TD MIPS + Révisions

1 Programmation assembleur

1. Écrire des programmes MIPS qui calculent et affichent les résultats des expressions suivantes. On suppose que la valeur de la variable x est initialement stockée dans le registre `$a0`. Vous devez utiliser uniquement des registres (pas la mémoire), et aussi peu que possible.
   (a) $x^2 + 3x + 5$
   (b) $x(x+1)(x+2)(x+3)$
2. Écrire un programme MIPS traduisant le programme suivant. On supposera que la valeur de la variable x est stockée dans le registre `$a0`, et qu'une valeur renvoyée par `return` doit être placée dans le registre `$v0`.

```
int y = 0;
int z = 1;
while (y < x) {
  z = 2*z;
  y = y+1;
}
return z;
```

3. Traduire en MIPS la fonction suivante, en respectant la condition d'appel : les deux premiers arguments doivent être passés respectivement via les registres `$a0` et `$a1` et le résultat renvoyé via le registre `$v0`.

```
let rec compte n =
  if n = 0 then
    0
  else
    n mod 2 + compte (n/2)
```

2 Valeurs optionnelles

2.1 Analyse syntaxique

On s'intéresse à des expressions admettant une forme de valeur optionnelle similaire au type option de Caml. Ainsi, Some e désigne une valeur optionnelle calculée par e, et None désigne une valeur optionnelle absente. Pour accéder à la valeur d'une option e, on se donne une construction ?e : d, qui évalue l'expression e et teste la valeur obtenue :

— si e s'évalue en Some v alors le résultat est v,
— si e s'évalue en None alors le résultat est la valeur de l'expression d (qu'on appelle le résultat par défaut).

Voici la syntaxe complète de nos expressions, avec à gauche la grammaire elle-même et à droite son écriture en Menhir.

e	::=		*expressions*
	|	n	constante
	|	x	variable
	|	$e + e$	addition
	|	(e)	groupe
	|	None	absence de valeur
	|	Some e	valeur optionnelle
	|	$?e : e$	utilisation d'une option

```
expr:
  | INT
  | IDENT
  | expr PLUS expr
  | LPAR expr RPAR
  | NONE
  | SOME expr
  | QM expr COLON expr
```

Cette grammaire contient trois conflits *shift/reduce*. La figure **??** donnée à la fin du sujet contient des extraits du fichier .conflicts généré par Menhir.

Questions.

1. Pour chacun de ces trois conflits, donner
   — une entrée aboutissant au conflit, et
   — des arbres de dérivation justifiant les différentes possibilités.
2. On veut résoudre l'ambiguïté de la grammaire de sorte que l'expression

```
Some 1 + ?x:2 + 3
```

soit analysée comme

```
Some ((1 + ?x:2) + 3)
```

Quel est la construction la plus prioritaire ? La moins prioritaire ? Donner des déclarations de précédence et de priorité pour compléter la grammaire Menhir.

2.2 Types

On reprend les expressions avec valeurs optionnelles de l'exercice précédent. On définit un jugement de typage $\Gamma \vdash e : \tau$ signifiant que l'expression e est de type τ dans l'environnement Γ.

$$\frac{}{\Gamma \vdash n : \mathsf{int}} \qquad \frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau} \qquad \frac{\Gamma \vdash e_1 : \mathsf{int} \qquad \Gamma \vdash e_2 : \mathsf{int}}{\Gamma \vdash e_1 + e_2 : \mathsf{int}}$$

$$\frac{}{\Gamma \vdash \mathsf{None} : \tau\,\mathsf{option}} \qquad \frac{\Gamma \vdash e : \tau}{\Gamma \vdash \mathsf{Some}\, e : \tau\,\mathsf{option}}$$

Questions

1. Donner une règle de typage pour les expressions de la forme $?e_1 : e_2$.
2. Soit un environnement Γ tel que $\Gamma(\mathsf{x}) = \mathsf{int}$ et $\Gamma(\mathsf{y}) = \mathsf{int\ option}$. Les expressions suivantes sont-elle bien typées dans l'environnement Γ ? Donner si c'est possible une dérivation de typage.
   (a) `1 + Some 2`
   (b) `x + ?y:42`
   (c) `?None:(?Some y:Some x)`
3. Démontrer que l'expression `?Some e:Some d` est bien typée si et seulement si l'expression `Some(?e:d)` est bien typée, et que ces deux expressions ont alors le même type.

2.3 Compilation

Pour représenter en mémoire des valeurs optionnelles telles que celles du type `'a option` de Caml, on propose de procéder ainsi :

— La valeur `None` est représentée comme l'entier `0`.
— La valeur `Some v` est représentée par un pointeur vers un bloc alloué dans le tas, formé d'une part d'un entête contenant le nombre d'éléments dans le bloc (en l'occurrence, 1), et d'autre part d'un champ par élément du bloc (en l'occurrence, un unique champ pour l'unique élément `v`).

Note : la valeur `0` n'est jamais reconnue comme un pointeur valide.

Questions

1. Décrire l'état des registres et du tas après l'exécution du code suivant, et préciser la valeur contenue dans le registre `$v0`.

```
li   $a0, 8
li   $v0, 9
syscall
li   $t0, 1
sw   $t0, 0($v0)
li   $t0, 2
sw   $t0, 4($v0)
move $t0, $v0
li   $v0, 9
syscall
sw   $t0, 4($v0)
li   $t0, 1
sw   $t0, 0($v0)
```

2. Supposons que le registre `$a0` contienne la valeur `Some (Some (Some 3))`. Donner une configuration possible du tas, en précisant les adresses des blocs représentés et leur contenu, sachant que la première adresse du tas est `0x10040000`.
3. Supposons que le registre `$a0` contienne une valeur de la forme `Some (Some` n`)`. Écrire un fragment de code MIPS qui place la valeur n dans le registre `$v0`.
4. Donner le code MIPS pour une fonction f qui prend en entrée une valeur de type `int option` et est telle que :

$$\begin{aligned} f(\mathsf{None}) &= -1 \\ f(\mathsf{Some}\, n) &= n + 1 \end{aligned}$$

   Le paramètre est passé par le registre `$a0` et le résultat est passé par le registre `$v0`.
5. Supposons que les registres `$a0` et `$a1` contiennent chacun une valeur de type `int option`. Écrire un fragment de code MIPS qui écrit `1` dans le registre `$v0` si les deux valeurs sont égales, et `0` sinon.