UNIVERSITÉ GRENOBLE ALPES

UFR IM²AG

DÉPARTEMENT LICENCE SCIENCES ET TECHNOLOGIE

LICENCE SCIENCES & TECHNOLOGIES, 1re ANNÉE

UE INF201
ALGORITHMIQUE ET PROGRAMMATION FONCTIONNELLE

2021-2022

[git] • Branche master@e876d3f • Version v1.3.0-polyTP_partI.0 (14/01/2022) par François Puitg

EXERCICES DE TRAVAUX DIRIGÉS

Dans le titre de chaque exercice, problème ou annale d'examen est indiqué TDn ou TPn, où n est un entier entre 1 et 12 correspondant au numéro de séance de travaux dirigés ou pratiques **à partir duquel** l'énoncé peut être abordé.

Il y a plus d'énoncés que de séances ; ne pas hésiter à s'entraîner sur les énoncés qui n'auront pas été abordés en séance.

Il est fortement recommandé de réviser chaque partie en cherchant à résoudre un ou deux énoncés d'annales.

Table des matières

[git] • Branche master@e876d3f • Version v1.3.0-polyTP_partI.0 (14 janvier 2022) par François Puitg

Rappels

- *définir* = donner une définition,
  définition = spécification + réalisation ;
- *spécifier* = donner une spécification (le « quoi »),
  spécification = profil + sémantique + examples et/ou propriétés ;
- *réaliser* = donner une réalisation (le « comment »),
  réalisation = algorithme (langue naturelle) + implémentation (OCaml) ;
- *implémenter* = donner une inplémentation (OCaml).

Dans certains cas, certaines de ces rubriques peuvent être omises.

Première partie

TYPES, EXPRESSIONS ET FONCTIONS

Chapitre 1

Appropriation des notations

Objectif Se familiariser avec

- la notation fonctionnelle,
- la vérification du typage d'une expression.

Les exercices qui suivent sont présentés sous forme de tableaux à compléter, constitués des colonnes : contexte, expression, type et valeur. Une ligne d'un tableau doit être comprise de la façon suivante : dans ce contexte, l'expression a tel type et telle valeur.

L'expression, la valeur et le type doivent être indiqués tantôt sous forme littérale, tantôt en Ocaml, tantôt sous les deux formes.

Rappel L'évaluation d'une expression dépend des valeurs associées aux noms qui apparaissent dans l'expression. Le *contexte* d'une expression désigne les associations nom-valeur qui donnent un sens à cette expression. Un contexte est un ensemble d'associations $nom \mapsto valeur$. En mathématiques (resp. en OCaml), on le définit grâce à la construction « $soit\ nom = valeur$ » (resp. `let nom = valeur`). Lorsqu'une expression peut être évaluée sans contexte, on dit que son contexte d'évaluation est vide dénoté par le symbole $\emptyset$ (ensemble vide).

Sommaire

1.1 TD1 Notation des types et des valeurs

Q1. Compléter le tableau ci-dessous :

Si besoin, on pourra utiliser une fonction de conversion :

[git] • Branche master@e876d3f • Version v1.3.0-polyTP_partI.0 (14/01/2022) par François Puitg

SPÉCIFICATION 1	
PROFIL	`float_of_int` : $\mathbb{Z} \to \mathbb{R}$
SÉMANTIQUE	`(float_of_int x)` est le réel correspondant à l'entier `x`.

Cette fonction est prédéfinie dans la librairie standard d'OCaml.

1.2 TD1 Opérations logiques

En informatique, comme en mathématiques, la notion d'égalité apparaît dans différentes situations, et notamment :

- pour définir des liaisons nom ↦ valeur,
- pour effectuer des comparaisons entre valeurs.

En OCaml, ces deux situations sont dénotées par le même symbole : =. Dans les tableaux suivants, on prendra donc soin de ne pas confondre le = des colonnes «CONTEXTE» du = des colonnes «EXPRESSION».

Table de vérité des opérateurs logiques

Q1. Compléter le tableau ci-dessous :

	CONTEXTE	EXPRESSION	SÉMANTIQUE
N°		littérale ↔ OCaml	OCaml
1	`let` $a = true$	a ↔ .	....
2	`let` $a = false$	a ↔ .	....
3	`let` $a = false$	a = vrai ↔	....
4	`let` $a = true$	a = vrai ↔	....
5	`let` $a = false$	$\neg a$ ↔	....
6	`let` $a = true$	$\neg a$ ↔	....
7	`let` $a = false$	a = faux ↔	....
8	`let` $a = true$	a = faux ↔	....

Considérons la table de vérité de la conjonction, notée ∧ (`&&` en OCaml), usuellement appelée «et».

Q2. Compléter la table ci-dessous :

	CONTEXTE	EXPRESSION	SÉMANTIQUE
N°		litterale ↔ OCaml	OCaml
9	`let a = false and b = false`	a ∧ b ↔	.....
10	`let a = false and b = true`	a ∧ b ↔	.....
11	`let a = true and b = false`	a ∧ b ↔	.....
12	`let a = true and b = true`	a ∧ b ↔	.....

[git] • Branche master@e876d3f • Version v1.3.0-polyTP_partI.0 (14/01/2022) par François Puitg

N°	CONTEXTE	EXPRESSION littérale	EXPRESSION ↔ OCAML	TYPE math	TYPE ↔ OCAML	SÉMANTIQUE littérale ↔ OCAML
1	∅	3	↔ `3`	$\mathbb{Z}$	↔ `int`	3 ↔ `3`
2	∅	$-2,5+1,0$	↔ `-. 2.5.....`	$\mathbb{R}$	↔ `.....`	`.....` ↔ `.....`
3	∅	$3,4+2$	↔ `.....`	..	↔ `.....`	`.....` ↔ `.....`
4	∅	¬ vrai	↔ `.....`	.	↔ `.....`	`....` ↔ `.....`
5	∅	$3 \neq 2$	↔ `3<>2`	.	↔ `.....`	`....` ↔ `.....`
6	∅	2 + vrai	↔ `.....`		`.....`	`.....`
7	∅	'3'	↔ `....`	.....	↔ `.....`	.. ↔ `....`
8	∅	'a'	↔ `....`	.....	↔ `.....(*minuscule*)`	.. ↔ `....`
9	∅	''	↔ `...`	.....	↔ `.....`	... ↔ `.....`
10	∅	$'a' \leq 'b' \leq 'A'$	↔ `.....`	.	↔ `.....`	`....` ↔ `.....`
11	∅	x ou faux	↔ `.....`	.	↔ `.....`	. ↔ .
12	`let pi = 3.14`	$pi+1,0$	↔ `.....`	..	↔ `.....`	`....` ↔ `.....`
13	`let k = 3`	$2k$	↔ `2*k`	$2\mathbb{Z}$	↔ `.....(*pair*)`	. ↔ .

Considérons la table de vérité de la disjonction, notée $\vee$ (`||` en OCaml), usuellement appelée « ou ».

Q3. Compléter la table ci-dessous :

	CONTEXTE	EXPRESSION	SÉMANTIQUE
N°	OCAML	litterale ↔ OCAML	OCAML
13	.....	a ∨ b ↔	....
14	.....	a ∨ b ↔	....
15	.....	a ∨ b ↔	....
16	.....	a ∨ b ↔	....

Identités remarquables

On suppose que le contexte est quelconque.

Q4. En observant les tables de vérité de $\neg a$ ($\neg$ est la négation, usuellement appelée « non »), $a =$ vrai et $a =$ faux, déduire une simplification de chacune des expressions suivantes :

expression litterale.	forme simplifiée
$a =$ vrai	.
$a =$ faux	...

Q5. Donner une expression équivalente (colonne de droite) de chacune des expressions suivantes (colonne de gauche) en utilisant les opérateurs $\neg$ et $\vee$, mais sans utiliser l'opérateur $\wedge$:

expression			expression équivalente		
litterale	↔	OCAML	litterale	↔	OCAML
$\neg a \wedge \neg b$	↔	.....	.....	↔	.....
.....	↔	.....	$\neg(\neg a \vee \neg b)$	↔	.....

1.3 TD1 Opérations arithmétiques et logiques

Q1. Si l'expression est typable dans le contexte proposé, préciser son type et donner sa valeur.

[git] • Branche master@e876d3f • Version v1.3.0-polyTP_partI.0 (14/01/2022) par François Puitg

N°	CONTEXTE	EXPRESSION	TYPE	VALEUR
1	`let a = 4 and b = 7`	`a + b`	`int`	`11`
2	`let a = 4 and b = 2.3`	`a + b`	.....	.....
3	`let a = 4 and b = true`	`a + b`	.....	.....
4	`let a = 7 and b = 4`	`a < b`	....	.....
5	`let a = 4 and b = 7 and c = 5`	`a - b - c`	...	...
6	`let a = 4 and b = 7 and c = 5`	`a < b < c`	.....	.....
7	`let a = 4 and b = 7 and c = 5 and d = 3`	`a < b && c < d`	....	.....
8	`let a = 4 and b = 7 and c = 5`	`(a > b && a < c) || a < b`	....	....
9	`let a = 4 and b = 7 and c = 5`	`a > b && (a < c || a < b)`	....	.....
10	`let a = 4 and b = 7 and c = 5`	`(a > b && a < c) || a > b`	....	.....
11	`let a = 4 and b = 7 and c = 5`	`(a > b || a <= b) && a < c`	....	....
12	.....	`(a < b) || (a > b)`	....	`true`
13	.....	`(a < b) || (a >= b)`	....	....

1.4 TD1 Chaînes de caractères

Dans le tableau suivant, le contexte est noté mathématiquement comme un ensemble de liaisons nom ↦ valeur plutôt qu'avec la syntaxe OCaml `let ... = ...`

Q1. Si l'expression est typable dans le contexte proposé, préciser son type et donner sa valeur.

N°	CONTEXTE (math.)	EXPRESSION (OCaml)	TYPE (math. ↔ OCaml)	VALEUR (OCaml)
1	$a \mapsto$ «*hibou*» $b \mapsto'$ *x*′	`a ^ b`	..... ↔	.....
2	$a \mapsto$ «200» $b \mapsto 7$	`a ^ "b"`	..... ↔	.....
3	$a \mapsto$ «*hib*» $b \mapsto$ «*o*»	`a ^ b`	..... ↔	.....
4	$a \mapsto$ «␣» $b \mapsto$ «␣»	`a ^ b`	..... ↔	.....
5	$a \mapsto$ «*hib*» $b \mapsto$ «*o*»	`a @ b`	..... ↔	.....
6	$a \mapsto'a'$ $b \mapsto$ «*bc*»	`ab`	..... ↔	.....
7	$a \mapsto$ «*a*» $b \mapsto$ «*bc*»	`ab`	..... ↔	.....
8	$a \mapsto'a'$ $b \mapsto$ «*b*»	`"ab"`	..... ↔	.....
9	$a \mapsto$ «*B*» $b \mapsto$ «*A*»	`a ^ b ^ b ^ a`	..... ↔	.....
10	$a \mapsto$ "␣" $b \mapsto$ «*AB*»	`a ^ b ^ b ^ a`	..... ↔	.....
11	$a \mapsto$ «*a*» $b \mapsto$ « ^ »	`a b a`	..... ↔	.....

1.5 TD1 Expressions conditionnelles

L'expression `if … then … else …`

`if … then … else …` est une expression qui a un type et une valeur et donc peut être utilisée au cœur d'une autre expression !

Q1. Compléter le tableau suivant :

N°	CONTEXTE	EXPRESSION	TYPE	VALEUR
1	`let a = 4 and b = 7`	`if a < b then 3 else 4`	`int`	.
2	`let a = 4 and b = 7`	`(if a < b then 3 else 4) + a`	...	.
3	`let a = 4 and b = 7`	`if a < b then 3 else (4 + a)`	...	.

Un `if` sans `else` ?

On considère la fonction suivante :

SPÉCIFICATION 1 — valeur absolue

PROFIL $abs : \mathbb{Z} \to \mathbb{N}$

SÉMANTIQUE $abs\ (e)$ est la valeur absolue de e.

EX. ET PROP. (i) $abs\ (-3) = 3$

RÉALISATION 1

ALGORITHME expression conditionnelle déterminant le signe de e

IMPLÉMENT.

```
let abs (e:int) : int =
  if e<0 then -e
```

Q2. Pourquoi cette implémentation est-elle incorrecte ?

Q3. Donner une implémentation correcte.

Expressions conditionnelles à valeur booléenne

On suppose que le contexte est quelconque.

Q4. Donner une expression OCaml équivalente de chacune des expressions suivantes, en utilisant uniquement les opérateurs logiques :

N°	EXPRESSION	EXPRESSION ÉQUIVALENTE OCAML
1	`if a then true else false`	.
2	`if a then false else true`	.....
3	`if a then b else false`	.....
4	`if a then true else b`	.....
5	`if not a then true else b`	.....

[git] • Branche master@e876d3f • Version v1.3.0-polyTP_partI.0 (14/01/2022) par François Puitg

1.6 TD1 Vérification des types dans une expression

On s'intéresse à la vérification des types des noms qui apparaissent dans une expression algébrique par rapport aux spécifications des opérations qu'elles mettent en jeu.

Par exemple, pour que l'expression `if a then 1. else x` soit correcte du point de vue des types, les contraintes suivantes doivent être respectées :

- `a` doit être de type booléen,
- `x` doit être de type réel.

Ces contraintes sont la conséquence du fait que l'expression qui suit le `if` doit être de type booléen et que la construction `if then else` étant une expression, ses deux branches doivent avoir le même type. Si ces conditions sont respectées, l'expression `if a then 1. else x` est de type réel.

Autre exemple : dans l'expression `a = b`, la comparaison n'a de sens que si `a` et `b` sont de même type, disons t. Si cette contrainte est respectée, l'expression est de type booléen.

Types de base

Q1. Pour chacune des expressions ci-dessous, donner les contraintes de types que doivent respecter les noms y apparaissant, puis le type de l'expression si ces contraintes sont respectées, sinon proposer une correction de l'expression.

N°	EXPRESSION (OCAML)	CONTRAINTE	TYPE
1	$a \vee b$	....	..
2	`if a then (x && y) else t`	....	....
3	`3 + (if x=y+1 then a else b)`	....	....
4	$(a < b) \wedge (c < d)$	....	..
5	$a = (b <= c)$	....	..
6	`if not b then 5. else z`	....	..

Types construits, fonctions

Q2. Pour chacune des expressions ci-dessous, donner les contraintes de types que doivent respecter les noms y apparaissant, puis le type de l'expression si ces contraintes sont respectées.

- `(a+.1., not b, 5)`

- `let c=2 and d=4. in (c+3, d)`
- `if a && g then (a,b) else (x,g)`
- `let f x = x+.1. in f 5.`
- ```
  let h (u) (v) (z) = match u with
      | true -> if a=b then 7. else v
      | false -> z
  ```
  ```

Chapitre 2

Types, expressions et fonctions

Sommaire

2.1 TD1 Signe du produit

Étant donnés deux entiers, on veut – **sans** calculer leur produit – déterminer si le produit des deux nombres est positif ou nul ou bien s'il est strictement négatif.

Q1. Compléter ci-dessous la spécification de la fonction *prodPos* qui répond à cette demande :

SPÉCIFICATION 1 — signe du produit

PROFIL prodPos : .....

SÉMANTIQUE ($prodPos\ x\ y$) est vrai si et seulement si le signe du produit xy est positif ou nul

EX. ET PROP. (i) $(prodPos\ -2\ -3) = \ldots.$

(ii) $\forall x \in \mathbb{Z}, (prodPos\ x\ 0) = \ldots.$

(iii) ..... = faux

Q2. Compléter chacune des implémentations suivantes de la fonction *prodPos*. On prendra soin :

- d'indenter le code,
- de préciser en commentaire quelles sont les expressions booléennes au niveau des `else`.

RÉALISATION 1 — signe du produit

ALGORITHME a) analyse par cas dirigée par le résultat de la fonction et composition conditionnelle

IMPLÉMENT.
```
let prodPos_1 ..... =                                    1
    if ..... then                                        2
        true                                             3
    else .....                                           4
       false                                             5
```

ALGORITHME b) analyse par cas dirigée par les données de la fonction et composition conditionnelle

IMPLÉMENT.
```
let prodPos_2 ..... =                                    1
    if x > 0 then                                        2
    .....                                                3
    .....                                                4
    .....                                                5
    .....                                                6
    else (* x ≤ 0 *)                                     7
    .....                                                8
    .....                                                9
    .....                                                10
    .....                                                11
    .....                                                12
    .....                                                13
    .....                                                14
```

Q3. L'étudiant JeSuisPlusMalin propose une implémentation *prodPos_2'* identique à *prodPos_2* mais sans les lignes 9, 13 et 14. Son implémentation respecte-elle la spécification de *prodPos* ? Si non, donner un contre-exemple.

2.2 TD1 Une date est-elle correcte ?

On considère une date représentée par deux entiers j et m : j est le numéro du jour dans le mois et m est le numéro du mois dans l'année. On veut déterminer si les deux entiers correspondent à une date valide d'une année non bissextile[1]. On spécifie pour cela les types *jour* et *mois* ainsi qu'une fonction *estJourDansMois* :

DÉFINITION D'ENSEMBLES

déf $jour = \{1, \ldots, 31\}$

déf $mois = \{1, \ldots, 12\}$

SPÉCIFICATION 1 — jour d'un mois	
PROFIL	estJourDansMois : .....
SÉMANTIQUE	(*estJourDansMois j m*) est *vrai* si et seulement si j et m caractérisent respectivement le jour et le mois d'une date d'une année non bissextile.
EX. ET PROP.	(i) (*estJourDansMois* 28 1) = ...
	(ii) (*estJourDansMois* 31 4) = ...
	(iii) (*estJourDansMois* 18 13) n'a pas de sens puisque .....
	(iv) (*estJourDansMois* 0 4) n'a pas de sens puisque .....

Q1. Compléter la spécification de la fonction *estJourDansMois* ci-dessus, et l'implémentation des ensembles *jour* et *mois* ci-dessous :

```
type jour = .... (* restreint à ..... *)   1
type mois = .... (* restreint à ..... *)   2
```

Q2. Compléter la réalisation de la fonction *estJourDansMois* avec deux implémentations différentes basées sur les algorithmes suivants :

RÉALISATION 1 — jour d'un mois	
ALGORITHME	1) composition conditionnelle sous forme d'expressions conditionnelles imbriquées examinant successivement les 3 cas : mois à 31, mois à 28, mois à 30.
IMPLÉMENT.	. . .
ALGORITHME	2) composition booléenne examinant successivement les 3 cas : mois à 31, mois à 28, mois à 30. L'utilisation d'expressions conditionnelles est interdite.
IMPLÉMENT.	. . .

[1] Le mois de février des années non bissextiles a 28 jours.

[git] • Branche master@e876d3f • Version v1.3.0-polyTP_partI.0 (14/01/2022) par François Puitg

2.3 TD1 Quelle heure est-il ?

Le type *horaire* **.** On étudie la représentation de l'heure affichée par une montre en mode AM/PM.

Les heures sont limitées à l'intervalle $\{0, \dots, 11\}$.

L'indication *am*/*pm* indique si l'heure du jour est située entre minuit et midi (avant midi, en latin *ante meridiem*, abrégé en *am*) ou entre midi et minuit (après midi, en latin *post meridiem*, abrégé en *pm*).

On choisit de représenter une date horaire par un quadruplet formé de trois entiers et la valeur Am ou Pm de type énuméré *meridien* .

Les entiers représentent les heures, minutes et secondes.

On définit ainsi les ensembles suivants :

déf $heure = \{0, \dots, 11\}$
déf $minute = \{0, \dots, 59\}$
déf $seconde = \{0, \dots, 59\}$
déf $meridien = \{\text{Am}, \text{Pm}\}$
déf $horaire = heure \times minute \times seconde \times meridien$

Exemples :

- Le quadruplet $(3, 0, 0, \text{Pm})$ représente l'heure exprimée habituellement par 3 : 00 : 00 PM.
- Le quadruplet $(2, 25, 30, \text{Am})$ représente l'heure 2 : 25 : 30 AM.
- Noter que $(0, 0, 0, \text{Am})$ correspond à minuit et que $(0, 0, 0, \text{Pm})$ correspond à midi.

Incrémenter l'horaire d'une seconde. On considère la fonction suivante :

SPÉCIFICATION 1 — incrémenter d'une seconde	
PROFIL	$inc\_hor : horaire \to horaire$
SÉMANTIQUE	$inc\_hor\ (h)$ est l'horaire qui suit d'une seconde l'horaire h.
EX. ET PROP.	(i) $inc\_hor\ (2, 25, 30, \text{Am}) = (2, 25, 31, \text{Am})$ (ii) $inc\_hor\ (11, 59, 59, \text{Pm}) = (0, 0, 0, \text{Am})$

Q1. Implémenter les types *heure*, *minute*, *seconde*, *meridien* et *horaire*, puis donner une réalisation de la fonction *inc_hor*.

Q2. Utiliser la fonction *inc_hor* pour réaliser la fonction *ajout3sec* :

SPÉCIFICATION 2	
PROFIL	$ajout3sec : horaire \to horaire$
SÉMANTIQUE	$ajout3sec\ (h)$ est l'horaire qui vient 3 secondes après h.

2.4 TD2 Fractions

On étudie un type correspondant à la notion de fraction positive ou nulle. On veut pouvoir :

- construire des fractions irréductibles à partir de deux entiers correspondant au numérateur et au dénominateur,

- disposer de certaines opérations sur les fractions.

Il s'agit de spécifier le type *fraction* et une fonction de construction associée. Un objet de type *fraction* est un couple d'entiers formé du numérateur et du dénominateur et tels que leur plus grand diviseur commun (*pgcd*) vaut 1 ; la fraction est donc sous forme irréductible :

$$\textbf{\textit{déf}} \quad fraction = \{(n,d) \in \mathbb{N} \times \mathbb{N}^* \mid pgcd(n,d) = 1\}$$

Q1. Implémenter le type *fraction*.

La fonction de construction d'un objet de type *fraction*, nommée *frac* a pour paramètres deux entiers *num* et *den* correspondants à la réduction de la fraction $\frac{num}{den}$.

EX. ET PROP. (i) (*frac* 21 60) construit le couple correspondant à la réduction de la fraction $\frac{21}{60}$: (*frac* 21 60) = (7, 20)
(ii) (*frac* 42 120) = (7, 20)
(iii) (*frac* 7 20) = (7, 20)

Pour mettre une fraction sous forme irréductible, on dispose d'une fonction de calcul du plus grand diviseur commun de deux entiers, dont le profil est le suivant :

PROFIL pgcd : $\mathbb{Z}^* \to \mathbb{Z}^* \to \mathbb{Z}^*$

Q2. Compléter la définition de la fonction *frac* ci-dessous :

SPÉCIFICATION 1 — construction de fractions	
PROFIL	frac : .....
SÉMANTIQUE	.....
RÉALISATION 1	
ALGORITHME	.....
IMPLÉMENT.	/!\ conditions utilisation : ....

2.5 TD2 Géométrie élémentaire

Un point dans le plan Euclidien est repéré par ses coordonnées (abscisse et ordonnée).

Q1. Définir les ensembles mathématiques associés aux abscisses, ordonnées et coordonnées d'un point.

Q2. Donner la spécification de la fonction *longueur* qui calcule distance entre deux points.

Q3. Donner une réalisation de *longueur*, la fonction « racine carrée » – prédéfinie en OCaml – ayant le profil suivant : `sqrt` : $\mathbb{R}^+ \to \mathbb{R}^+$. La réalisation de *sqrt* n'est pas demandée.

On souhaite manipuler des figures géométriques telles que :

- les *triangles*, définis par leurs trois sommets ;
- les *cercles*, définis par leur centre et leur rayon ;
- les *rectangles*, dont les cotés sont parallèles aux axes, définis par leur coin inférieur gauche, et leur coin supérieur droit.

On définit pour cela l'ensemble *figure* suivant :

$$\begin{aligned} \textbf{\textit{déf}} \quad figure \quad = \quad & \{ \textsc{Triangle}(t) \mid t \in \ldots\ldots\ldots\ldots\ldots\ldots \} \bigcup \\ & \{ \textsc{Cercle}(.) \mid \ldots\ldots\ldots\ldots\ldots \} \bigcup \\ & \{ \ldots\ldots\ldots\ldots\ldots \mid \ldots\ldots\ldots\ldots\ldots\ldots \} \end{aligned}$$

Q4.

a) Que représente le terme Triangle dans cette définition ?

b) Qu'est-ce-que t ? Quel est son type ? Comment appelle-t-on un tel type ?

c) Compléter la définition du type *figure* ci-dessus.

d) Donner une implémentation de ce type.

Q5. Donner la spécification de la fonction *peri* qui calcule le périmètre d'une figure.

Q6. Réaliser la fonction *peri* sans oublier d'indiquer l'algorithme retenu.

2.6 TD2 Relations sur des intervalles d'entiers

On caractérise un intervalle **fermé** d'entiers par la donnée de ses bornes inférieure et supérieure :

$$\textbf{\textit{déf}} \quad intervalle = \mathbb{Z} \times \mathbb{Z}$$

Soit $bi, bs \in \mathbb{Z}$; le couple (bi, bs) décrit un intervalle d'entiers de borne inférieure bi et de borne supérieure bs, à condition que la contrainte $bi \leq bs$ soit respectée.

2.6.1 Relations entre points et intervalles

Pour situer un entier par rapport à un intervalle, on spécifie les trois fonctions suivantes :

SPÉCIFICATION 1 — prédicats sur les intervalles	
PROFIL	dans : $\mathbb{Z} \to intervalle \to \mathbb{B}$
SÉMANTIQUE	$(dans\ x\ i) = vrai \Leftrightarrow x \in i$
PROFIL	précède : $\mathbb{Z} \to intervalle \to \mathbb{B}$
SÉMANTIQUE	$(précède\ x\ i)$ si et seulement si x est strictement inférieur aux entiers de i
PROFIL	suit : $\mathbb{Z} \to intervalle \to \mathbb{B}$
SÉMANTIQUE	$(suit\ x\ i)$ si et seulement si x est strictement supérieur aux entiers de i

[git] • Branche master@e876d3f • Version v1.3.0-polyTP_partI.0 (14/01/2022) par François Puitg

Q1. Quels seraient les profils de ces trois fonctions si le type *intervalle* n'avait pas été défini ? Remarquer comment cette définition facilite la compréhension des profils des fonctions.

Q2. Compléter la spécification donnée ci-dessus par des exemples.

Q3. Réaliser les fonctions *precede*, *dans* et *suit* sans composition conditionnelle (l'utilisation de `if_then_else` est interdite).

2.6.2 Relations entre intervalles

Q4. Compléter la spécification de la fonction suivante :

SPÉCIFICATION 2 — relations sur les intervalles	
PROFIL	coteAcote : *intervalle* → *intervalle* → $\mathbb{B}$
SÉMANTIQUE	*coteAcote* (i_1, i_2) est vrai si et seulement si i_1 et i_2 sont contigus sans se toucher.
EX. ET PROP.	(i) (*coteAcote* $(1,3)$ $(4,10)$) = *vrai* (ii) (*coteAcote* $(2,8)$ $(-1,1)$) = *vrai* (iii) (*coteAcote* $(1,3)$ $(3,4)$) = ... (iv) (*coteAcote* $(2,8)$ $(3,4)$) = ... (v) cas où les intervalles ont une intersection vide : (*coteAcote* .....

Q5. Implémenter la fonction *coteAcote* sans composition conditionnelle (l'utilisation de `if_then_else` est interdite) ; donner des exemples d'utilisation pertinents.

SPÉCIFICATION 3 — relations sur les intervalles (suite)	
PROFIL	chevauche : *intervalle* → *intervalle* → $\mathbb{B}$
SÉMANTIQUE	(*chevauche* i_1 i_2) est vrai si et seulement si i_1 et i_2 n'ont pas de bornes communes, aucun n'est inclus dans l'autre mais ils ont une intersection non vide.

Q6. Implémenter la fonction *chevauche* sans composition conditionnelle (utilisation de `if_then_else` interdite) ; donner des exemples d'utilisation pertinents.

2.7 TD3 Le code de César

On désire coder un texte ne comportant que des lettres majuscules et des espaces, en remplaçant chaque caractère par un autre caractère selon les règles suivantes (code dit « de César ») :

- à un espace correspond un espace,
- à la lettre `'A'` correspond la lettre `'D'`, à `'B'` correspond `'E'`, ..., à `'W'` correspond `'Z'`, à `'X'` correspond `'A'`, à `'Y'` correspond `'B'`, à `'Z'` correspond `'C'`.

Inversement, on veut pouvoir décoder un texte codé selon ce qui précède. Pour cela on étudie une fonction de codage, *codeCar*, et sa fonction réciproque, *decodeCar*.

Q1. Spécifier (profil, sémantique, exemples et propriétés) les fonctions *codeCar* et *decodeCar*.

2.7.1 Version 1

Q2. Expliquer brièvement comment procéder à la main pour coder ou décoder un caractère.

Q3. En se limitant à un alphabet de 5 lettres $\{A, B, C, D, E\}$, implémenter chacune des fonctions *codeCar* et *décodeCar* par composition conditionnelle sous forme d'expressions conditionnelles imbriquées.

Q4. Proposer une autre implémentation en utilisant le filtrage

2.7.2 Version 2

Le code de César est fondé sur un décalage sur un alphabet considéré comme circulaire, c'est-à-dire que les lettres sont rangées dans l'ordre alphabétique et que la dernière lettre de l'alphabet est suivie par la première lettre :

`'A'` puis `'B'` puis `'C'` puis ...puis `'Z'` puis `'A'` ...

Le code d'une lettre est la lettre située 3 positions après elle dans l'ordre défini ci-dessus.

Idée *Assurer le codage en associant à chaque lettre un entier, par exemple son rang dans l'alphabet et utiliser des opérations sur les entiers pour exprimer le codage ou le décodage.*
Cette décomposition du problème est illustrée par le schéma suivant :

$$l \xrightarrow{\quad codeCar \quad} l'$$
$$l \xrightarrow{rangCar} r \xrightarrow{plus3} r' \xrightarrow{iemeCar} l'$$

- l dénote une lettre quelconque et l' dénote le code de l ; r désigne l'entier associé à l et r' celui associé à l'.
- Les flèches sont étiquetées par un nom de fonction. Elles montrent la manière de réaliser la fonction *codeCar* par composition fonctionnelle.

Les trois fonctions intermédiaires qui apparaissent sur le schéma sont spécifiées comme suit :

déf $nat1\_26 = \{1, \dots, 26\}$

SPÉCIFICATION 2	
PROFIL	$plus3 : nat1\_26 \to nat1\_26$
SÉMANTIQUE	$plus3\ (r)$ décale le rang r de 3 de manière circulaire dans $\{1, \dots, 26\}$
PROFIL	$rangCar : majuscule \to nat1\_26$
SÉMANTIQUE	$rangCar\ (l)$ est le rang dans l'alphabet de la lettre l donnée
PROFIL	$iemeCar : nat1\_26 \to majuscule$
SÉMANTIQUE	$iemeCar\ (r)$ est la lettre de rang r dans l'alphabet

Q5. Compléter la spécification par des exemples.

Q6. Implémenter le type *nat1_26* et la fonction *plus3*.

Q7. En utilisant le code ASCII de 'A' définit comme suit, implémenter *iemeCar* et *rangCar* :
`let cst_ASCII_A = int_of_char('A')`
où `int_of_char` est la fonction prédéfinie dans la librairie standard d'OCaml :

SPÉCIFICATION	
PROFIL	`int_of_char` : *char* → {0, … , 255}
SÉMANTIQUE	`(int_of_char c)` est le code ASCII de *c*.

Q8. Implémenter *codeCar*

Q9. Donner une réalisation de la fonction *decodeCar* en se basant sur un principe analogue à ce qui a été fait pour *codeCar*.

2.7.3 Généralisation

On généralise le principe du codage en se basant sur un décalage de d positions dans l'alphabet ($d = 3$ étant le cas particulier précédent). On notera qu'il est inutile d'envisager des décalages de plus de 25 ; on pourra à cette fin utiliser l'ensemble $\{1, \dots, 25\}$.

Q10. Spécifier les fonctions généralisées de codage *codeCarGen* et de décodage *decodeCarGen*, puis donner leur réalisation en utilisant les fonctions intermédiaires adéquates.

Q11. Réimplémenter *codeCar* en utilisant uniquement *codeCarGen*.

2.8 TD3 À propos de dates

Parmi les diverses formes d'expression de la date d'un jour dans le calendrier grégorien, nous nous intéressons à la suivante : une date est caractérisée par un triplet d'entiers composé du quantième du jour dans le mois, du quantième du mois dans l'année et de l'année. Ainsi, le triplet $(22, 9, 1990)$ caractérise le 22ème jour du 9ème mois de l'année 1990.

On ne considère ici que des dates prises dans la période $\{1900, \dots, 2100\}$. On rappelle qu'une année est dite *bissextile* si son expression numérale est divisible par 4, comme 1968, 1972, 1976. Toutefois les années séculaires[2] ne sont pas bissextiles, sauf celles dont les deux premiers chiffres forment un nombre divisible par 4, comme 1600, 2000, 2400.

Dans ce qui suit, on étudie diverses fonctions sur les dates.

2.8.1 Caractéristiques d'une date grégorienne

La période considérée est caractérisée par ses années de début et de fin. Une année est dénotée par un entier compris entre ces deux années.

- Le mois est dénoté par un entier compris entre 1 et 12.
- Un jour dans une année est désigné par un entier compris entre 1 et 365 ou 366 selon que l'année est bissextile ou non.
- Un jour dans le mois est représenté par un entier dans un intervalle qui dépend du mois considéré, mais qui est toujours inclu dans l'intervalle $\{1, \dots, 31\}$.

[2] *séculaire* signifie « de changement de siècle »

On définit les constantes et les types suivants :

déf $\text{AnDéb} = 1900,\ \text{AnFin} = 2100$
déf $\mathit{quantieme} = \{1, \ldots, 366\}$
déf $\mathit{jour} = \{1, \ldots, 31\},\ \mathit{mois} = \{1, \ldots, 12\}$
déf $\mathit{année} = \{\text{AnDéb}, \ldots, \text{AnFin}\}$
déf $\mathit{date} = \mathit{jour} \times \mathit{mois} \times \mathit{année}$

On dispose d'une fonction *ent_en_ch* spécificée ainsi :

SPÉCIFICATION 1	
PROFIL	`string_of_int` : $\mathbb{Z} \to \mathit{string}$
SÉMANTIQUE	`string_of_int`(n) est la chaine correspondant à l'entier n
EX. ET PROP.	(i) $(\mathit{string\_of\_int}\ 125) =$ `"125"` (ii) $(\mathit{string\_of\_int}\ 5) =$ `"5"` (iii) $(\mathit{string\_of\_int}\ 0) =$ `"0"`

Cette fonction est prédéfinie dans la librairie standard d'OCaml.

2.8.2 Quantième d'une date dans son année

On étudie une fonction *date_en_quant* :

SPÉCIFICATION 2	
PROFIL	$\mathit{date\_en\_quant} : \mathit{jour} \times \mathit{mois} \times \mathit{année} \to \mathit{quantieme}$
SÉMANTIQUE	$\mathit{date\_en\_quant}(j, m, a)$ est le numéro du jour dans l'année
EX. ET PROP.	(i) $\mathit{date\_en\_quant}(15, 11, 1993) = 319$

Pour déterminer le quantième d'une date dans son année, on commence par calculer le quantième du premier jour du mois donné dans le cas d'une année non bissextile. On définit pour cela la fonction :

SPÉCIFICATION 3	
PROFIL	$\mathit{mois\_en\_quant} : \mathit{mois} \to \mathit{quantieme}$
SÉMANTIQUE	$\mathit{mois\_en\_quant}(m)$ est le quantième du *premier jour* du mois m dans une année non bissextile.
EX. ET PROP.	(i) $\mathit{mois\_en\_quant}(11) = 304$

Q1.

a) Réaliser la fonction *date_en_quant* dans le cas d'une année non bissextile à l'aide de la fonction *mois_en_quant*.

b) Compléter la réalisation précédente pour tenir compte des années bissextiles. On définit pour cela le prédicat :

SPÉCIFICATION 4	
PROFIL	$est\_biss : année \to \mathbb{B}$
SÉMANTIQUE	(*est_biss* a) est vrai si et seulement si a est une année bissextile.

c) Réaliser la fonction *est_bis* en tenant compte de la période considérée $\{\text{AnDéb}, \ldots, \text{AnFin}\}$.

2.8.3 Affichage d'une date

On considère la fonction :

SPÉCIFICATION 5	
PROFIL	$date\_en\_ch : date \to string$
SÉMANTIQUE	$date\_en\_ch(d)$ est la chaine représentant la date d
EX. ET PROP.	(i) $date\_en\_ch(15, 11, 1993) =$ « 15 novembre 1993 » (ii) $date\_en\_ch(1, 11, 1993) =$ « 1er novembre 1993 »

Pour réaliser *date_en_ch* on introduit une fonction :

SPÉCIFICATION 6	
PROFIL	$mois\_en\_ch : mois \to string$
SÉMANTIQUE	$mois\_en\_ch(m)$ est le nom du mois m sous forme de chaine.

Q2. Réaliser la fonction *date_en_ch* en utilisant la fonction *mois_en_ch*.

2.8.4 Date du lendemain

Soit la fonction :

SPÉCIFICATION 7	
PROFIL	$lendemain\_en\_ch : date \to string$
SÉMANTIQUE	$lendemain\_en\_ch(d)$ est la chaine représentant le jour qui suit la date d ou bien « inconnue » si d est la dernière date de la période

Pour réaliser *lendemain_en_ch* on introduit une fonction :

SPÉCIFICATION 8	
PROFIL	*inc_date* : *date* → *date*
SÉMANTIQUE	*inc_date*(*d*) est la date du jour qui suit *d*

Pour déterminer la date du lendemain, il faut savoir si c'est la fin d'un mois et pour cela connaître le dernier jour de chaque mois :

SPÉCIFICATION 9	
PROFIL	*est_fin_mois* : *date* → $\mathbb{B}$
SÉMANTIQUE	*est_fin_mois*(*d*) vaut *vrai* si et seulement si la date (j, m, a) est le dernier jour du mois *m*

SPÉCIFICATION 10	
PROFIL	*dernier_jour* : *mois* → *jour*
SÉMANTIQUE	*dernier_jour*(*m*) est le numéro du dernier jour du mois *m* dans le cas d'une année non bissextile

Q3.

a) Réaliser dans cet ordre les fonctions *lendemain_en_ch*, *inc_date*, *est_fin_mois*.

b) Réaliser dans cette ordre les fonctions *mois_en_ch*, *dernier_jour* et *mois_en_quant*.

2.9 TD3 Nomenclature de pièces

On considère ici un problème d'informatisation d'un stock de pièces d'un magasin de fournitures pour automobiles, et plus particulièrement la manière de référencer ces pièces. Une nomenclature rassemble l'ensemble des conventions permettant de regrouper les pièces en différentes catégories et d'associer à chaque pièce une référence unique servant à la désigner. Ces conventions sont ici les suivantes :

- À un premier niveau, les pièces sont réparties selon la matière dans laquelle elles sont fabriquées :

  déf *matiere* = {ZINC, PLOMB, FER, CUIVRE, ALU, CARBONE, AUTRES}

- À un deuxième niveau, on différencie les pièces d'une même matière par un entier entre 0 et 999.

  déf *numero* = $\{0, \ldots, 999\}$

Une réference est définie par 5 caractères : les deux premiers correspondent à deux lettres de la matière de la pièce et les trois caractères suivants sont le numéro de la pièce (avec des zéros en tête si nécessaire). Exemples :

- le nom AL053 désigne la pièce de numéro 053 dans la famille des pièces en aluminium
- le nom CA053 désigne la pièce de numéro 053 dans la famille des pièces en carbone

[git] • Branche master@e876d3f • Version v1.3.0-polyTP_partI.0 (14/01/2022) par François Puitg

2.9.1 Types associés à la notion de référence

Pour représenter les références et leurs noms, on définit les ensembles *reference* et *nom* :

déf $reference = matiere \times numero$
déf $nom = majuscule \times majuscule \times chiffre \times chiffre \times chiffre$

Exemple : le couple (Alu, 53) est la valeur de type *reference* correspondant au nom `('A','L','0','5','3')`. Inversement, le nom `('C','A','0','5','3')` est représenté informatiquement par la référence (Carbone, 53).

Q1. Implémenter les types *matiere*, *numero*, *reference* et *nom*.

2.9.2 Fonctions associées à la notion de référence

On considère une fonction, nommée *nom_en_ref*, spécifiée comme suit :

SPÉCIFICATION 1 — conversion d'un nom en référence	
PROFIL	nom_en_ref : $nom \to reference$
SÉMANTIQUE	*nom_en_ref*(n) est la valeur de type *reference* associée au nom n.
EX. ET PROP.	(i) *nom_en_ref* `('C','A','0','5','3')` = (Carbone, 53) (ii) *nom_en_ref* `('A','L','0','5','3')` = (Alu, 53)

Pour réaliser la fonction *nom_en_ref*, on introduit les fonctions *chiffreVbase10* et *cc_en_mat* :
déf $base10 = \{0, ..., 9\}$

SPÉCIFICATION 2	
PROFIL	*chiffreVbase10* : $chiffre \to base10$
SÉMANTIQUE	*chiffreVbase10* (c) est l'entier associé au caractère décimal c.
EX. ET PROP.	(i) (*chiffreVbase10* `'2'`) = 2 (ii) (*chiffreVbase10* `'0'`) = 0
PROFIL	*cc_en_mat* : $caractère \times caractère \to matiere$
SÉMANTIQUE	*cc_en_mat* (c_1, c_2) est la valeur de type *matiere* dont le nom commence par le caractère c_1 suivi de c_2 et la valeur Autres si aucune matière ne correspond.
EX. ET PROP.	(i) *cc_en_mat* `('C','A')` = Carbone (ii) *cc_en_mat* `('C','U')` = Cuivre (iii) *cc_en_mat* `('X','P')` = Autres (iv) *cc_en_mat* `('B','Z')` = Autres

Q2. Donner la réalisation de la fonction *nom_en_ref* en utilisant *cc_en_mat* et *chiffreVbase10*.

Q3. Donner deux implémentations de la fonction *chiffreVbase10* :

RÉALISATION 2	
ALGORITHME	1) composition conditionnelle, sous forme d'expressions conditionnelles imbriquées.

IMPLÉMENT. .
.
.
ALGORITHME 2) composition conditionnelle, sous forme de filtrage
IMPLÉMENT. .
.
.

NB : vous avez vu (ou vous verrez) en TP une implémentation de la fonction *chiffreVbase10* qui n'utilise pas d'expression conditionnelle.

Q4. Implémenter la fonction *cc_en_mat*.

2.9.3 Comparaison de références : relation d'ordre sur les types

Les opérateurs OCaml de comparaison (=, <>, <, <=, >, >=) sont polymorphes et s'appliquent à tous les types OCaml de base (dont les booléens, les caractères et les nombres), ainsi qu'aux types construits par composition.

Pour un type somme, l'ordre OCaml entre deux valeurs est défini implicitement par l'ordre entre les constructeurs du type (et si les valeurs partagent le même constructeur par l'ordre sur l'eventuel paramètre du constructeur). Dans la définition d'un type somme, les constructeurs sont énumérés par ordre croissant de valeur.

L'ordre implicite entre deux n-uplets d'un même type produit est l'ordre de leurs premiers éléments distincts de même rang (en partant de la gauche) : le premier élément d'un type produit est donc prépondérant dans une comparaison.

Q1. L'ordre implicite sur les noms correspond-t-il à l'ordre implicite sur les références ?

On définit *ordrebis*, un nouvel ordre sur les références, dans lequel la valeur relative des numéros prime sur la matière.

Q2. Implémenter la fonction suivante :

SPÉCIFICATION 3	
PROFIL	inf_ref : *reference* → *reference* → $\mathbb{B}$
SÉMANTIQUE	(*inf_ref* r_1 r_2) vaut *vrai* si et seulement si r_1 est avant r_2 suivant *ordrebis*.
EX. ET PROP.	(i) (*inf_ref* (*Carbone*, 13) (*Zinc*, 45)) = *vrai* (ii) (*inf_ref* (*PLOMB*, 12) (*ALU*, 943)) = *vrai*

[git] • Branche master@e876d3f • Version v1.3.0-polyTP_partI.0 (14/01/2022) par François Puitg

Chapitre 3

Annales

Sommaire

3.1 Partiel 2014–2015 TD3 «un dîner presque parfait»

Le but de l'exercice est de calculer le prix d'un repas, sachant que :

- un repas est constitué d'une entrée, plat principal, fromage et/ou dessert ;
- il y a trois tailles pour l'entrée ;
- le fromage se prend à l'unité ou au plateau ;
- certains plats nécessitent un supplément.

3.1.1 Modélisation

On definit les types OCaml suivants :

```
(* Trois plats principaux possibles : *)
type platPrincipal = P1 | P2 | P3

(* Une seule entrée possible, mais en trois tailles : *)
type tailleEntree = Petite | Moyenne | Grande

(* x morceaux de fromage à l'unité : M(x), ou au plateau : P *)
type fromage = M of int | P
```

```
(* Trois desserts possibles : *)
type dessert = D1 | D2 | D3
```

On supposera que les coûts des différents éléments du repas sont toujours des entiers naturels ($\mathbb{N}$).

Q1. (0,5pt) Implémenter en OCaml un type appelé `cout` représentant une somme entière d'argent. On prendra soin de préciser toute contrainte éventuelle sur les éléments de ce type.

3.1.2 Coûts des éléments du repas

Une petite entrée coûte 3 €, une moyenne 5 et une grande 7.

Q2. (1pt) Implémenter une fonction *coutEntree* qui renvoie le coût d'une entrée en fonction de sa taille.

Les plats principaux sont à 10 €, avec un *supplément* de 3 € pour `P3`. On définit donc les constantes suivantes :

```
let cstPRIXPRINCIPAL: cout = 10
let cstSUPLP3: cout = 3
```

Q3. (2pt) Spécifier (profil, sémantique, exemples ou propriétés) puis réaliser une fonction *coutPrincipal* qui renvoie le coût d'un plat principal.

Un morceau de fromage à l'unité coûte 2 € tandis que le plateau coûte 6 €.

Q4. (1,5pt)

a) Définir des constantes représentant ces différents coûts, et implémenter une fonction *coutFromage* qui renvoie le coût du fromage.

b) Donner une expression OCaml permettant de vérifier qu'il vaut mieux, d'un point de vue financier, prendre le plateau que quatre fromages.

Q5. (0,5pt) Proposer une implémentation en OCaml de la règle : « les desserts sont tous à 5 € » sous forme d'une fonction.

3.1.3 À table

On suppose qu'il existe deux formules pour les repas :

$Formule_1$: une entrée et un plat,

$Formule_2$: un plat et un dessert ou un fromage.

On donne la définition mathématique de l'ensemble « fromage ou dessert » :

$$fromOUdess \stackrel{def}{=} \{D(d) \ / \ d \in dessert\} \cup \{F(f) \ / \ f \in fromage\}$$

Q6. (2pt)

a) Quel est la nature de D et F ?

b) Implémenter $fromOUdess$ en OCaml

c) Quel est le profil (*ensemble de départ* → *ensemble d'arrivée*[1]) de `D` ? De `F` ?

Un repas est implémenté en OCAML de la manière suivante :

```
type repas =                                          1
  | F1 of tailleEntree * platPrincipal                2
  | F2 of platPrincipal * fromOUdess                  3
```

Q7. (0,5pt) Donner les définitions en OCAML des constantes correspondant aux repas suivants :

a) une formule 1 composée d'une petite entrée et du plat principal 3,

b) une formule 2 composée du plat principal 1 et d'un morceau de fromage.

Q8. (0,5pt) Quel est le coût des repas de la question précédente ?

Q9. (1,5pt) Implémenter une fonction *coutRepas* qui calcule le coût d'un repas.

Étendons maintenant la notion de repas :

$Formule_1$: une entrée et un plat,

$Formule_2$: un plat et un dessert ou un fromage,

$Formule_3$: une entrée, un plat, un fromage *et*/*ou* un dessert.

Q10. (2pt) Modéliser la notion « fromage et/ou dessert » par un type appelé `fromETOUdess`, puis implémenter le nouveau type `repas2`.

Q11. (2pt) Donner les modifications à apporter à la fonction *coutRepas* précédente pour calculer le coût d'un `repas2`.

3.2 Partiel 2013–2014 TD3 colorimétrie

En informatique, le terme *colorimétrie* regroupe les différents systèmes de gestion des couleurs des périphériques (écran, projecteur, imprimante, scanner, appareil photo, vidéo, ...).

Le but de cet exercice est d'implémenter le format de codage *RVB*, cet acronyme désignant les trois couleurs primaires[2] : rouge, vert et bleu.

3.2.1 Les couleurs

Dans le système colorimétrique RVB, on considère qu'une couleur est un mélange des trois couleurs – rouge, vert et bleu – selon une certaine intensité.

L'*intensité* est un entier naturel compris entre 0 (intensité minimale : couleur correspondante absente) et 255 (intensité maximale de la couleur correspondante).

On définit donc une couleur comme un triplet (r, v, b), où la composante r représente l'intensité du rouge, v l'intensité du vert et b celle du bleu.

Q1. (1pt) Compléter l'implémentation du type `tripletRVB` :

```
type intensite = int          (* restreint à {0, ..., 255} *)
type tripletRVB = ...............................................   2
```

[1] ou, plus précisément, *domaine* → *codomaine*

[2] en synthèse additive

Les couleurs primaires ont une intensité maximale pour leur propre composante, minimale pour les autres ; le noir est défini [1] comme l'absence de couleur ; le blanc est défini [1] comme le mélange à intensité maximale des trois couleurs primaires.

Q2. (1,25pt) Implémenter ces cinq couleurs grâce à la construction OCaml :

```
let ... : tripletRVB = ...
```

3.2.2 Le gris

Un triplet RVB est dit *gris* quand l'intensité de ses trois composantes est la même.

Q3. (1,5pt) Définir (spécification + réalisation) une fonction appelée *estGris* qui teste si un triplet RVB donné quelconque est gris. Dans la section « exemple » de la spécification, on veillera à donner un exemple retournant vrai et un exemple retournant faux.

Le *niveau de noir* d'un triplet RVB gris est le pourcentage de noir présent dans ce triplet. Par exemple, le niveau de noir du blanc est 0, tandis que le niveau de noir du noir est 100.

Q4. (1,25pt) Définir (spécification + réalisation) une fonction appelée *niveauNoir* qui donne le niveau de noir d'un triplet RVB (gris) quelconque, sous forme d'un réel entre 0 et 100.

3.2.3 Barbouillons[3]

Afin de pouvoir mélanger les couleurs, on spécifie la fonction suivante :

SPÉCIFICATION 2	
PROFIL	barbouille : $tripletRVB \rightarrow tripletRVB \rightarrow tripletRVB$
SÉMANTIQUE	(barbouille t_1 t_2) est la couleur obtenue en mélangeant t_1 et t_2
EX. ET PROP.	(i) soit jaune $= (255, 255, 0)$, (barbouille rouge vert) = (barbouille vert rouge) = jaune , (ii) (barbouille bleu jaune) = (barbouille jaune bleu) = blanc

Q5. (1,5pt) En se basant sur l'algorithme suivant, donner l'implémentation de cette fonction.

ALGORITHME Les intensités sont additionnées composante par composante. Si le résultat de cette addition dépasse 255, l'intensité correspondante est fixée à 255.

3.2.4 Du gris et des couleurs

On souhaite maintenant manipuler divers modèles de gestion de couleurs, en se basant sur la définition suivante :

DÉFINITION D'UN ENSEMBLE

$$couleur \stackrel{def}{=} \{Rouge, Vert, Bleu\} \bigcup \{Noir(p) \ / \ p \in [0, 100]\} \bigcup \{Rvb(t) \ / \ t \in tripletRVB\}$$

Dans cette définition, p est un réel compris entre 0 et 100.

Q6. (1,75pt)

a) Quelle est la nature de *Rouge*, *Vert*, *Bleu*, *Noir* et *Rvb* dans la définition ci-dessus ?

b) Donner une implémentation en OCaml de l'ensemble *couleur*.

[3] de barbouille, n.f. fam. : peinture de qualité médiocre

Couleurs primaires

Les couleurs *primaires* sont le rouge pur, le vert pur, le bleu pur, et ce sont les seules.

Q7. (1,75pt) Implémenter en OCaml une fonction appelée *estPrimaire* qui teste si une couleur quelconque est primaire.

Couleurs complémentaires

Le *complément à x* d'un nombre est la quantité qui lui manque pour « aller à » x. Par exemple, le complément à 10 de 7 est 3, car $7 + 3 = 10$.

Le *complémentaire* d'une couleur est défini par les règles suivantes :

- le complémentaire du rouge est le cyan (intensités RVB = 0, 255, 255), le complémentaire du vert est le magenta (intensités = 255, 0, 255), le complémentaire du bleu est le jaune (255, 255, 0) ;
- le complémentaire d'un noir de niveau n est le noir dont le niveau est le complément à 100 de n ;
- le complémentaire d'une couleur dont les intensités RVB sont r, v et b est la couleur dont les intensités sont les compléments à 255 de r, v et b.

Q8. (2pt) Implémenter en OCaml une fonction appelée *complementaire* qui retourne la couleur complémentaire d'une couleur donnée quelconque.

Barbouillons encore

On veut à nouveau mélanger deux couleurs, en réutilisant les objets déjà définis. Pour celà, on spécifie une fonction de conversion des couleurs vers les triplets RVB :

PROFIL couleurVtriplet : *couleur* → *tripletRVB*

SÉMANTIQUE couleurVtriplet(c) est le triplet RVB correspondant à c.

Q9. (1,5pt) Implémenter *couleurVtriplet* en OCaml.

Q10. (1,5pt) En déduire une implémentation de la fonction de mélange des couleurs, spécifiée ainsi :

PROFIL barbouilleC : *couleur* → *couleur* → *couleur*

SÉMANTIQUE (barbouilleC c_1 c_2) est la couleur obtenue en mélangeant c_1 et c_2

Deuxième partie

DÉFINITIONS RÉCURSIVES

Troisième partie

ORDRE SUPÉRIEUR

[git] • Branche master@e876d3f • Version v1.3.0-polyTP_partI.0 (14/01/2022) par François Puitg

Quatrième partie

STRUCTURES ARBORESCENTES