

## TD7 - Termes et Equivalence

### Exercice 1 *Termes, examen 2 2012.*

On cherche à modéliser un morceau de musique simple pour le chant. Une partition est une suite de *notes* de musique et de *silences*. Une note de musique est caractérisée par une *durée* et une *hauteur*. Un silence est caractérisé par une durée.

On choisit de représenter la hauteur d'une note par un entier de 1 à 88. Chaque entier correspond à une touche sur un clavier de piano, numérotées de la note la plus grave (numéro 1) à la note la plus aigüe (numéro 88).



La *durée* représente un nombre de *temps* pendant lequel il faut chanter (ou au contraire se taire pour un silence). Par exemple une croche représente un demi-temps, une note noire représente un temps, une note blanche représente deux temps. ... Cette durée sera représentée par un nombre rationnel ( $d \in \mathbb{Q}$ ).

La signature utilisée pour des termes représentant un **chant** se compose de trois symboles :

- **note** :  $\mathbb{Q} \times \mathbb{N} \rightarrow \text{chant}$
- **silence** :  $\mathbb{Q} \rightarrow \text{chant}$
- **seq** :  $\text{chant} \times \text{chant} \rightarrow \text{chant}$

Le terme **note**( $d, h$ ) représente la note de durée  $d$  et de hauteur  $h$  ; **silence**( $d$ ) représente un silence de durée  $d$  ; **seq**( $c_1, c_2$ ) représente un morceau où on commence par exécuter les notes et silences de  $c_1$  puis ceux de  $c_2$ . L'exemple ci-dessous illustre la représentation d'un morceau.



```
seq(seq(silence(0.5),note(0.5,40)),  
seq(note(1,42),seq(note(0.75,44),note(0.25,45))))
```

1. Définir de façon récursive sur les termes qui représentent un chant une fonction *nb\_temps* qui calcule un entier représentant la durée totale du champs (le nombre de temps).
2. Définir de façon récursive une fonction *le* qui étant donnés une hauteur de note  $h$  et un chant  $c$ , renvoie un booléen qui est vrai si la note  $h$  est plus basse que toutes les notes du chant  $c$ .
3. Définir de façon récursive une fonction *in* qui étant donnés une hauteur de note  $h$  et un chant  $c$ , renvoie un booléen qui est vrai si une note de hauteur  $h$  apparaît dans le chant  $c$ .
4. Donner le principe d'induction sur les termes représentant un chant.
5. Montrer en utilisant ce principe que pour tout chant  $c$ , si  $le(h_1, c)$  et  $in(h_2, c)$  alors  $h_1 \leq h_2$ .
6. On cherche à caractériser un chant dans lequel les notes vont en montant. C'est à dire que si une note  $n$  est chantée avant la note  $m$  alors la hauteur de  $n$  est inférieure ou égale à la hauteur de  $m$ . On note **montant**( $c$ ) cette propriété. Donner des règles d'inférence pour définir **montant**( $c$ ).

#### Indications

- On remarquera que une note seule ou un silence vérifient la propriété **montant** et que pour que les notes de **seq**( $c_1, c_2$ ) aillent en montant, il faut et il suffit qu'il existe une hauteur  $h$  telle que les notes de  $c_1$  soient plus basses que  $h$  et les notes de  $c_2$  soient plus hautes que  $h$  et que  $c_1$  et  $c_2$  soient eux-mêmes des chant qui aillent en montant.

- On pourra utiliser sans la redéfinir une fonction  $ge$  analogue à  $le$  qui étant donné une hauteur  $h$  et un chant  $c$  renvoie vrai si la note  $h$  est plus haute que toutes les notes du chant  $c$ .

**Exercice 2** *Classes d'équivalence.* On se place sur l'ensemble  $\mathbb{Z}$  des entiers relatifs.

Soit la relation  $R(x, y) \stackrel{\text{def}}{=} x - y$  est pair.

1. Montrer que  $R$  est une relation d'équivalence.
2. Donner les classes d'équivalence de cette relation.

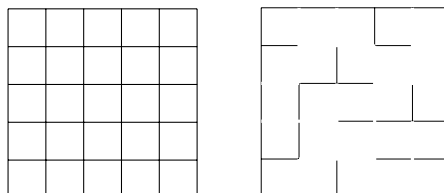
**Exercice 3** *Partitions*

1. Soit  $G$  un graphe non-orienté, montrer que la relation  $R(x, y)$  qui est vraie s'il existe un chemin de  $x$  à  $y$  est une relation d'équivalence. Quel est le nom donné aux classes d'équivalence de cette relation ?
2. La structure union-find introduite en cours permet de représenter des partitions finies. Soit  $A$  un ensemble fini d'éléments de taille  $n$ , on notera  $U$  l'ensemble des structures union-find qui représentent les partitions de  $A$ . On ne dit pas comment ces partitions sont représentées mais on se donne les objets suivants pour travailler sur cette structure.
  - $init \in U$  représente la partition initiale formée des  $n$  singletons d'éléments de  $A$
  - $taille \in U \rightarrow \mathbb{N}$  renvoie le nombre de paquets dans la partition
  - $repr \in A \times U \rightarrow A$ ,  $repr(x, u)$  renvoie un représentant canonique du paquet dans lequel se trouve  $x$ . C'est-à-dire que deux éléments seront dans le même paquet s'ils ont le même représentant.
  - $union \in A \times A \times U \rightarrow U$ ,  $union(x, y, u)$  renvoie une nouvelle partition dans laquelle les paquets de  $x$  et de  $y$  ont été fusionnés (s'ils étaient différents, sinon la partition est inchangée).

Exprimer comme des formules logiques la spécification des objets introduits ci-dessus, en particulier comment peut-on caractériser les objets suivants

- $taille(init), repr(x, init)$
- $taille(union(x, y, u)), repr(z, union(x, y, u))$

3. On cherche à construire un labyrinthe "aléatoire". Pour cela on va considérer une grille formée de  $n \times n$  cases. On va retirer des murs de telle manière qu'il y ait à la fin toujours un chemin possible entre deux cases quelconques du labyrinthe.



On suppose que l'on note  $A$  l'ensemble des cases de la grille et que l'on a une fonction aléatoire  $choix\text{-mur}$  qui nous renvoie au hasard un des murs interne de la grille. Un mur est représenté par le couple  $(x, y)$  des deux cases qu'il sépare.

En utilisant la structure union-find introduite précédemment ainsi que la fonction  $choix\text{-mur}$ , proposer (en pseudo-code) un algorithme pour construire un labyrinthe. Le résultat de la fonction pourra être la liste des murs à retirer de la grille.

Justifier pourquoi votre algorithme est correct.