Université de Paris M1 Informatique

Preuves assistées par ordinateur
Examen du 3 mai 2021 – durée : 3h

Documents autorisés

Le barème ci-dessous est indicatif et est susceptible d'être modifié.

Exercice 1 ($\approx$ 6 points) – Déduction naturelle

1. Les séquents suivants sont-ils dérivables en déduction naturelle ? Si oui, en donner des dérivations en forme normale (au choix : soit en style arbre de formules, soit en style arbre de séquents, soit sous la forme d'un λ-terme de preuve). Si non, donner un contre-modèle (au choix : avec un modèle booléen qui rend le séquent faux ou en analysant les formes normales possibles).
   (a) $(\neg A) \vee B \;\vdash\; A \Rightarrow B$
   (b) $(A \wedge C) \vee (B \wedge C) \;\vdash\; (A \vee B) \wedge C$
   (c) $\forall x\,(A(x) \wedge B(x)) \;\vdash\; (\forall x\, A(x)) \wedge (\forall x\, B(x))$
   (d) $(\exists x\, A(x)) \wedge (\exists x\, B(x)) \;\vdash\; \exists x\,(A(x) \wedge B(x))$
   (e) $\exists x\,(A \Rightarrow B(x)) \;\vdash\; A \Rightarrow (\exists x\, B(x))$
2. On considère la preuve suivante de $\forall n\,\forall m\,\exists r\,(r \geq m \wedge r \geq n)$ représentée en λ-calcul :
$$H_1 := \lambda n.\,\lambda m.\,((n+1) * (m+2), (Ax_1\,n\,m, Ax_2\,n\,m))$$
   où Ax_1 et Ax_2 sont des axiomes donnés.
   On considère la preuve suivante de $\exists p\,(p \geq m \wedge p \geq n) \Rightarrow \exists q\,(q \geq 0)$, elle aussi formulée en λ-calcul :
$$H_2 := \lambda H.\texttt{match}\ H\ \texttt{with}\ (p, (x, y)) \;\Rightarrow\; (p+1, Ax_3\,(p+1))\ \texttt{end}$$
   où Ax_3 est un axiome.
   On considère la preuve suivante de $\exists q\,(q \geq 0)$:
$$H := H_2\,(H_1\,3\,4)$$
   Donner la forme normale de la preuve H de $\exists q\,(q \geq 0)$ (on pourra si nécessaire faire référence à Ax_1, Ax_2 et Ax_3). En particulier, quel est la valeur du témoin justifiant l'introduction du $\exists$?

Exercice 2 ($\approx$ 4 pts) : Schémas inductifs pour un type d'expressions arithmétiques

On considère le type `expr` défini en Coq par :

```
Inductive expr :=
| Cst : nat -> expr                (* Constante numérique:  n       *)
| Add : expr -> expr -> expr       (* Expression somme:     e1 + e2 *)
| Mul : expr -> expr -> expr.      (* Expression produit:   e1 * e2 *)
```

1. Indiquer le schéma de récursion canoniquement associé au type `expr` pour construire des objets de type `B`.

2. Indiquer le schéma d'induction canoniquement associé au type `expr` pour prouver des propriétés de la forme `P e` pour `P` un prédicat et `e` une expression.
3. Indiquer le schéma d'analyse de cas dépendant canoniquement associé au type `expr` pour prouver (sans appel récursif) des propriétés de la forme `T e` pour `T` un type dépendant et `e` une expression.
4. Indiquer une définition alternative de `expr` comme type du Système F dont les habitants en forme normale sont les codages de Church des habitants de `expr`.

Exercice 3 ($\approx$ 10 pts) : Évaluation d'expressions arithmétiques en Coq

A – Expressions arithmétiques On considère une nouvelle fois le type `expr` défini dans l'exercice 2.

1. Définir en Coq une fonction récursive `value : expr -> nat` calculant la valeur d'une expression.

B – Machine à environnement On considère une « machine à environnement » permettant d'évaluer des expressions. Les états de la machine, notés $\langle e \mid k \rangle$ sont des paires d'un code e de type `expr` et d'une continuation k de type `cont`. Le type `cont` est défini comme suit :

```
Inductive cont :=
| End : cont                 (* Fin du calcul *)
| AddL : expr -> cont -> cont   (* dit d'évaluer le 2e arg d'une addition *)
| AddR : nat -> cont -> cont    (* dit de finaliser le calcul d'une addition *)
| MulL : expr -> cont -> cont   (* dit d'évaluer le 2e arg d'une multiplication *)
| MulR : nat -> cont -> cont    (* dit de finaliser le calcul d'une multiplication *)
```

La machine obéit aux règles de réduction suivantes :

instructions disant comment évaluer le code

$\langle \texttt{Add}\ e_1\ e_2 \mid k \rangle$	$\longrightarrow$	$\langle e_1 \mid \texttt{AddL}\ e_2\ k \rangle$	évaluer e_1 avant de continuer par l'addition à e_2
$\langle \texttt{Mul}\ e_1\ e_2 \mid k \rangle$	$\longrightarrow$	$\langle e_1 \mid \texttt{MulL}\ e_2\ k \rangle$	évaluer e_1 avant de continuer par la multiplication avec e_2

instructions disant comment continuer quand le code est déjà évalué

$\langle \texttt{Cst}\ n \mid \texttt{AddL}\ e\ k \rangle$	$\longrightarrow$	$\langle e \mid \texttt{AddR}\ n\ k \rangle$	évaluer e avant de continuer par l'addition à n
$\langle \texttt{Cst}\ n \mid \texttt{AddR}\ m\ k \rangle$	$\longrightarrow$	$\langle \texttt{Cst}\ (m+n) \mid k \rangle$	réaliser l'addition de n et m
$\langle \texttt{Cst}\ n \mid \texttt{MulL}\ e\ k \rangle$	$\longrightarrow$	$\langle e \mid \texttt{MulR}\ n\ k \rangle$	évaluer e avant de continuer par la multiplication avec n
$\langle \texttt{Cst}\ n \mid \texttt{MulR}\ m\ k \rangle$	$\longrightarrow$	$\langle \texttt{Cst}\ (m*n) \mid k \rangle$	réaliser la multiplication de n et m

Pour évaluer une expression e, la machine commence dans l'état initial $\langle e \mid \texttt{End} \rangle$. Elle termine dans le seul type d'état pour lequel aucune règle ne s'applique, à savoir $\langle \texttt{Cst}\ n \mid \texttt{End} \rangle$.

2. Écrire en Coq une fonction `reduce : expr * cont -> option (expr * cont)` qui prend en argument un état, applique une étape de réduction et retourne le nouvel état. Si l'état est final, la fonction renvoie `None`.
3. On considère le type de trace d'exécution suivant :

   ```
   CoInductive trace :=
   | result : nat -> trace          (* La machine a atteint l'état final <Cst n|End> *)
   | reduction : (expr * cont) -> trace -> trace.  (* La machine continue de réduire *)
   ```

   Écrire une fonction `exec : expr -> trace` qui produit la trace de l'évaluation de la machine à environnement, c'est-à-dire la suite des étapes produites par `reduce`.

4. Écrire en Coq une fonction `compute : expr -> nat -> option nat` qui prend en argument une expression e et un entier n représentant un nombre d'étapes de calcul à faire, et qui exécute le processus suivant : initialiser la machine avec l'expression e, réduire n fois, renvoyer `Some p` si la machine a atteint un état final $\langle \texttt{Cst}\ p \,|\, \texttt{End} \rangle$ et renvoyer `None` sinon. On pourra si on le souhaite réutiliser `exec`.

C – Sémantique

5. Définir un prédicat inductif `multireduce : expr * cont -> expr * cont -> Prop` qui prend en argument deux états et qui exprime que la machine à environnement peut passer du premier état au second par un nombre arbitraire d'étapes de réduction.

D – Certification On prouve que la machine évalue correctement.

6. Formuler un énoncé Coq spécifiant que la machine abstraite évalue correctement les expressions, c'est-à-dire que pour toute expression e il existe un nombre d'opérations au bout desquelles `compute` renvoie un résultat qui est exactement la valeur de l'expression telle que calculée par `value`.
7. Formuler une autre spécification de la correction de la machine reposant cette fois sur `multireduce`.
8. Donner *informellement* les grandes lignes de la preuve de correction, en utilisant au choix soit la spécification 6. soit la spécification 7.

Rappels : le type `option` est défini en Coq par :

```
Inductive option (A:Type) :=
| Some : A -> option A
| None : option A.
```

et l'addition et la multiplication sur `nat` se nomment `Nat.add` (notation `+`) et `Nat.mul` (notation `*`).