Examen Intra

IFT-2035

May 26, 2021

Directives

- Répondre dans le *fichier de réponses* associé.
- Chaque question vaut 6 points, pour un total de 24+1 points.
- Les questions ne sont pas placées par ordre de difficulté.

0 Identification et code d'honneur **(1 point)**

1. Écrire votre nom et matricule dans le fichier de réponses.
2. Promettre de faire cet examen sans faire recours à de l'aide extérieure.

1 Syntaxe

Soient les expressions suivantes en notation préfixe:

1. `+ a * b - c d`
2. `* + - a b c d`
3. `+ ∧ a ∧ b c d`
4. `not = + a b - c d`
5. `&& not > a b = c sin + d 45`
6. `if < a 42 * b c sqrt d`

Après s'être souvenu que la mise à la puissance (représentée par ∧ ci-dessus) est associative à droite, écrire ces expressions en format infixe, postfixe, et en format parenthèsé à la Lisp. En notation infixe utiliser un *minimum* de parenthèses.

2 Types

Le code ci-dessous utilise une version simplifiée de Haskell qui n'a pas de classes de types et où les nombres sont tous des entiers de type Int.

Donner le type des expressions ci-dessous.
Par exemple, la réponse pour $(2, \lambda x \to x)$ serait: $(Int, \alpha \to \alpha)$
Le type donné devrait être aussi polymorphe (général) que possible.

1. $[(1,2),(\text{"1"},\text{"2"})]$
2. $([1,2],[\text{"1"},\text{"2"}])$
3. $\mathsf{map}\ (+)\ []$
4. $\mathsf{let}\ f\ x = [x,x]$
   $\quad g\ y = f\ y$
   $\mathsf{in}\ (f, f\ g)$

Donner le type des expressions suivantes et de leurs trous $\bullet$.
Par exemple, la réponse pour $(2, \bullet\ +\ 1)$ serait: (Int, Int) et Int

5. $map\ \bullet\ [(1,2)]$
6. $(map\ snd\ \bullet)\ ++\ [\text{"}hello\text{"}]$

Donner un exemple de code dont le type est celui demandé.
Par exemple, la réponse pour $\alpha \to \alpha$ devrait être: $\lambda x \to x$

7. $(\mathsf{Int}, \mathsf{Int}) \to \mathsf{Int}$
8. $\mathsf{Int} \to (\mathsf{Int}, \mathsf{Int})$
9. $(\mathsf{Int} \to \alpha \to \beta, \alpha) \to [\beta]$
10. $\mathsf{Int} \to \alpha$

Précisions: Les fonctions *map*, *fst*, *snd*, et $++$ sont (pré)définies comme suit:

$$map\ f\ [] = []$$
$$map\ f\ (x : xs) = f\ x : map\ f\ xs$$

$$fst\ (x, y) = x$$
$$snd\ (x, y) = y$$

$$++\ ::\ [\alpha] \to [\alpha] \to [\alpha]$$

3 Portée

Soit le code ci-dessous qui est écrit en Haskell et utilise donc la portée statique:

```
let a◯ b◯ = b◯ in
let c◯ b◯ = λd◯-> a◯ d◯ + b◯ in
let d◯ = λb◯-> b◯ (a◯ c◯) in
let c◯ (b◯, d◯) = b◯ (d◯, a◯)
in (d◯, λd◯-> c◯ (d◯, a◯))
```

Renommer toutes les variables (e.g. en y ajoutant un 0, 1, 2, ... dans le cercle) pour que chaque variable ait un nom différent des autres. Bien sûr ce renommage ne doit pas changer la sémantique du code.
Écrire juste les numéros ajoutés (dans l'ordre!) dans le fichier réponses.

4 Évaluateur

Soit l'évaluateur ci-dessous, écrit en Haskell, pour un langage fonctionnel proche du λ-calcul:

```
type Var = ...                  type Env = ...
data Val = Vnum Int             data Exp = Enum Int
         | Vfun Env Var Exp               | Evar Var
                                          | Efun Var Exp
                                          | Ecall Exp Exp
                                          | Elet Var Exp Exp
env_lookup :: Env -> Var -> (Env, Exp)
env_add    :: Env -> Var -> (Env, Exp) -> Env

eval (env, Enum n) = n
eval (env, Evar x) = env_lookup x env
eval (env, Ecall fun actual)
  = case eval fun of
      Vnum _ -> error "Un nombre n'est pas une fonction"
      Vfun env' formal body ->
        eval (env_add env' formal actual, body)
```

Où *env_lookup* et *env_add* sont des fonctions qu'on présume prédéfinies, qui permettent respectivement de trouver les infos d'une variable et d'ajouter une variable dans un environnement.

1. Haskell signale des erreurs de typage dans *eval*: Indiquer-les et montrer comment corriger le code.
2. Donner le type de *eval*.
3. *eval* implémente-t-il l'appel par nom (CBN) ou l'appel par valeur (CBV)? Justifier brièvement.
4. *eval* implémente-t-il la portée *statique* ou la portée *dynamique*? Justifier brièvement.
5. Indiquer de manière concise comment changer le code pour obtenir l'autre sorte de portée.
6. Haskell se plaint qu'*eval* n'est pas "exhaustive": Compléter sa définition.