

# Récurtivité



04 mars 2019

Damien Morard

Au cours de cette séance d'exercices, vous écrirez des règles récursives.

## Exercice 1 : Factorielle



Commençons par l'exemple le plus simple pour la récursivité : le calcul de la factorielle. Remplissez la règle **factorielle** permettant de calculer :

$$X! = \begin{cases} X * (X - 1)! & \text{if } X > 0 \\ 1 & \text{if } X = 0 \end{cases}$$

Cette règle est évaluée à vrai si et seulement si la variable **Resultat** vaut  $X!$ .

Avant de commencer les exercices, créez un projet swift où vous aurez bien mis le lien du projet LogicKit dans les dépendances du fichier **Package.swift**. Vous pouvez directement cliquer [ici](#) pour revoir exactement toutes les étapes.

Deuxième point très important, dans le fichier **Source/main.swift** où se trouvera votre code, vous penserez à importer :

```
import LogicKit
import LogicKitBuiltins
```

Dernière information essentielle, nous allons utiliser les types déjà intégrés dans swift pour manipuler les **Nat** et les **List**. Pensez bien à regarder le [README](#) où vous aurez une table pour chaque type avec les opérations les concernant. Vous pourrez trouver des exemples sur le code source de la première séance d'exercice [ici](#).

```
let n: Term = Nat.from(3)
let l: Term = List.from(elements:
  [1,2,3].map(Nat.from))
```

1. Écrivez la règle **factorielle**.

```
.rule("factorielle", x, resultat) ...
```

2. Que se passe-t-il si vous cherchez à obtenir **x** ?

```
let res = kb.ask(.fact("factorielle", x, 6)).
for binding in res {
  print(res["x"])
}
```

Est-ce qu'il y a un problème ? Si oui, pourquoi ?

Testez ce nouveau code :

```
let res = kb.ask(.fact("factorielle", x, 6)).
for binding in res.prefix(1) {
  print(res["x"])
}
```

## Exercice 2 : Somme



Nous souhaitons maintenant faire la somme des entiers contenus dans une liste. Pour cela, construisez tout d'abord la liste au moyen de la règle **liste**, puis calculez la somme des entiers qu'elle contient au moyen de **somme**.

1. Écrivez la règle (récursive) suivante, qui crée la liste **res** contenant les entiers de **premier** à **dernier** inclus.

```
.rule("enum", premier, dernier, res) ...
```

Une fois que celle-ci fonctionnera, vous allez tenter d'utiliser le logger de LogicKit ! C'est un outil pratique qui vous sera sûrement très utile pour debug et comprendre où vous vous êtes trompé. Attention cependant à le faire sur les exemples les plus simples possibles, la liste des opérations peut très vite devenir compliquée !

```
let enum = kb.ask(.fact("enum", Nat.from(0), Nat.from(2),
  res), logger: DefaultLogger(useFontAttributes: false))
```

Vous n'avez pas besoin de faire un **print** pour que la liste des exécutions s'affichent.

2. Écrivez la règle (récursive) suivante :

```
.rule("somme", liste, res) ...
```

3. Pouvez-vous obtenir toutes les listes permettant d'obtenir un résultat ? Si oui, comment ? Si non, pourquoi ?

### Exercise 3 : Palindrome



Écrivez la règle `palindrome` prenant en paramètre une liste de caractères. Cette règle n'est évaluée à vrai que si cette liste forme un palindrome, c'est-à-dire qu'elle peut être lue de gauche à droite comme de droite à gauche, en formant le même mot.

```
.rule("palindrome", mot) :- ...
```

1. Pour cela, écrivez une règle `renverser` qui renverse l'ordre des éléments d'une liste. Vous pourrez vous aider de la règle `concat` qui existe déjà sous LogicKit.

```
.rule("renverser", l1, l2) ...
```

La fonction `concat` en LogicKit permet de concaténer deux listes. Les deux arguments **DOIVENT** être des listes de termes, et non un simple terme. N'oubliez pas non plus d'importer la base de connaissance des listes! (L'opérateur `+` est utilisé pour la fusion de deux bases de connaissances).

```
// Two ways to write a list of terms
let list1: Term = List.from([1,2].map(Nat.from))
let list2: Term = List.cons(Nat.from(4), List.empty)
let res: Term = .var("res")
let query = kb.ask(List.concat(list1, list2, res))

for binding in query {
  print(binding["res"])
}
```

2. Écrivez maintenant la règle `palindrome`.
3. Pouvez-vous obtenir tous les palindromes possibles? Si oui, comment? Si non, pourquoi?