/*

```
*/
import cloud.greentea.{*,given}
import cats.effect.std.Console
import cats.effect._
import cats.{~> as -->, arrow as ar, *}
/*
```

API

@author Sylvain Ductor, 2022

Déclaration

Une norme implémente une catégorie monoidale. Elle peut être étendue en une catégorie de Ductor afin d'intégrer des effets de bords. Pour cela elle considère:

- Un environnement `ε`, accessible en lecture et en écriture et cela en fonction du couple d'agents considéré,
- Des (pré)sources (ou axiomes), `+[A]` lisant cet environnement et retournant un type `A`
- Des (pré)puits (ou preuves), `-[B]` écrivant un objet de type `B` dans l'environnement.

A titre illustratif, supposons que nous souhaitons une interface de communication avec le terminal. Nous choisissons d'utiliser pour cela l'objet `Console[IO]` de la librairie `cats.effect` sur lequel se fonde l'implémentation de `GreenTea` (voir https://typelevel.org/cats-effect/api/3.x/cats/effect/std/Console.html)

Source

Nous commençons par la source, c'est à dire la flèche qui va lire et retourner l'entrée de l'utilisateur. Nous souhaitons une invite de commande, puis, que celle-ci retourne la commande entrée.

```
*/
val console_reader = ((ε : Console[IO]) =>
    ε.println("requesting :") >> ε.readLine
).+[String]
/*
```

A l'instar d'une prénorme, nous étiquetons une fonction Scala avec `+[String]` pour indiquer qu'il s'agira d'une source de `String`. Cette fonction est de type `[Console[IO] => IO[String]]` et va être généralisée en une source de type `(∅ ~> String)` lors de l'intégration d'un environnement qui fournira `Console[IO]` en lecture. `∅` représente le type «zero».

Le symbole `>>` provient de l'encapsulateur de classe `Applicative` (voir https://adit.io/posts/2013-04-17-functors,_applicatives,_and_monads_in_pictures.html) contractuellement implémenté par `IO`. Il séquence les deux calculs en "oubliant" le premier. Sémantiquement, il est similaire à un `;` , seulement, s'il avait été un `;` l'effet de bord `println` n'aurait pas été effectué du fait du fonctionnement de `scala.cats.effect.IO`.

Puit

Le puit, quant-à lui, considère un objet à écrire et met à jour l'environnement.

```
*/
val console_printer = ((b : String, ε : Console[IO]) =>
        ε.println(b).map(_ => ε)
).-[String]
/*
```

La fonction Scala est cette fois-ci du type `[(String,Console[IO]) => IO[Console[IO]]]` et va être généralisée en un puit de type `(String ~> ∅)`

API

Une api est un couple (source,puit) portant sur le même environnement.

```
*/
val console_api = (console_reader,console_printer).api
/*
```

Usage

Considérons une prénorme, `sayHello`, demandant le nom de son interlocuteur et lui disant bonjour.

```
*/
val sayHello = ( (name : String) => s"Hello $name" ).~>[String,String]
/*
```

Nous introduisons ici une deuxième méthode pour déclarer un interpréteur : en faisant un objet l'étendre. Au sein de l'objet (et seulement en son sein), non seulement l'interpréteur est chargé, mais de plus certains types et routines sont accessibles (par exemple Effect). Cette approche est la plus pratique hors d'une console sbt. A l'instar de python, maintenir la tabulation indique à Scala que l'on est toujours dans l'environnement objet.

```
*/
object tuto_api extends interpretor.default :

    println(sayHello.yes.execWith("World").getResult())
/*
```

Tant une présource qu'un prépuit sont des alias de prénormes et se comporte donc similairement.

```
*/
    val entry : Maybe[String] = //devrait être Often
        console_reader.yes.exec.getResult()

    console_printer.yes.execWith(s"Il a écrit : $entry").run()
/*
```

Une norme peut charger une api en lecture écriture ou les deux. Son type est alors modifié de façon cohérente ?. Elle devient une source si elle charge l'api en lecture, un puit si elle la charge en écriture ou un effet (`( ∅ ~> ∅ )`) si elle charge les deux. Ainsi, les différentes flèches de la catégorie de Ductor (norme, source, puit, effet) forment une cloture transitive. Sous forme d'effet la norme est dite «indépendante», elle est capable d'obtenir par elle même son type d'entrée et de sortie et peut alors s'assembler avec d'autres normes plus librement. En effet «Les monades sont des monoides dans la catégorie des endofuncteurs».

```
*/
    val helloEffect : Effect[Id] =
        sayHello.yes.effectOn(console_api)
    helloEffect.exec.getResult()

    sayHello.yes.readFrom(console_api).exec.getResult()

    sayHello.yes.writeTo(console_api).execWith("World").getResult()
/*
```

En l'absence de type d'entrée, `exec` est utilisé en place de `execWith` pour tranformer virtuellement la norme en son résultat.

Les environnements sbt console et scastie sont connus pour être limités au niveau de la lecture en console. Le programme ci-dessus peut être correctement exécuté avec `sbt run` qui va checher la définition annotée par @main et executer son contenu, c'est-à-dire charger l'objet en l'executant ligne par ligne

```
*/
@main
def run_tuto_api = tuto_api
/*
```

*/