Flux et Systèmes Dynamiques

Objectifs

— Utilisation des flux.
— Simulation de systèmes dynamiques.

On utilisera l'archive `sourceEtu.tar` fournie, incluant le fichier source `tp7_etu.ml` qui contient des éléments permettant la réalisation des exercices de la séance.

1 Simulation de systèmes dynamiques

On s'intéresse à la simulation de systèmes dynamiques, comme vu en TD. Le module Flux ainsi que d'autres fonctions utilitaires sont fournis. On cherche à représenter par des flux le système dynamique constitué d'une balle rebondissante dans un espace 2D plat, soumise au comportement suivant :

1. la balle est soumise à l'accélération constante de la pesanteur ;
2. la balle rebondit sur des murs formant une boîte rectangulaire.

On utilisera le type suivant (fourni) pour représenter l'état d'un système, i.e. la paire ((x, y), (dx, dy)) où (x, y) est la position et (dx, dy) la vitesse :

```
type etat = (float * float) * (float * float)
```

L'interface Frame fournie permet de spécifier certains paramètres de la simulation.

▷ **Exercice 1 (Modèle simplifié)**
Compléter le module FreeFall*, contenant la fonction* run : etat -> etat Flux.t *qui définit le flux des états successifs d'une balle en chute libre, pris à chaque pas de temps, en fonction de l'état initial. On définira pour cela des flux auxiliaires représentant la position, la vitesse et l'accélération d'un modèle de balle, en ne prenant en compte que la condition* (1) *de chute libre. On rappelle que l'accélération est alors donnée par le vecteur constant* $(0, -g)$*. On pourra utiliser la définition de l'intégrateur* integre *fournie.*

▷ **Exercice 2 (Simulation)**
Obtenir une simulation de votre modèle de balle à l'aide de la fonction Drawing.draw *fournie, permettant d'afficher position et vitesse ainsi qu'un rendu graphique en temps-réel de la balle. On définira pour cela un module* F : Frame *avec lequel on pourra créer des instances des modules* FreeFall *et* Drawing *adaptées aux paramètres de simulation choisis dans* F*, et enfin on exécutera* draw (run etat0) *pour un état initial* etat0 : etat *de votre choix.*

On va maintenant prendre en compte la condition (2) de rebonds sur les murs. La détection de collision dépend de la position et de la vitesse. La collision induit un *changement de mode*, i.e. le système jusque-là décrit par un certain flux sera décrit, après la collision, par un autre flux construit à partir de l'état où le premier flux a été interrompu.

▷ Exercice 3 (Changement de mode)

Définir unless : 'a Flux.t -> ('a -> bool) -> ('a -> 'a Flux.t) -> 'a Flux.t, *où* unless flux cond f_flux *est constitué des valeurs du flux* flux *jusqu'à ce qu'éventuellement, la condition* cond *soit vraie de la valeur courante (*v*), auquel cas la suite du flux résultat est* (f v)*. Par exemple, pour un flux* a*, en supposant que* k *est le premier indice tel que* cond a_k = true *et que* f a_k = cons b_0 (cons b_1 (...))*, alors :*

unless (cons a_0 (...(cons a_k (...)))) cond f = (cons a_0 (...(cons a_{k-1} (cons b_0 (cons b_1 (...)))))))

▷ Exercice 4 (Détection de collision)

1. *Définir* contact_x : float -> float -> bool, *où* contact_x x dx *détecte une collision si et seulement si la position* x *est en dehors des bornes* F.box_x *et* dx *est dirigée vers l'extérieur.*
2. *Définir* rebond_x : float -> float -> float, *où* rebond_x x dx *inverse la vitesse* dx *si et seulement si un contact en* x *est détecté.*
3. *Définir de même* contact_y *et* rebond_y*.*

▷ Exercice 5 (Modèle complet)

Définir le module Bouncing*, contenant la fonction* run : etat -> etat Flux.t *qui calcule le flux des états d'une balle à partir d'un état initial, en prenant en compte les conditions* (1) *et* (2)*. Une nouvelle simulation devra être démarrée par la même fonction* run *à chaque fois qu'une collision est détectée, avec un nouvel état "initial" où la vitesse est modifiée.*

▷ Exercice 6 (Simulation complète)

Obtenir une simulation de votre modèle complet de balle.

En laissant tourner la simulation, arrivera un moment où celle-ci ralentira significativement, cela est dû à l'intervention du *garbage collector*, qui doit consacrer une partie du temps processeur à récupérer les éléments des flux construits qui ne sont plus utiles (toutes les positions passées par exemple). D'autres implantations des flux avec une empreinte mémoire plus légère sont possibles.