

11 Les streams et les séquences infinies

11.1 Exercices simples

Pour tous les exercices dans cette section, utilisez une structure de données abstraite pour les streams. L'interface de cette structure est fournie ci-dessous:

```
(define head car)

(define (tail stream)
  (force (cdr stream)))

(define-macro cons-stream
  ;; delay evaluation until needed
  (lambda (a b)
    `(cons ,a (delay ,b))))
```

Remarque. Utilisez le langage Pretty Big (Assez Gros Scheme).

Exercice 11.1 Implémentez les fonctions `(nth-stream n s)` et `(nth-value n s)` pour calculer le n -ième stream et la n -ième valeur dans le stream s . Par exemple,

```
>(define (allIntegersFrom n)
  (cons-stream n (allIntegersFrom (+ 1 n))))
> (nth-stream 5 (allIntegersFrom 10))
(15 . #<struct:promise>)
> (nth-value 5 (allIntegersFrom 10))
15
```

Exercice 11.2 Implémentez la fonction `(print-stream s)` pour afficher toutes les valeurs d'un stream fini. Implémentez la fonction `(print-stream-to s n)` pour afficher les n premières valeurs d'un stream (fini ou infini). Par exemple

```
> (print-stream-to 20 (allIntegersFrom 10))
[10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29]
```

Exercice 11.3 Implémentez les fonctions `accumulate`, `map`, `filter` et `for-each` pour les streams.

Conseil. La définition de ces fonctions pour les streams est presque la même que pour les listes.

Exercice 11.4 Implémentez un stream infini qui représente une séquence d'approximations de $e = 2.71828182845904590\dots$ en utilisant la formule de Taylor:

$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \dots$$

Par exemple,

```
> (print-stream-to 5 estream)
[1.0 2.0 2.5 2.6666666666666665 2.7083333333333335]
```

Exercice 11.5 Utilisez `filter` pour obtenir un stream infini qui contient tous les nombres entiers qui ne sont pas divisibles par 2, 3 ou 5.

11.2 Combining streams

Pour tous les exercices dans cette section, vous pouvez utiliser les fonctions suivantes:

```
(define (combine-streams combiner s1 s2)
  (cons-stream (combiner (head s1) (head s2))
               (combine-streams combiner (tail s1) (tail s2))))
```

```
(define (add-streams s1 s2)
  (combine-streams + s1 s2))
```

```
(define (multiply-streams s1 s2)
  (combine-streams * s1 s2))
```

```
(define (divide-streams s1 s2)
  (combine-streams / s1 s2))
```

Exercice 11.6 Prédisez le résultat de l'affichage des expressions suivantes:

1.

```
(define ones (cons-stream 1 ones))
(print-stream-to 10 ones)
```
2.

```
(print-stream-to 10
  (add-streams (tail allIntegers) allIntegers))
```
3.

```
(define (addrec stream)
  (cons-stream (head stream)
               (addrec (add-streams (tail stream) stream))))
(print-stream-to 10 (addrec ones))
```
4.

```
(define (addrec2 stream)
  (cons-stream (head stream)
               (addrec2 (add-streams stream stream))))
(print-stream-to 10 (addrec2 allIntegers))
```

Exercice 11.7 Utilisez la fonction `add-streams` pour la création d'une séquence infinie de nombres entiers, où chaque nombre à la position n doit être égal à la somme des n premiers nombres entiers.

À titre d'exemple, voici les 10 premières valeurs de cette séquence infinie:

```
[0 1 3 6 10 15 21 28 36 45 ...]
```

Exercice 11.8 Utilisez la fonction `multiply-streams` pour la création d'une séquence infinie de nombres entiers, où chaque nombre à la position n doit être égal au factoriel de n .

À titre d'exemple, voici les 10 premières valeurs de cette séquence infinie:

```
[1 1 2 6 24 120 720 5040 40320 362880 ...]
```

Exercice 11.9 Refaites l'exercice 12.4 pour créer un stream infini qui représente une séquence d'approximations de e . Créez ce stream à partir du résultat de l'exercice précédent, en utilisant une combinaison des opérateurs `add-streams` et `divide-streams`.

12 Programmation orientée objet

Exercice 12.1 *Implémentez une classe orientée objet en Scheme comme expliqué pendant le cours. La classe doit créer des objets qui se comportent comme montré ci-dessous:*

```
>(define number1 (new-number 5))
> (send number1 ' (positive))
#t
> (send number1 ' (decreasewith 10))
-5
> (send number1 ' (show))
-5
> (send number1 ' (negative))
#t
> (send number1 ' (increasewith 5))
0
> (send number1 ' (negative))
#f
```