Cooking a Haskell Curry with Applicative Functors

Gergő Érdi
http://gergo.erdi.hu/

Singapore Institute of Technology, March 2019.
CS Department
FP Day

Cooking a Haskell Curry with Applicative Functors

Gergő Érdi
http://gergo.erdi.hu/

Singapore Institute of Technology, March 2019.
Cooking School Department
Food Preparation Day

Recipes and ingredients

- Recipes don't contain the ingredients, only their descriptions
- But cooking a dish from a recipe needs the ingredients
- In this talk, everything else is modelled simply as pure functions, e.g. $\mathit{chopped} :: \mathit{Dish} \to \mathit{Dish}$

Recipes and ingredients

- Recipes don't contain the ingredients, only their descriptions (unless Home Chef, Blue Apron, etc.)
- But cooking a dish from a recipe needs the ingredients
- In this talk, everything else is modelled simply as pure functions, e.g. $\mathit{chopped} :: \mathit{Dish} \to \mathit{Dish}$

Running example: Haskell curry

1. Chop and fry onions
2. Chop some garlic
3. Mix curry paste and chilli
4. Stir all together

Pure functions: Infinite pantry

The pantry

$$get :: Pantry \to Ingredient \to Dish$$

```
curry :: Pantry → Dish
curry pantry = mixOf [fried (chopped onion), chopped garlic, mixOf spices]
  where
    onion = get pantry "onion"
    garlic = get pantry "garlic"
    spices = map (get pantry) ["curry paste", "chilli"]
```

Pure functions: Infinite pantry

The pantry

$$get :: Pantry \to Ingredient \to Dish$$

```
curry :: Pantry → Dish
curry pantry = mixOf [fried (chopped onion), chopped garlic, mixOf spices]
  where
    onion = get pantry "onion"
    garlic = get pantry "garlic"
    spices = map (get pantry) ["curry paste", "chilli"]
```

Problem: The rent is very high on infinitely large warehouses

IO monad: “Here, take the chequebook”

Invoice Orders

type *IO a*
instance *Monad IO*

$buy :: Shop \to Ingredient \to IO\ Dish$

```
curry :: Shop → IO Dish
curry shop = do
  onion ← buy shop "onion"
  garlic ← buy shop "garlic"
  buy myCousinsShop "gold-plated truffle lobster"
  spices ← mapM (buy shop) ["curry paste","chilli"]
  return (mixOf [fried (chopped onion), chopped garlic, mixOf spices])
```

IO monad: “Here, take the chequebook”

Invoice Orders

type *IO a*
instance *Monad IO*

$buy :: Shop \to Ingredient \to IO\ Dish$

```
curry :: Shop → IO Dish
curry shop = do
  onion ← buy shop "onion"
  garlic ← buy shop "garlic"
  buy myCousinsShop "gold-plated truffle lobster"
  spices ← mapM (buy shop) ["curry paste","chilli"]
  return (mixOf [fried (chopped onion), chopped garlic, mixOf spices])
```

Problem: Who knows what the chef will do?!

Custom monad: JIT shopping trips

A monad just for recipes

type *RecipeM a*
instance *Monad RecipeM*

buy :: *Ingredient* → *RecipeM Dish*

```
curry :: RecipeM Dish
curry = do
  onion ← buy "onion"
  garlic ← buy "garlic"
  spices ← mapM buy ["curry paste","chilli"]
  return (mixOf [fried (chopped onion), chopped garlic, mixOf spices])
```

Custom monad: JIT shopping trips

A monad just for recipes

type *RecipeM a*
instance *Monad RecipeM*

buy :: *Ingredient* → *RecipeM Dish*

```
curry :: RecipeM Dish
curry = do
  onion ← buy "onion"
  garlic ← buy "garlic"
  spices ← mapM buy ["curry paste","chilli"]
  return (mixOf [fried (chopped onion), chopped garlic, mixOf spices])
```

Problem: What if the mall is down in the valley, but the kitchen is up on a very high mountain; you could even say the kitchen is in the Clouds...

Bulk shopping (1st try)

type *RecipeM a*
instance *Monad RecipeM*

$take :: Ingredient \rightarrow RecipeM\ Dish$

$ingredientsOf :: RecipeM\ a \rightarrow [Ingredient]$
$cook :: Monad\ m \Rightarrow ([Ingredients] \rightarrow m\ Pantry) \rightarrow RecipeM\ a \rightarrow m\ a$

Bulk shopping (1st try)

```
type RecipeM a
instance Monad RecipeM

take :: Ingredient → RecipeM Dish

ingredientsOf :: RecipeM a → [Ingredient]
cook :: Monad m ⇒ ([Ingredients] → m Pantry) → RecipeM a → m a
```

Problem: This is impossible to implement: what about buying a cookbook, and cooking a recipe from that? What are the ingredients of the following recipe?

```
myRecipe = do
  pasta ← take "pasta"
  cookbook ← take "101 Pasta Sauce Recipes"
  let sauceRecipe = cookbook !! 14
  sauce ← sauceRecipe
  return (mixOf [cooked pasta, sauce])
```

Bulk shopping (2nd try)

```
shop :: [Ingredient] → IO Pantry
get :: Pantry → Ingredient → Dish
```

```
curry :: ([Ingredient], Pantry → Dish)
curry = (ingredients, cook)
  where
    ingredients = ["onion","potato","curry paste","chilli"]
    cook pantry = mixOf [fried (chopped onion)
                        , chopped garlic
                        , mixOf spices]
      where
        onion  = get pantry "onion"
        garlic = get pantry "garlic"
        spices = map (get pantry) ["curry paste","chilli"]
```

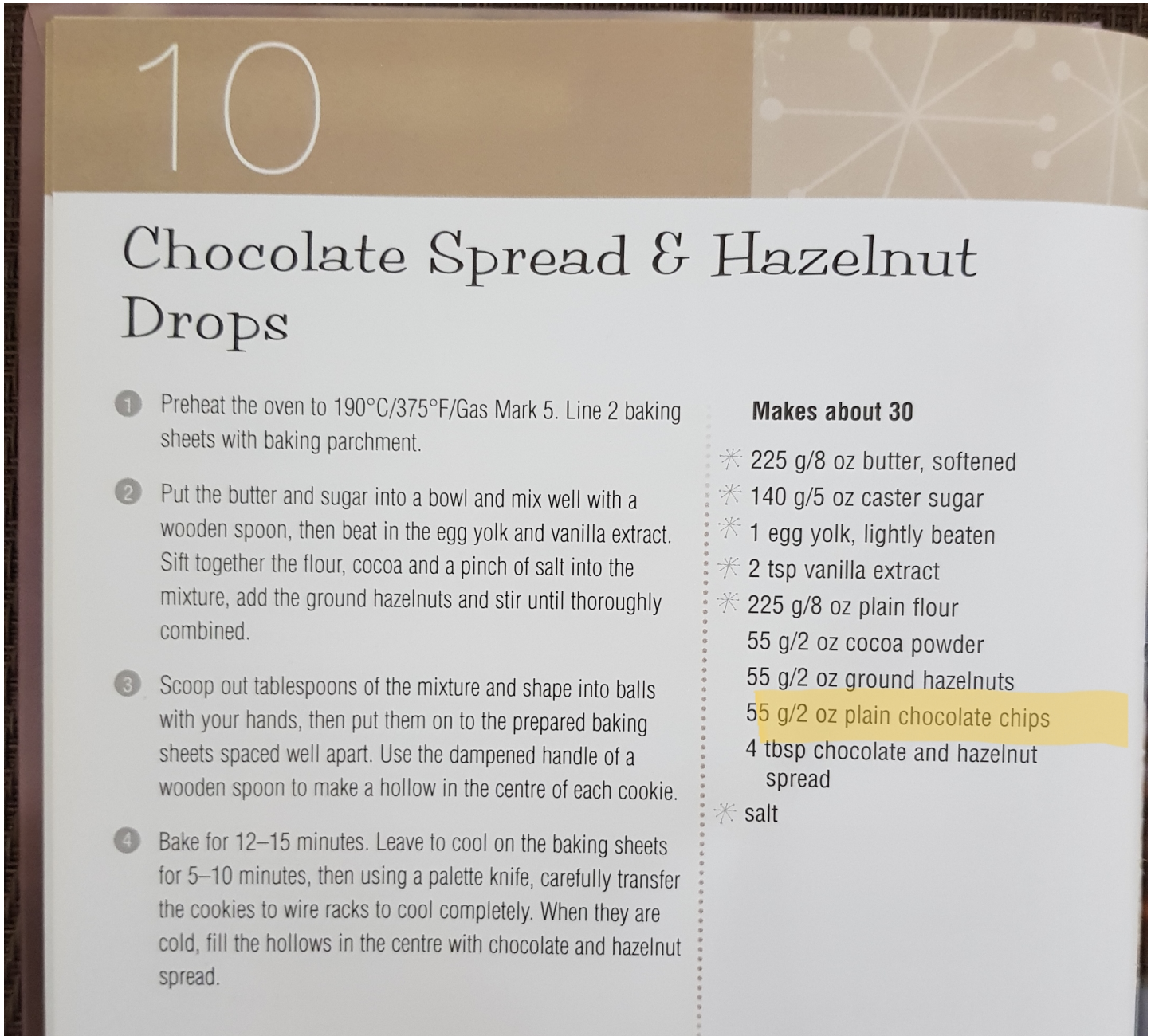
Bulk shopping (2nd try)

```
shop :: [Ingredient] → IO Pantry
get :: Pantry → Ingredient → Dish

curry :: ([Ingredient], Pantry → Dish)
curry = (ingredients, cook)
  where
    ingredients = ["onion","potato","curry paste","chilli"]
    cook pantry = mixOf [fried (chopped onion)
                        , chopped garlic
                        , mixOf spices]
      where
        onion = get pantry "onion"
        garlic = get pantry "garlic"
        spices = map (get pantry) ["curry paste","chilli"]
```

Problem: There is no connection between the ingredient list and the cooking instructions.

Bulk shopping



10

Chocolate Spread & Hazelnut Drops

1. Preheat the oven to 190°C/375°F/Gas Mark 5. Line 2 baking sheets with baking parchment.
2. Put the butter and sugar into a bowl and mix well with a wooden spoon, then beat in the egg yolk and vanilla extract. Sift together the flour, cocoa and a pinch of salt into the mixture, add the ground hazelnuts and stir until thoroughly combined.
3. Scoop out tablespoons of the mixture and shape into balls with your hands, then put them on to the prepared baking sheets spaced well apart. Use the dampened handle of a wooden spoon to make a hollow in the centre of each cookie.
4. Bake for 12–15 minutes. Leave to cool on the baking sheets for 5–10 minutes, then using a palette knife, carefully transfer the cookies to wire racks to cool completely. When they are cold, fill the hollows in the centre with chocolate and hazelnut spread.

Makes about 30

- 225 g/8 oz butter, softened
- 140 g/5 oz caster sugar
- 1 egg yolk, lightly beaten
- 2 tsp vanilla extract
- 225 g/8 oz plain flour
- 55 g/2 oz cocoa powder
- 55 g/2 oz ground hazelnuts
- 55 g/2 oz plain chocolate chips
- 4 tbsp chocolate and hazelnut spread
- salt

Static analysis with applicative functors

Applicative recipes

```
type Recipe a
instance Applicative Recipe

ingredientsOf :: Recipe a → [Ingredient]
cook :: Applicative f ⇒ ([Ingredient] → f Pantry) → Recipe a → f a

take :: Ingredient → Recipe Dish
```

```
curry :: Recipe Dish
curry = mixOf ⟨$⟩ sequenceA
    [fried ∘ chopped ⟨$⟩ onion
    , chopped        ⟨$⟩ garlic
    , mixOf          ⟨$⟩ spices
    ]
  where
    onion = take "onion"
    garlic = take "garlic"
    spices = traverse take ["curry paste","chilli"]
```

Static analysis with applicative functors

Applicative recipes

```
type Recipe a
instance Applicative Recipe

ingredientsOf :: Recipe a → [Ingredient]
cook :: Applicative f ⇒ ([Ingredient] → f Pantry) → Recipe a → f a

take :: Ingredient → Recipe Dish
```

```
{-# LANGUAGE ApplicativeDo #-}
curry :: Recipe Dish
curry = do
  onion ← take "onion"
  garlic ← take "garlic"
  spices ← traverse take ["curry paste","chilli"]
  pure (mixOf [fried (chopped onion), chopped garlic, mixOf spices])
```

Static analysis with applicative functors

Applicative recipes

```
type Recipe a
instance Applicative Recipe

ingredientsOf :: Recipe a → [Ingredient]
cook :: Applicative f ⇒ ([Ingredient] → f Pantry) → Recipe a → f a

take :: Ingredient → Recipe Dish
```

```
data Recipe a = MkRecipe
  { ingredientsOf :: [Ingredient]
  , run           :: Pantry → a
  }

take ingr = MkRecipe [ingr] (λpantry → get pantry ingr)

cook shopFor recipe = do
  pantry ← shopFor (ingredientsOf recipe)
  return (run recipe pantry)
```

Static analysis with applicative functors

Applicative recipes

type *Recipe a*
instance *Applicative Recipe*

$ingredientsOf :: Recipe\ a \to [Ingredient]$
$cook :: Applicative\ f \Rightarrow ([Ingredient] \to f\ Pantry) \to Recipe\ a \to f\ a$

$take :: Ingredient \to Recipe\ Dish$

Even better: applicatives compose!

type $Recipe = Product\ (Const\ [Ingredient])\ (Reader\ Pantry)$

So what?

- Think about desirable effects
- Think about composition
- Analyzing monadic computations is tricky ("the $\rightarrow$ in $\gg=$")
- Constraint on clients $\Leftrightarrow$ freedom of implementation
- Applicative functor interface: structure is known without running effects