregistrements standards avec un ensemble fini de champs seront étudiés dans la section suivante.

We extend System-F types with record types $\langle \rho \rangle$ where ρ is a row that describes the type of the fields.

On étend les types du Système F avec des types d'enregistrements $\langle \rho \rangle$ où ρ est une rangée utilisée pour décrire le type des champs.

$$\tau ::= \langle \rho \rangle \qquad\qquad \rho ::= \varphi \mid \partial\tau \mid \ell : \tau;\ \rho$$

φ is a row variable; $\partial\tau$ describes a row where every label has type τ; finally, $\ell{:}\tau;\ \rho$ describes a row where label ℓ has type τ and others are described by ρ.

φ est une variable de rangée ; $\partial\tau$ décrit une rangée où toute étiquette a le type τ ; enfin $\ell : \tau;\ \rho$ décrit une rangée où l'étiquette ℓ a le type τ et les autres sont décrites par ρ.

In fact, rows are further constrained by kinds so that labels cannot be repeated in a row, and all occurrences of a single row variable φ represent the same set of labels: in addition to the kind T for ordinary types, we introduce a kind $\mathsf{R}\ L$ where $L \subset \mathcal{L}$ is a (finite) set of labels that the row should *not* define.

En fait, les rangées sont par ailleurs contraintes par des sortes afin que les étiquettes ne puissent pas être répétées dans une rangée, et que toutes les occurrences d'une même variable de rangée φ représentent le même ensemble d'étiquettes : en plus de la sorte T pour les types ordinaires, nous introduisons une sorte $\mathsf{R}\ L$ où $L \subset \mathcal{L}$ est un ensemble (fini) d'étiquettes que la rangée ne doit pas définir.

$$(\to) : \mathsf{T} \to \mathsf{T} \to \mathsf{T} \qquad \langle \_ \rangle : \mathsf{R}\ \emptyset \to \mathsf{T} \qquad \partial : \mathsf{T} \to \mathsf{R}\ L \qquad (\ell : \_;\ \_) : \mathsf{T} \to \mathsf{R}\ (\{\ell\} \uplus L) \to \mathsf{R}\ L \qquad \varphi_L : \mathsf{R}\ L$$

Notice that $(\ell : \_;\ \_)_{\ell \in \mathcal{L}}$ are a countable family of constants and symbols ∂ and $(\ell : \_;\ \_)_{\ell \in \mathcal{L}}$ have several kinds (L is any finite subset of labels).

Remarquez que $(\ell : \_;\ \_)_{\ell \in \mathcal{L}}$ forment une famille dénombrable de constantes et les symbols ∂ et $(\ell : \_;\ \_)_{\ell \in \mathcal{L}}$ ont plusieurs sortes (L est un sous-ensemble fini quelconque d'étiquettes).

Notation: We usually drop the parentheses in rows and simply write $\ell_1 : \tau_1;\ \ell_2 : \tau_2;\ \tau$. We also omit the kind subscripts of row variables and write φ rather than φ_L, but still require that all occurrences of the same variable have the same kind.

Notation : En général, nous ignorons les parenthèses pour les rangées et écrivons simplement $\ell_1 : \tau_1;\ \ell_2 : \tau_2;\ \tau$. Nous ignorons également l'indice des variables de rangée et écrivons φ plutôt que φ_L, mais demandons toujours que toutes les occurrences d'une même variable aient la même sorte.

Question 8

Which of the following types are well-kinded?

Lesquels des types ci-dessous sont bien sortés ?

$$\sigma_1 \stackrel{\text{def}}{=} \tau \to \partial\tau \qquad \sigma_2 \stackrel{\text{def}}{=} \langle \ell : \tau; \partial\tau \rangle \qquad \sigma_3 \stackrel{\text{def}}{=} \langle \ell : \tau_1; \ell : \tau_2; \varphi \rangle \qquad \sigma_4 \stackrel{\text{def}}{=} \langle \ell : \tau_1; \varphi \rangle \to \langle \varphi \rangle$$

□

We add the following equations of left commutativity (1) and idempotence (2):

Nous ajoutons les équations suivantes de commutativité gauche (1) et d'idempotence (2) :

$$\ell_1 : \tau_1;\ (\ell_2 : \tau_2;\ \tau) = \ell_2 : \tau_2;\ (\ell_1 : \tau_1;\ \tau) \quad (\mathbf{1}) \qquad\qquad \partial\tau = (\ell : \tau;\ \partial\tau) \quad (\mathbf{2})$$

Type equivalence $\equiv$ includes equations (1) and (2) and is closed by equivalence and congruence (*i.e.* two types with the same constructor applied to equivalent arguments are equivalent). Notice that type equivalence acts only on rows and leaves the rest of the types unchanged up to congruence. In particular, types are decomposable, *i.e.* $\tau_1 \to \tau_2 \equiv \tau_1' \to \tau_2'$ iff $\tau_1 \equiv \tau_1'$ and $\tau_2 \equiv \tau_2'$, and similarly for all type constructors. Moreover, we have:

L'équivalence $\equiv$ sur les types inclut les équations (1) et (2) et est close par équivalence et congruence (*i.e.* deux types avec le même constructeur appliqué à des arguments équivalents sont équivalents). Remarquez que l'équivalence n'agit que sur les rangées et laisse les autres types inchangés à congruence près. En particulier, les types sont décomposables, *i.e.* $\tau_1 \to \tau_2 \equiv \tau_1' \to \tau_2'$ ssi $\tau_1 \equiv \tau_1'$ et $\tau_2 \equiv \tau_2'$, et similairement pour tous les constructeurs de type. De plus, nous avons :

$$\begin{array}{rcl} (\ell_1 : \tau_1;\, \rho_1) \equiv (\ell_2 : \tau_2;\, \rho_2) & \iff & \exists \rho',\ \rho_1 \equiv (\ell_2 : \tau_2;\, \rho') \wedge \rho_2 \equiv (\ell_1 : \tau_1;\, \rho') \\ (\ell : \tau;\, \rho) \equiv \partial\sigma & \iff & \tau \equiv \sigma \wedge \rho \equiv \partial\sigma \end{array}$$

We treat typing rules of system F up to equivalence. You may assume that the main results of the course can be extended by treating types up to equivalence. Hereafter, we will just use equality to mean equality up to equivalence (*i.e.* equivalence).

Nous traitons les règles de typage du système F à équivalence près. Vous pouvez supposer que les résultats principaux du cours s'étendent à ce traitement des types modulo equivalence. Ci-après, nous utilisons juste l'égalité pour signifier égalité up to equivalence (*i.e.* equivalence).

We extend expressions with the following constants:

Nous étendons les expressions avec les constantes suivantes :

$$\partial : \forall\alpha.\, \alpha \to \langle \partial\alpha \rangle \qquad \langle \ell = \_;\, \_ \rangle : \forall\alpha.\, \forall\beta.\, \forall\varphi.\, \alpha \to \langle \ell : \beta; \varphi \rangle \to \langle \ell : \alpha; \varphi \rangle \qquad \_/\ell : \forall\alpha.\, \forall\varphi.\, \langle \ell : \alpha; \varphi \rangle \to \alpha$$

We write $\langle \ell_1 = M_1; \ell_2 = M_2; M \rangle$ for $\langle \ell_1 = M_1; \langle \ell_2 = M_2; M \rangle \rangle$. In examples (only), we assume the existence of other constants, such as integers, booleans, and a conditional.

Nous écrivons $\langle \ell_1 = M_1; \ell_2 = M_2; M \rangle$ pour $\langle \ell_1 = M_1; \langle \ell_2 = M_2; M \rangle \rangle$. Dans les exemples (seulement), nous supposons l'existence d'autres constantes, comme les entiers, les booléens et une conditionnelle.

We may omit type applications when the record constants are fully applied, unless explicitly asked otherwise.
For example, we write $\langle \ell = M; N \rangle$ instead of $\langle \ell = \_;\, \_ \rangle\ \tau\ \sigma\ \rho\ M\ N$.

Nous omettons les applications de types lorsque les constantes d'enregistrements sont entièrement appliquées, sauf s'il est explicitement demandé le contraire. Par exemple, nous écrivons $\langle \ell = M; N \rangle$ à la place de $\langle \ell = \_;\, \_ \rangle\ \tau\ \sigma\ \rho\ M\ N$.

Question 9

Explain, on this example, why we can non-ambiguously omit type applications in $\langle \ell = \_;\, \_ \rangle\ \tau\ \sigma\ \rho\ M\ N$, *i.e.* in any typing context, the type arguments τ, σ, and ρ are uniquely dertermined by M and N.

Expliquez, sur cet exemple, pourquoi nous pouvons omettre les applications de types dans $\langle \ell = \_;\, \_ \rangle\ \tau\ \sigma\ \rho\ M\ N$, *i.e.* dans m'importe quel contexte de typage, les types τ, σ et ρ sont déterminés uniquement par M et N.

□

Question 10

Assume h_1 and h_2 are different labels. Consider the terms $M_0 \stackrel{\text{def}}{=} \partial 0$ and $M_1 \stackrel{\text{def}}{=} \langle \ell_1 = \mathit{true}; M_0 \rangle$.

Supposons que h_1 et h_2 soient des étiquettes différentes. Soit les termes $M_0 \stackrel{\text{def}}{=} \partial 0$ and $M_1 \stackrel{\text{def}}{=} \langle \ell_1 = \mathit{true}; M_0 \rangle$.

a) Is M_1 well typed (if positive give its type)? Justify either by showing a contradiction or giving a fully explicitly typed term (including all optional type applications).

a) M_1 est-il typable (si oui, donnez son type) ? Justifiez en montrant une contradiction ou en donnant un terme complètement typé (y compris les applications de type optionnelles).

b) Same question for M_1/ℓ_1.

b) Même question pour M_1/ℓ_1.

c) Idem for M_0/ℓ_2.

d) Idem for M_1/ℓ_2.

e) Idem for (if true then M_0 else M_1)$/\ell_2$.

c) Idem pour M_0/ℓ_2.

d) Idem pour M_1/ℓ_2.

e) Idem pour (if true then M_0 else M_1)$/\ell_2$.

□

The δ-reduction for records is defined by:

La δ-réduction pour les enregistrements est définie par :

$$\partial V/\ell \longrightarrow V \quad (\textbf{Shar}) \qquad \langle \ell = V; W\rangle/\ell \longrightarrow V \quad (\textbf{Proj}) \qquad \langle \ell' = V; W\rangle/\ell \longrightarrow W/\ell \quad (\textbf{Skip})$$

Question 11

Describe record values (call them R).

Décrivez les valeurs enregistrements (que l'on appelera R).

□

Question 12

a) What is the evaluation of M_1 (give the reduction steps)? **b)** Same question for $M1/\ell_2$

a) Quelle est l'évaluation de $M_1\ell_2$ (donnez les étapes de réduction) ? **b)** Même question pour $M1/\ell_2$.

□

Question 13

Tell for each constant whether it is a constructor or a destructor.

Dites pour chaque constante si c'est un constructeur ou un destructeur.

□

Question 14

Show that a value of a record type is a record value.

Montrez qu'un valeur d'un type d'enregistrement est une valeur d'enregistrement.

□

Question 15

Show the soundness of the extension, reusing the framework (and all results) of the course.

Montrez la sûreté de cette extension, en réutilisant le cadre (et tous les résultats) du cours.

□

2.2 Standard records / Enregistrements standards

We now turn to standard records, which are only defined on a finite number of fields and show how to obtain then as a special case of records with defaults. We introduce a unary type constructor pre to described present fields and a nullary one abs to described absent fields. We introduce two value constructors Abs and Pre and a destructor get with the following types:

Nous nous intéressons maintenant aux enregistrements ordinaires, qui ne sont définis que sur un nombre fini de champs et nous montrons comment les obtenirs comme un cas particulier des enregistrements avec valeurs par défaut. Nous introduisons un constructeur de type unaire pre pour décrire les champs présents et un constructeur constant abs pour décrire les champs absents. Nous introduisons deux constructeurs de valeur Abs et Pre et un destructeur get avec les types suivants :

$$\mathsf{Pre}\colon \forall\alpha.\,\alpha \to \mathsf{pre}\ \alpha \qquad \mathsf{Abs} : \mathsf{abs} \qquad \mathsf{get}\colon \forall\alpha.\,\mathsf{pre}\ \alpha \to \alpha$$

We propose the following operations, which we call *standard operations*:

Nous proposons les operations suivantes, que nous appelons *opérations standards* :

$$\{\} \stackrel{\text{def}}{=} \partial\mathsf{Abs} \qquad \{\ell : M; N\} \stackrel{\text{def}}{=} \langle \ell : \mathsf{Pre}\ M; N\rangle \qquad M.\ell \stackrel{\text{def}}{=} ?$$

We write $\{\ell_1 = M_1; \ldots \ell_n = M_n; M\}$ as a shorthand for $\{\ell_1 = M_1; \ldots \{\ell_n = M_n; M\}\}$ and just $\{\ell_1 = M_1; \ldots \ell_n = M_n\}$ when M is the empty record $\{\}$.

Nous écrivons $\{\ell_1 = M_1; \ldots \ell_n = M_n; M\}$ comme raccourci pour $\{\ell_1 = M_1; \ldots \{\ell_n = M_n; M\}\}$ et juste $\{\ell_1 = M_1; \ldots \ell_n = M_n\}$ lorsque M est l'enregistrement vide $\{\}$.

Question 16

Give the definition of $M.\ell$ that accesses the ℓ-field of a record, assuming that the field is defined.

Donnez la définition de $M.\ell$ qui accède au champ ℓ d'un enregistrement en supposant que le champ est défini.

□

Question 17

Should the definition of the semantics be adjusted as a result of the introduction of Pre, Abs, and get?

La définition de la sémantique doit-elle être ajustée suite à l'introduction de Pre, Abs et get ?

□

Question 18

Give the types of standard operations ($\{\}$, $\{\ell : \_; \_\}$, and $\_.\ell$).

Donnez les types des opérations standards ($\{\}$, $\{\ell : \_; \_\}$, et $\_.\ell$).

□

Question 19

Explain concisely and reusing the framework of the course why type soundness is preserved for standard records.

Expliquez, de façon concise et en vous appuyant sur le cadre utilisé dans le cours, pourquoi la sûreté est préservée pour les enregistrements standards.

□

Question 20

We wish to restrict the formation of types such that Pre M and Abs may only appear as record fields and nothing else may appear as record fields. For example, expressions such as $\lambda x : \tau.\,\mathsf{Abs}$ or Pre Abs, or $\langle \ell_1 = (\lambda x : \tau.\, M); N\rangle$ are undesired.

a) Can you do so by giving each type symbol another kind signature independent of the previous one (*i.e.* types will have to validate both signatures to be well-formed)?

b) Why could this break the subject reduction lemma? How could we verify that subject reduction is actually preserved? (Don't do it.)

Nous souhaitons restreindre la formation des types de telle façon que Pre M et Abs ne puissent apparaître que dans des champs d'enregistrements et rien d'autre ne puisse apparaître dans les champs d'enregistrements. Par exemple, les expressions telles que $\lambda x : \tau.\,\mathsf{Abs}$ ou Pre Abs ou $\langle \ell_1 = (\lambda x : \tau.\, M); N\rangle$ sont indésirées.

a) Pouvez-vous faire cela en fournissant pour chaque symbol de type une signature de sorte indépendente de la précédente (*i.e.* les types devront satisfaire les deux signatures pour être valides) ?

b) Pourquoi cela peut-il casser le lemme d'autoréduction ? Comment pourrait-on vérifier que l'autoréduction est bien préservée ? (Ne le faites pas.)

□

Question 21

a) Assuming that standard operations ($\{\}$, $\{\ell : \_; \_\}$, and $\_.\ell$) are the only constants made available to the user, explain why absent fields are never read.

b) How could we formally state this property? (don't prove it.)

a) En supposant que les opérations standards ($\{\}$, $\{\ell : \_; \_\}$ et $\_.\ell$) sont les seules constantes rendues disponibles à l'utilisateur, expliquez pourquoi les champs absents ne sont jamais lus.

b) Comment pourriez-vous formuler cette propriété (ne la démontrez pas).

□

Question 22

Records with defaults keep the old values of fields that have been redefined, even though these will no longer be accessible.

a) How could you fix that?

b) What would then be record values?

Les enregistrements avec défauts conservent les anciennes valeurs des champs qui ont été redéfinis, bien que celles-ci ne seront plus accessibles.

a) Comment pourriez-vous corriger cela ?

b) Quelles seraient alors les valeurs d'enregistrements ?

□

3 Solutions

Question 1

We have $!\,(\delta t) \longrightarrow_{\mathrm{cbv}} t$. Indeed, $!\,(\delta t)$ is sugar for $(\lambda x.t)\ @\ ()$, where $x \notin fv(t)$ and $()$ is a (closed) value. Thus, the reduction rule BetaV applies. The reduct is $t[()/x]$, that is, t. (The term t can then possibly be further reduced.)

Question 2

a) We have $(\lambda x.x)\ @\ (\lambda x.x) \longrightarrow_{\mathrm{cbn}} \lambda x.x$, through Beta. As $\lambda x.x$ is a value, no further reductions are possible.

b) The term $[\![\lambda x.x]\!]$ is $\lambda x.\,!\,x$. Thus, $[\![(\lambda x.x)\ @\ (\lambda x.x)]\!]$ is $(\lambda x.\,!\,x)\ @\ (\delta(\lambda x.\,!\,x))$.

c) We have

$$\begin{array}{lll} (\lambda x.\,!\,x)\ @\ (\delta(\lambda x.\,!\,x)) & \longrightarrow_{\mathrm{cbv}} & !\,(\delta(\lambda x.\,!\,x)) \qquad \text{by BetaV, as } \delta t \text{ is a value} \\ & \longrightarrow_{\mathrm{cbv}} & \lambda x.\,!\,x \qquad\qquad \text{because } !\,(\delta t) \longrightarrow_{\mathrm{cbv}} t, \text{ as noted earlier} \end{array}$$

No further reductions are possible.

Question 3

a) We have $(\lambda x.\lambda y.x)\ @\ (\lambda x.x) \longrightarrow_{\mathrm{cbn}} \lambda y.\lambda x.x$, through Beta. As $\lambda y.\lambda x.x$ is a value, no further reductions are possible.

b) The term $[\![(\lambda x.\lambda y.x)\ @\ (\lambda x.x)]\!]$ is $(\lambda x.\lambda y.\,!\,x)\ @\ (\delta(\lambda x.\,!\,x))$.

c) We have

$$(\lambda x.\lambda y.\,!\,x)\ @\ (\delta(\lambda x.\,!\,x)) \quad \longrightarrow_{\mathrm{cbv}} \quad \lambda y.\,!\,(\delta(\lambda x.\,!\,x)) \qquad \text{by BetaV, as } \delta t \text{ is a value}$$

No further reductions are possible. We note that a "force/delay" redex $!\,(\delta(\lambda x.\,!\,x))$ appears, but is buried under a λ-abstraction, so cannot be reduced under call-by-value.

Question 4

The conjecture is false. The previous question provides a counter-example. Indeed, if the conjecture were true, we should have

$$[\![(\lambda x.\lambda y.x)\ @\ (\lambda x.x)]\!] \longrightarrow^{\star}_{\mathrm{cbv}} [\![\lambda y.\lambda x.x]\!]$$

that is,

$$(\lambda x.\lambda y.\,!\,x)\ @\ (\delta(\lambda x.\,!\,x)) \longrightarrow^{\star}_{\mathrm{cbv}} \lambda y.\lambda x.\,!\,x$$

but we instead have

$$(\lambda x.\lambda y.\,!\,x)\ @\ (\delta(\lambda x.\,!\,x)) \longrightarrow_{\mathrm{cbv}} \lambda y.\,!\,(\delta(\lambda x.\,!\,x)) \not\longrightarrow_{\mathrm{cbv}}$$

The "force/delay" redex buried under a λ-abstraction is the problem.

Question 5

The proof is by induction over the derivation of $t \longrightarrow_{\mathrm{cbn}} u$. The relation $\longrightarrow_{\mathrm{cbn}}$ is defined by two rules, Beta and AppL, so the proof has two cases. (No size argument is necessary! See the Coq version of this proof.)

Case Beta. The hypothesis is $(\lambda x.t)\ @\ u \longrightarrow_{\mathrm{cbn}} t[u/x]$. The goal is:

$$[\![(\lambda x.t)\ @\ u]\!] \Rightarrow^{\star}_{\mathrm{cbv}} [\![t[u/x]]\!]$$

By definition of the program transformation, the term on the left-hand side is $(\lambda x.[\![t]\!])\ @\ (\delta[\![u]\!])$. This term takes a step of parallel reduction, as follows:

$$(\lambda x.[\![t]\!])\ @\ (\delta[\![u]\!]) \Rightarrow_{\mathrm{cbv}} [\![t]\!][\delta[\![u]\!]/x]$$

(This is in fact also an ordinary call-by-value reduction step. It is easy to check that parallel call-by-value reduction is reflexive and therefore contains ordinary call-by-value reduction.) Thus, in order to prove the goal, there remains to establish the following fact:

$$[\![t]\!][\delta[\![u]\!]/x] \Rightarrow^{\star}_{\text{cbv}} [\![t[u/x]]\!]$$

This is the statement of a substitution lemma, which (as suggested) we assume and do not prove. The proof is carried later on (Questions 6 and 7).

Case AppL. The hypothesis is $t \mathbin{@} u \longrightarrow_{\text{cbn}} t' \mathbin{@} u$, where $t \longrightarrow_{\text{cbn}} t'$. The goal is

$$[\![t \mathbin{@} u]\!] \Rightarrow^{\star}_{\text{cbv}} [\![t' \mathbin{@} u]\!]$$

where (by induction) we may assume $[\![t]\!] \Rightarrow^{\star}_{\text{cbv}} [\![t']\!]$. By definition of the transformation, the goal is

$$[\![t]\!] \mathbin{@} \delta[\![u]\!] \Rightarrow^{\star}_{\text{cbv}} [\![t']\!] \mathbin{@} \delta[\![u]\!]$$

The result follows immediately from the fact that parallel reduction in the left-hand side of an application is permitted. (Technically, the fact that $t \Rightarrow_{\text{cbv}} t'$ implies $t \mathbin{@} u \Rightarrow_{\text{cbv}} t' \mathbin{@} u$ follows from ParaApp and from the fact that parallel reduction is reflexive. The fact that $t \Rightarrow^{\star}_{\text{cbv}} t'$ implies $t \mathbin{@} u \Rightarrow^{\star}_{\text{cbv}} t' \mathbin{@} u$ is proved by induction over the reduction sequence, using the previous fact. It was acceptable not to go into that level of detail.)

Question 6

The proof is by induction on the structure of the term t. Thus, the proof has three cases, for variables, λ-abstractions, and applications. (No size argument is necessary! See the Coq version of this proof.)

Case variable. The goal is $[\![x]\!][\theta] \Rightarrow_{\text{cbv}} [\![x[\sigma]]\!]$. We have:

$$\begin{array}{rll}
 & [\![x]\!][\theta] & \\
= & (!\,x)[\theta] & \text{by definition of the transformation} \\
= & !\,(x[\theta]) & \text{by definition of } ! \text{ and of substitution} \\
= & !\,(\theta(x)) & \text{by definition of substitution} \\
\Rightarrow_{\text{cbv}} & [\![\sigma(x)]\!] & \text{by the hypothesis } \theta \mathrel{\mathcal{R}} \sigma \\
= & [\![x[\sigma]]\!] & \text{by definition of substitution}
\end{array}$$

Case λ-abstraction. The goal is $[\![\lambda x.t]\!][\theta] \Rightarrow_{\text{cbv}} [\![(\lambda x.t)[\sigma]]\!]$. We have:

$$\begin{array}{rll}
 & [\![\lambda x.t]\!][\theta] & \\
= & (\lambda x.[\![t]\!])[\theta] & \text{by definition of the transformation} \\
= & \lambda x.([\![t]\!][\theta]) & \text{by definition of substitution, assuming } x \# \theta, \text{ w.l.o.g.} \\
\Rightarrow_{\text{cbv}} & \lambda x.[\![t[\sigma]]\!] & \text{by the induction hypothesis and by ParaLam} \\
= & [\![\lambda x.(t[\sigma])]\!] & \text{by definition of the transformation} \\
= & [\![(\lambda x.t)[\sigma]]\!] & \text{by definition of substitution, assuming } x \# \sigma, \text{ w.l.o.g.}
\end{array}$$

Case application. The goal is $[\![t \mathbin{@} u]\!][\theta] \Rightarrow_{\text{cbv}} [\![(t \mathbin{@} u)[\sigma]]\!]$. We have:

$$\begin{array}{rll}
 & [\![t \mathbin{@} u]\!][\theta] & \\
= & ([\![t]\!] \mathbin{@} (\delta[\![u]\!]))[\theta] & \text{by definition of the transformation} \\
= & [\![t]\!][\theta] \mathbin{@} \delta([\![u]\!][\theta]) & \text{by definition of } \delta \text{ and of substitution} \\
\Rightarrow_{\text{cbv}} & [\![t[\sigma]]\!] \mathbin{@} \delta[\![u[\sigma]]\!] & \text{by the induction hypotheses and by ParaApp, ParaLam} \\
= & [\![t[\sigma] \mathbin{@} u[\sigma]]\!] & \text{by definition of the transformation} \\
= & [\![(t \mathbin{@} u)[\sigma]]\!] & \text{by definition of substitution}
\end{array}$$

Question 7

The property proposed in the answer to Question 5 was:

$$[\![t]\!][\delta[\![u]\!]/x] \Rightarrow^{\star}_{\text{cbv}} [\![t[u/x]]\!]$$

Thus, it suffices to prove the stronger statement:

$$[\![t]\!][\delta[\![u]\!]/x] \Rightarrow_{\text{cbv}} [\![t[u/x]]\!]$$

To check that this statement follows from the Substitution lemma that was proved in the previous question, it suffices to check that two singleton substitutions are related, as follows:

$$[\delta[\![u]\!]/x] \; \mathcal{R} \; [u/x]$$

By definition of $\mathcal{R}$, we have to check that, for every variable y, the following holds:

$$!\,(y[\delta[\![u]\!]/x]) \Rightarrow_{\mathrm{cbv}} [\![y[u/x]]\!]$$

When $y = x$ holds, this boils down to

$$!\,(\delta[\![u]\!]) \Rightarrow_{\mathrm{cbv}} [\![u]\!]$$

which holds, as noted at Question 1. When $y \neq x$ holds, this boils down to

$$!\,y \Rightarrow_{\mathrm{cbv}} [\![y]\!]$$

which holds because $[\![y]\!]$ is $!\,y$ and parallel reduction is reflexive.

Question 8

Only σ_2 is well-formed.

Question 9

By convention $\langle \ell = M; N\rangle$ stands for $\langle \ell = \_; \_\rangle \; \tau \; \sigma \; \rho \; MN$, say M_0. In explicitly typed System F, this only makes sense in some context Γ. Assume $\Gamma \vdash M_0 : \tau_0$ for some τ_0. By inversion of typing, this inplies $\Gamma \vdash M : \tau$ and $\Gamma \vdash N : \langle \ell : \sigma; \rho\rangle$. In explicitly typed System F, this uniquely determines τ and $\langle \ell : \sigma; \rho\rangle$, which itself uniquely dertermines σ and ρ up to equivalence.

Question 10

a) M_0 is $\langle \_\rangle$ int 0. It has type $\langle \partial\mathsf{int}\rangle$, which is equivalent to $\langle \ell_1 : \mathsf{int}; \partial\mathsf{int}\rangle$.
M_1 is $\langle \ell_1 = \_; \_\rangle$ bool int $(\partial\mathsf{int})$ true M_0. It has type $\langle \ell_1 : \mathsf{bool}; \partial\mathsf{int}\rangle$.

b) M_1/ℓ_1 is $(\_/\ell_1)$ bool $(\partial\mathsf{int})$ M_1. It has type bool.

c) By idempotence M_0 also has type $(\ell_2 : \mathsf{int}; \partial\mathsf{int})$.
Hence, M_0/ℓ_2 is $(\_/\ell)$ int $(\partial\mathsf{int})$ M_0, which has type int.

d) Notice that M_0 also has types $\langle \ell_1 : \mathsf{bool}; \ell_2 : \mathsf{int}; \partial\mathsf{int}\rangle$, and by left associativity $\langle \ell_2 : \mathsf{int}; \ell_1 : \mathsf{bool}; \partial\mathsf{int}\rangle$.
M_1/ℓ_2 is $(\_/\ell_2)$ int $(\ell_1 : \mathsf{bool}; \partial\mathsf{int})$ M_1, which has type int.

e) This is ill-typed, since this would require M_0 and M_1 to have the same type,
i.e. $\langle \partial\mathsf{int}\rangle = \langle \ell_1 : \mathsf{bool}; \partial\mathsf{int}\rangle$. This would imply $\partial\mathsf{int} = \ell_1 : \mathsf{bool}; \partial\mathsf{int}$, which in turn would imply $\mathsf{int} = \mathsf{bool}$, which is false, indeed.

Question 11

Record values R are given by the grammar

$$R := \partial V \mid \langle \ell = V; R\rangle$$

i.e. of the form $\langle \ell_1 = V_1; \ldots \langle \ell_n = V_n; \partial V\rangle \ldots\rangle$; or $\langle \ell_1 = V_1; \ldots \ell_n = V_n; \partial V\rangle$ when using syntactic sugar.

Question 12

a) M_1 est une valeur et ne se réduit pas.

b) $M_1/\ell_2 \stackrel{\mathsf{def}}{=} \langle \ell_1 = true; \partial 0\rangle/\ell_2 \;\longrightarrow\; \partial 0/\ell_2 \;\longrightarrow\; 0$

Question 13

$\_/\ell$ is a destructor while ∂ and $\langle \ell = \_; \_\rangle$ are constructors.

Question 14

Assume $\bar{\alpha} \vdash V : \tau$. By case analysis, V is one of the following (two-by-two incompatible) forms:

- $\Lambda\alpha.\, W$; then τ is is a polymorphic type.
- $\lambda x.\, \alpha M$; then τ is an arrow type.
- a partial application of a primitive; then τ is a polymophic or an arrow type.
- a full application of a constructor:
  - ∂W; then τ is a record type.
  - $\langle \ell : V; W \rangle$; then τ is a record type.

(In each case, we used inversion of typing.) Observe that record types, polymorphic types, and arrow types are two-by-two incompatible. Since the only case where τ is a record type is the last one, this can only happen when values are full applications of constructors, *i.e.* record values.

Question 15

Reusing the parameterized framework, we just need to show that constants satisfy both progress and subject-reduction requirements for constants.

Progress for constants: The only destructor is $(\_/\ell)$. We must should that well-typed full applications to values do reduce. Assume $\bar{\alpha} \vdash V/\ell : \tau$ (**1**). We show that V/ℓ reduces. By inversion of typing applied to (1), we know that $\bar{\alpha} \vdash V : \langle \ell : \tau;\, \rho \rangle$ for some row ρ. In particular, V is a record value, hence of the form ∂W or $\langle \ell' : W;\, R \rangle$. In the former case, it reduces by (Shar). In the latter case, it reduces by (Proj) if $\ell = \ell'$; and by (Skip) otherwise.

Subject reduction for constants:

We must show that δ-reduction preserves well-typedness.

Subcase $\partial V/\ell \longrightarrow V$. Assume $\bar{\alpha} \vdash \partial V/\ell : \tau$. By inversion of typing, we have $\bar{\alpha} \vdash \partial V : \langle \ell : \tau;\, \rho \rangle$, which in turn implies $\bar{\alpha} \vdash V : \tau$, as expected.

Subcase $\langle \ell = V; W \rangle/\ell \longrightarrow V$. Assume $\bar{\alpha} \vdash \langle \ell = V; W \rangle/\ell : \tau$. By inversion of typing, we have $\bar{\alpha} \vdash \langle \ell = V; W \rangle : \langle \ell : \tau;\, \rho \rangle$ which in turn implies $\bar{\alpha} \vdash V : \tau$, as expected.

Subcase $\langle \ell' = V; W \rangle/\ell \longrightarrow W/\ell$. Assume $\bar{\alpha} \vdash \langle \ell' = V; W \rangle/\ell : \tau$. By inversion of typing, we have $\bar{\alpha} \vdash \langle \ell' = V; W \rangle : \langle \ell : \tau;\, \rho \rangle$, which in turn implies $\bar{\alpha} \vdash V : \sigma'$ and $\bar{\alpha} \vdash W : \rho'$ (**2**) with $\langle \ell' : \sigma';\, \rho' \rangle = \langle \ell : \tau;\, \rho \rangle$. Therefore, there exists ρ_0 such that $\rho' = \ell : \tau;\, \rho_0$ (**3**) and $\rho = \ell' : \sigma';\, \rho_0$. From (3) and (2), we have $\bar{\alpha} \vdash W/\ell : \tau$, as expected.

Question 16

$M.\ell \stackrel{\mathsf{def}}{=} \mathsf{get}\ (M/\ell)$

Question 17

Yes, we need a new reduction rule:

$$\mathsf{get}\ (\mathsf{Pre}\ V) \longrightarrow V$$

Question 18

$$\begin{array}{rcl} \{\} & : & \langle \partial\mathsf{Abs} \rangle \\ \{\ell : \_; \_\} & : & \forall\alpha.\, \forall\beta.\, \forall\varphi.\, \alpha \to \langle \ell : \beta; \varphi \rangle \to \langle \ell : \mathsf{Pre}\ \alpha; \varphi \rangle \\ \_.\ell & : & \forall\alpha.\, \forall\varphi.\, \langle \ell : \mathsf{Pre}\ \alpha; \varphi \rangle \to \alpha \end{array}$$

Question 19

We have shown type soundness of records with defaults. We have then introduced two independent types, pre and abs. which are algebraic datatypes. abs is isomomorphic to the unit type, which is sound. pre is a one-ary product, which is also sound.

We have three independent extensions, whose values belong to disjoint types. So the combination of extensions is also sound.

Notice that the encoding is just syntactic sugar, and is not involved with type soundness.

Question 20

a) We introduce two new kinds N for "normal" types and F for fields

$$\mathsf{pre} : \mathsf{N} \to \mathsf{F} \qquad \mathsf{abs} : \mathsf{F} \qquad \partial : \mathsf{F} \to \mathsf{F} \qquad (\ell : \_; \_) : \mathsf{F} \to \mathsf{F} \to \mathsf{F} \qquad \langle \_ \rangle : \mathsf{F} \to \mathsf{N}$$

(We need not distinguish fields from rows, since this previous signature already does that.)

b) Kinds restrict types. Fewer types means fewer well-typed programs. This could inadvertantly break subject reduction. Therefore, we must check that well-kindedness of types is preserved during reduction.

Question 21

a) Record access $\_.\ell$ calls record projection $\_/\ell$, but only when the field is actually there, thanks to the type of record access.

b) We could in fact restrict the type of $\_/\ell$ to $\forall\alpha.\,\forall\varphi.\,\langle \ell : \mathsf{pre}\ \alpha; \varphi\rangle \to \alpha$ and still prove subject reduction. This would ensure that absent fields are never projected, hence never read.

(We could even change the semantics of $\_/\ell$ so that it suceeds only on fields of the form $\mathsf{Pre}\ V$—hence does not reduce on Abs fields—and show type soundness in this setting.)

Question 22

a) We restrict record extension $\langle \ell : \_; \_\rangle$ to records that do not already have field ℓ:

$$\forall\alpha.\,\forall\beta.\,\forall\varphi.\,\alpha \to \langle \ell : \mathsf{Abs}; \varphi\rangle \to \langle \ell : \mathsf{Pre}\ \alpha; \varphi\rangle$$

Hence a field ℓ is never added to a row that already has field ℓ.

We then introduce a more flexible record extension as a new destructor:

$$\langle \ell \Leftarrow \_; \_\rangle \ : \ \forall\alpha.\,\forall\beta.\,\forall\varphi.\,\alpha \to \langle \ell : \sigma; \varphi\rangle \to \langle \ell : \mathsf{Pre}\ \alpha; \varphi\rangle$$

that performs cleaning during the construction of a new record, using the following reduction rules:

$$\begin{array}{lcl} \langle \ell \Leftarrow V; \partial W\rangle & \longrightarrow & \langle \ell = V; \partial W\rangle \\ \langle \ell \Leftarrow V; \langle \ell : V';\ R\rangle\rangle & \longrightarrow & \langle \ell \Leftarrow V; R\rangle \\ \langle \ell \Leftarrow V; \langle \ell' : V';\ R\rangle\rangle & \longrightarrow & \langle \ell' = V'; \langle \ell \Leftarrow V; R\rangle\rangle \end{array}$$

b) $\langle \ell_1 : \mathsf{Pre}\ \tau_1; \ldots \ell_n : \mathsf{Pre}\ \tau_n; \partial\mathsf{Abs}\rangle$