TD Types

Dans la première partie de cette feuille, on reprend comme base le mini-langage d'expressions avec arithmétique et tableaux utilisé dans le cours. Rappel des règles de typage :

$$\frac{}{\Gamma \vdash n \ : \ \texttt{int}} \qquad \frac{\Gamma \vdash e_1 \ : \ \texttt{int} \qquad \Gamma \vdash e_2 \ : \ \texttt{int}}{\Gamma \vdash e_1 \ \texttt{-} \ e_2 \ : \ \texttt{int}} \qquad \frac{}{\Gamma \vdash x \ : \ \Gamma(x)}$$

$$\frac{\Gamma \vdash e_1 \ : \ \tau \qquad \ldots \qquad \Gamma \vdash e_k \ : \ \tau}{\Gamma \vdash \texttt{[}e_1\texttt{, } \ldots\texttt{, } e_k\texttt{]} \ : \ \tau\texttt{[]}} \qquad \frac{\Gamma \vdash e_1 \ : \ \tau\texttt{[]} \qquad \Gamma \vdash e_2 \ : \ \texttt{int}}{\Gamma \vdash e_1\texttt{[}e_2\texttt{]} \ : \ \tau}$$

Exercice 1 (Expressions typables) Peut-on trouver un type pour la variable `t` tel que les expressions suivantes soient typables ? Si oui, préciser les types possible et donner une dérivation, et si non expliquer l'incohérence.

1. `t[1] + 2`
2. `t[t[3]]`
3. `t[3][4]`
4. `t[3][t[4]]`
5. `[t[1], t[3]]`
6. `[t, t[0]]`

Correction :

1. *Ok, avec type* `int[]` *pour* `t` *et résultat de type* `int`.
2. *Ok, avec type* `int[]` *pour* `t` *et résultat de type* `int`.
3. *Ok, avec type* τ`[][]` *pour* `t` *et résultat de type* ☒.
4. *Non typable, car le type de* `t` *devrait avoir à la fois la forme* τ`[][]` *et la forme* `int[]`.
5. *Ok, avec type* τ`[]` *pour* `t` *et résultat de type* τ.
6. *Non typable, car le type* τ *de* `t` *devrait avoir la forme* $\tau = \tau'$`[]`, *avec* $\tau' = \tau$.

□

Exercice 2 (Paires) On veut étendre le langage avec une notion de paire. On ajoute à la syntaxe des expressions les trois constructions suivantes :

- $\texttt{(}e_1\texttt{, } e_2\texttt{)}$ pour la construction d'une paire avec les valeurs des expressions e_1 et e_2,
- $\texttt{fst(}e\texttt{)}$ pour l'extraction de la première composante de la paire obtenue en évaluant e,
- $\texttt{snd(}e\texttt{)}$ pour l'extraction de la deuxième composante,

On étend également la syntaxe des types avec la construction :

- $\tau_1 \times \tau_2$ pour le type des paires dont la première composante a le type τ_1 et la deuxième composante a le type τ_2.

Donner des règles de typage pour ce langage étendu.

Correction :

$$\frac{\Gamma \vdash e_1 \ : \ \tau_1 \qquad \Gamma \vdash e_2 \ : \ \tau_2}{\Gamma \vdash \texttt{(}e_2\texttt{, } e_2\texttt{)} \ : \ \tau_1 \times \tau_2} \qquad \frac{\Gamma \vdash e \ : \ \tau_1 \times \tau_2}{\Gamma \vdash \texttt{fst(}e\texttt{)} \ : \ \tau_1} \qquad \frac{\Gamma \vdash e \ : \ \tau_1 \times \tau_2}{\Gamma \vdash \texttt{snd(}e\texttt{)} \ : \ \tau_2}$$

□

Exercice 3 (Booléens) On veut étendre le langage avec des booléens. On ajoute :

- les constantes `true` et `false`,
- les opérations binaires e_1 `<` e_2, e_1 `=` e_2 et e_1 `&&` e_2,
- une expression conditionnelle $e_0 ? e_1 : e_2$.

Donner des règles de typage pour ce langage étendu. *Attention à ce que chaque opération soit aussi permissive que possible, mais pas plus que cela !* Que penser d'une expression de la forme $e_0 ? e_1$ sans résultat alternatif ? (vous pouvez réfléchir à ce qui se passe en caml avec une expression similaire)

Correction : On étend nos types avec un type de base `bool`.

$$\frac{}{\Gamma \vdash \texttt{true} : \texttt{bool}} \qquad \frac{}{\Gamma \vdash \texttt{false} : \texttt{bool}} \qquad \frac{\Gamma \vdash e_1 : \texttt{int} \quad \Gamma \vdash e_2 : \texttt{int}}{\Gamma \vdash e_1 \texttt{ < } e_2 : \texttt{bool}}$$

$$\frac{\Gamma \vdash e_1 : \tau \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash e_1 \texttt{ = } e_2 : \texttt{bool}} \qquad \frac{\Gamma \vdash e_1 : \texttt{bool} \quad \Gamma \vdash e_2 : \texttt{bool}}{\Gamma \vdash e_1 \texttt{ \&\& } e_2 : \texttt{bool}}$$

$$\frac{\Gamma \vdash e_0 : \texttt{bool} \quad \Gamma \vdash e_1 : \tau \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash e_0 \texttt{?} e_1 \texttt{:} e_2 : \tau}$$

Â noter : dans les cas `=` *et* `?`*, les expressions* e_1 *et* e_2 *ont des types arbitraires mais identiques. Pour le cas de* `=`*, on pourrait également accepter deux opérandes de types arbitraires, avec l'idée que deux valeurs de types différents seraient toujours considérées comme non égales.*

Si l'on voulait typer une expression de la forme $e_0 \texttt{?} e_1$*, il faudrait que* e_1 *ait le même type qu'une expression* e_2 *inexistante : on ne peut rien en faire dans notre système. En caml l'équivalent de cette forme est* `if` e_0 `then` e_1*, qui signifie en réalité* `if` e_0 `then` e_1 `else ()` *et impose donc que* e_1 *ait le type* `unit`.

□

Exercice 4 (Opérateurs paresseux) Dans le cadre de l'extension du langage avec les booléens, définir par des équations récursives une fonction F qui transforme une expression e en éliminant l'opérateur `&&` : chaque opération e_1 `&&` e_2 doit être remplacée par une expression conditionnelle.

Démontrer que si $\Gamma \vdash e : \tau$, alors $\Gamma \vdash F(e) : \tau$.

Correction : Le cas principal de la transformation est l'équation

$$F(e_1 \texttt{ \&\& } e_2) \quad = \quad F(e_1)\texttt{?}F(e_2)\texttt{:false}$$

Pour tous les autres cas, F est un morphisme.

On démontre la propriété par récurrence sur la dérivation de $\Gamma \vdash e : \tau$*. Seul cas intéressant :* $\Gamma \vdash e_1 \texttt{ \&\& } e_2 : \texttt{bool}$*, avec les prémisses* $\Gamma \vdash e_1 : \texttt{bool}$ *et* $\Gamma \vdash e_2 : \texttt{bool}$*. Par hypothèses de récurrence on a* $\Gamma \vdash F(e_1) : \texttt{bool}$ *et* $\Gamma \vdash F(e_2) : \texttt{bool}$*. En outre* $\Gamma \vdash \texttt{false} : \texttt{bool}$ *avec un type qui est bien égal au type de* e_2*, on peut donc appliquer la règle de typage du* `?` *pour obtenir* $\Gamma \vdash F(e_1)\texttt{?}F(e_2)\texttt{:false} : \texttt{bool}$.

□

Exercice 5 (Préservation du typage par substitution) On définit l'opération de substitution $e^{\{x \leftarrow e'\}}$ de la variable x par l'expression e' dans l'expression e par les équations suivantes.

$$\begin{aligned}
n^{\{x\leftarrow e'\}} &= n \\
(e_1 \texttt{ - } e_2)^{\{x\leftarrow e'\}} &= {e_1}^{\{x\leftarrow e'\}} \texttt{ - } {e_2}^{\{x\leftarrow e'\}} \\
y^{\{x\leftarrow e'\}} &= \begin{cases} e' & \text{si } x = y \\ y & \text{si } x \neq y \end{cases} \\
(e_1\texttt{[}e_2\texttt{]})^{\{x\leftarrow e'\}} &= {e_1}^{\{x\leftarrow e'\}}\texttt{[}{e_2}^{\{x\leftarrow e'\}}\texttt{]} \\
\texttt{[}e_1\texttt{, ..., }e_k\texttt{]}^{\{x\leftarrow e'\}} &= \texttt{[}{e_1}^{\{x\leftarrow e'\}}\texttt{, ..., }{e_k}^{\{x\leftarrow e'\}}\texttt{]}
\end{aligned}$$

Montrer que si on a les jugements $\Gamma \vdash e' : \tau'$ et $\Gamma, x : \tau' \vdash e : \tau$, alors on a également $\Gamma \vdash e^{\{x\leftarrow e'\}} : \tau$.

Correction : Récurrence sur la dérivation de $\Gamma, x : \tau' \vdash e : \tau$.

- *Cas* $\Gamma, x : \tau' \vdash n : \texttt{int}$ *immédiat.*
- *Cas* $\Gamma, x : \tau' \vdash e_1 \texttt{ - } e_2 : \texttt{int}$ *avec* $\Gamma, x : \tau' \vdash e_1 : \texttt{int}$ *et* $\Gamma, x : \tau' \vdash e_2 : \texttt{int}$*. Par hypothèse de récurrence on a* $\Gamma \vdash {e_1}^{\{x\leftarrow e'\}} : \texttt{int}$ *et* $\Gamma \vdash {e_2}^{\{x\leftarrow e'\}} : \texttt{int}$*. On en déduit* $\Gamma \vdash {e_1}^{\{x\leftarrow e'\}} \texttt{ - } {e_2}^{\{x\leftarrow e'\}} : \texttt{int}$*, avec justement* ${e_1}^{\{x\leftarrow e'\}} \texttt{ - } {e_2}^{\{x\leftarrow e'\}} = (e_1 \texttt{ - } e_2)^{\{x\leftarrow e'\}}$.
- *Cas* $\Gamma, x : \tau' \vdash y : \tau$*, avec* τ *le type associé à* y *par l'environnement étendu* $(\Gamma, x : \tau')$*. Deux sous-cas :*
  - *Si* $x = y$*, alors* $\tau = \tau'$ *et* $y^{\{x\leftarrow e'\}} = e'$*. On conclut alors car par hypothèse,* $\Gamma \vdash e' : \tau'$.
  - *Si* $x \neq y$*, alors* $\tau = \Gamma(y)$ *et* $y^{\{x\leftarrow e'\}} = y$*. Conclusion directe par la règle de typage des variables.*
- *Les deux cas des tableaux sont similaires au cas de la soustraction.*

□

Dans cette seconde partie, on regarde le typage d'un mini-langage fonctionnel, comportant de l'arithmétique, des variables locales et des fonctions. Voici les grammaires des expressions et des types de ce langage :

e	$::=$	n	constante entière
	$\mid$	e `-` e	opération binaire
	$\mid$	x	variable
	$\mid$	`let` x `=` e `in` e	définition locale
	$\mid$	`fun` x `->` e	fonction
	$\mid$	e e	application
τ	$::=$	`int`	entiers
	$\mid$	$\tau \to \tau$	fonctions

Et voici ses règles de typage :

$$\frac{}{\Gamma \vdash n \ : \ \texttt{int}} \qquad \frac{\Gamma \vdash e_1 \ : \ \texttt{int} \qquad \Gamma \vdash e_2 \ : \ \texttt{int}}{\Gamma \vdash e_1 \ \texttt{-} \ e_2 \ : \ \texttt{int}}$$

$$\frac{}{\Gamma \vdash x \ : \ \Gamma(x)} \qquad \frac{\Gamma \vdash e_1 \ : \ \sigma \qquad \Gamma, x : \sigma \vdash e_2 \ : \ \tau}{\Gamma \vdash \texttt{let } x \texttt{ = } e_1 \texttt{ in } e_2 \ : \ \tau}$$

$$\frac{\Gamma, x : \sigma \vdash e \ : \ \tau}{\Gamma \vdash \texttt{fun } x \texttt{ -> } e \ : \ \sigma \to \tau} \qquad \frac{\Gamma \vdash e_1 \ : \ \sigma \to \tau \qquad \Gamma \vdash e_2 \ : \ \sigma}{\Gamma \vdash e_1 \ e_2 \ : \ \tau}$$

Exercice 6 (Expressions typables) Les expressions suivantes sont-elles typables ? Si oui donner une dérivation, et si non expliquer l'incohérence.

1. `let f = fun x -> x+1 in f (f 1)`
2. `let f = fun x -> x+1 in f f`
3. `let f = fun x -> fun y -> x in f 1`
4. `let f = fun x -> fun y -> x in f 1 2 3`
5. `let f = fun x -> fun y -> x in f (fun z -> z) 2 3`
6. `fun x -> fun y -> fun z -> x z (y z)`

Correction :

1. *Typable.* f : *int* $\to$ *int et résultat int.*

$$\dfrac{\dfrac{\dfrac{}{x : int \vdash x \ : \ int} \qquad \dfrac{}{x : int \vdash 1 \ : \ int}}{x : int \vdash x + 1 \ : \ int}}{\vdash \ fun\ x \texttt{ -> } x + 1 \ : \ int \to int} \qquad \dfrac{\dfrac{}{f : int \to int \vdash f \ : \ int \to int} \qquad \dfrac{\dfrac{}{f : int \to int \vdash f \ : \ int \to int} \qquad \dfrac{}{f : int \to int \vdash 1 \ : \ int}}{f : int \to int \vdash f\ 1 \ : \ int}}{f : int \to int \vdash f\ (f\ 1) \ : \ int}$$

$$\vdash \ let\ f = fun\ x \texttt{ -> } x+1\ in\ f\ (f\ 1) \ : \ int$$

2. *Non typable.* f : *int* $\to$ *int non applicable à elle-même.*
3. *Typable.* f : *int* $\to \tau \to$ *int et résultat* $\tau \to$ *int pour un* τ *arbitraire.*
4. *Non typable.* f : *int* $\to$ *int* $\to$ *int donne* $f\ 1\ 2$: *int, non applicable au troisième argument* 3.
5. *Typable.* f : (*int* $\to$ *int*) $\to$ *int* $\to$ (*int* $\to$ *int*) *et fun z -> z* : *int* $\to$ *int pour un résultat int.*
6. *Typable, de type* $(\tau_0 \to \tau_1 \to \tau_2) \to (\tau_0 \to \tau_1) \to \tau_0 \to \tau_2$ *pour des* τ_i *arbitraires.*

□

Exercice 7 (Point fixe) On veut étendre le langage avec une définition récursive de variable locale :

— let rec $x = e_1$ in e_2

Donner une règle de typage pour cette nouvelle construction, et une dérivation de typage pour l'expression suivante :

```
let rec f =
  fun x -> 1 + f x
in
  f 0
```

Correction : Différence par rapport au let classique : le type τ_1 de la variable x introduite apparaît aussi dans le contexte de e_1.

$$\frac{\Gamma, x : \tau_1 \vdash e_1 : \tau_1 \qquad \Gamma, x : \tau_1 \vdash e_2 : \tau_2}{\Gamma \vdash \textit{let rec } x = e_1 \textit{ in } e_2 : \tau_2}$$

L'expression est typable, avec $f : int \to int$.

□

Exercice 8 (Monotonie) Pour deux environnements Γ et Δ, on note $\Gamma \subseteq \Delta$ si pour tout $x \in \mathrm{dom}(\Gamma)$ on a $x \in \mathrm{dom}(\Delta)$ et $\Gamma(x) = \Delta(x)$.

Montrer que si $\Gamma \vdash e : \tau$ et $\Gamma \subseteq \Delta$, alors $\Delta \vdash e : \tau$.

Correction : Récurrence sur la dérivation de $\Gamma \vdash e : \tau$. À noter, dans cette récurrence il faut maintenir Δ général : l'énoncé sur lequel on fait la récurrence est précisément $\forall \Delta, \Gamma \subseteq \Delta \implies \Delta \vdash e : \tau$.

- *Cas $\Gamma \vdash x : \Gamma(x)$. Par hypothèse d'inclusion on a $x \in dom(\Delta)$ et $\Gamma(x) = \Delta(x)$, donc $\Delta \vdash x : \Gamma(x)$.*
- *Cas $\Gamma \vdash fun\ x \text{ -> } e : \tau_1 \to \tau_2$ avec $\Gamma, x : \tau_1 \vdash e : \tau_2$. Soit Δ tel que $\Gamma \subseteq \Delta$. Alors par stabilité de l'inclusoin on a $(\Gamma, x : \tau_1) \subseteq (\Delta, x : \tau_1)$, et par hypothèse de récurrence on déduit $\Delta, x : \tau_1 \vdash e : \tau_2$ (c'est pour les cas comme celui-ci qu'on a besoin de l'énoncé généralisé). Par la règle d'inférence pour fun on déduit enfin $\Delta \vdash fun\ x \text{ -> } e : \tau_1 \to \tau_2$.*
- *...*

□