G53IDS - Interim Report

Embedded Domain Specific Language for Describing Recipes in Haskell

James Burton - 4251529 - psyjb6

November 20, 2017

Contents

1   Introduction

Consider the following recipe to make a cup of tea:

```
- Boil some water
- Pour over a teabag
- Wait for 3 minutes
- Remove the teabag
- Add milk (optional)
```

This is a very simple but useful recipe that many people will perform, in some cases, many times a day over their lives. What we can realise by looking at this recipe is that it actually consists of many smaller recipes, such as boiling water and combining tea with milk, performed in a certain order. This raises the question, to what extent does the order matter and to what extent can we rearrange things in order to make the recipe more efficient? No doubt you have done this, maybe subconsciously, while cooking at home. Furthermore which steps can be done concurrently in the event that multiple people are cooking e.g. in a professional kitchen with a full brigade?

Perhaps closer to computer science, we could also ask, how could we instruct a robot to do this? After doing some research on robotic chefs it appears that not a huge number exist. There is one home cooking robot [1] which uses motion capture in order to learn recipes. In my opinion this is rather restrictive. It presumes that the human performs the recipe in the optimal manner and it would be very difficult to model a brigade system in this way. In reality there is a limited set of fundamental actions that one becomes able to perform when learning to cook. Recipes can then be performed using a sequence of these actions. Representing recipes like this would allow us to take a robot programmed to perform each of the fundamental actions and tell it how to cook literally anything.

So we've established that if we can structure recipes in a more formal way then we will have a great amount of freedom in terms of how we process them whether it be optimisation for a human chef or full on automation. My contributions / planned contributions to this are the following:

- Define a set of combinators as an EDSL in Haskell and show that they can be used to describe a wide variety of recipes (Section 2).

- The combinators describe a recipe but we then need to know how to execute a recipe as a sequence of the fundamental actions mentioned

above. Fortunately, as described above, recipes are just a combination of simpler recipes meaning that we can recursively define the actions necessary to perform complex recipes as long as we have manually defined which action are required for each combinator (Section 3).

- We now have a way to describe recipes and a way to show the sequence of actions to complete a recipe but now we need to apply them to something. Using the operational semantics of recipes we can optimise and schedule recipes for a given kitchen system. We can then express this in several ways including printing steps, drawing the cooking process as a graph or simulating the recipe within the given kitchen setup (Section 4).

- It may also be useful to provide an intermediate language between informal English and our combinators, some sort of markup language, that can then be parsed in.

2 Describing Recipes

In this section I shall outline the EDSL for describing recipes. The EDSL is implemented in the functional programming language called Haskell. Haskell has been used for many EDSLs in the past [2]. Michael Snoyman, creator of Yesod (a Haskell Web Framework), stated many advantages of using Haskell for EDSLs among which were that the type system helps catch mistakes and Haskell allows us to overload almost any syntax [3].

2.1 A Cup of Tea

Consider our cup of tea example from earlier. We can start by defining all of our ingredients:

```
milk, teabag, water :: Recipe
milk = Ingredient "milk"
teabag = Ingredient "teabag"
water = Ingredient "water"
```

The next step is to start describing what we want to transform those ingredients into for example:

```
boilingWater, blackTea :: Recipe
boilingWater = heat 100 water
blackTea = (teabag >< boilingWater) >>> Wait 5
```

combinator1	combinator2
combinator3	combinator4

Table 1: Combinators for defining recipes

We've introduced three things here. `heat` is simply a function meaning heat the given recipe to the given temperature. `><` is our combine operator and simply means mix the given recipes together. Finally `>>>` is our sequencing operator meaning do the first recipe then the second. The types for these are as follows:

```
heat :: Temperature -> Recipe -> Recipe
(><) :: Recipe -> Recipe -> Recipe
(>>>) :: Recipe -> Recipe -> Recipe
```

We can now use the above to define our cup of tea recipe as follows:

```
cupOfTea :: Recipe
cupOfTea = blackTea >< milk
```

Now you may notice that we haven't mentioned preparation of ingredients for example measuring how much of them to use. While experimenting with the basic representation of a recipe and the different ways to interpret it we thought it would be beneficial to keep the combinators as simple and abstract as possible. This means that, for now, we will be describing recipes "cooking show" style where we presume everything is conveniently measured and prepared in a bowl next to the chef.

2.2 Currying - But Not What You Think

In this section I shall introduce the full set of combinators we are currently working with and proceed to build up a more complex recipe.

2.3 Conditionals

2.4 Moving Forward

At the time of writing this interim report, the combinators we have are decent but not perfect. There are several crucial pieces of information not captured such as measurements and, as mentioned above, conditionals and optional recipes are not yet implemented.

Now that we have a more detailed idea of the components of the project, we can move towards implementing them. That will naturally provide an

opportunity to scrutinise the set of combinators we are working with by analysing any issues we run into.
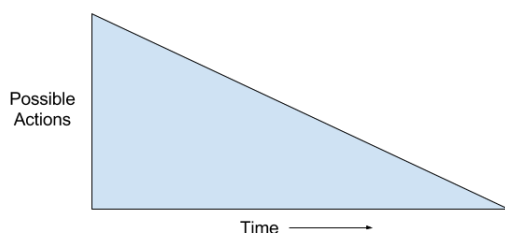
3 Executing a Recipe

Figure 1: Number of possible actions over time.

4 Implementation

5 Progress

This project shall use the scrum development methodology. With the gaps between supervisor meetings acting as the sprints. The scrum methodology seems most appropriate for this project as it is naturally very flexibly and doesn't require a huge amount of detailed planning in advance. This suits the research nature of the project as it is impossible to precisely state what
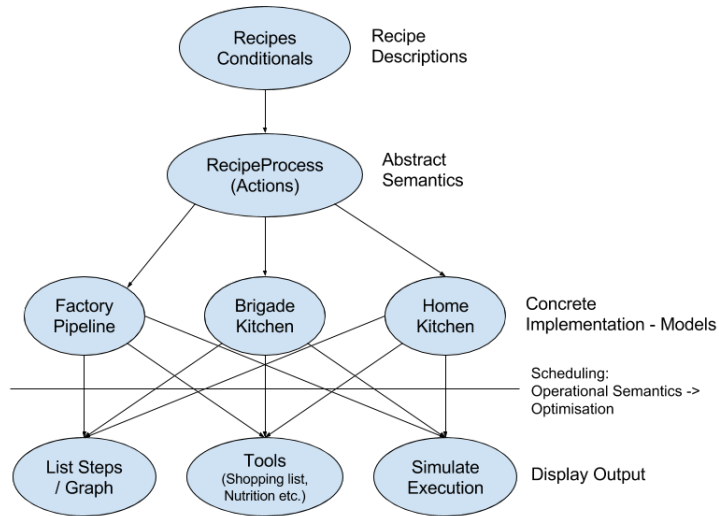
Figure 2: Components of the recipe system.

will be done and when. Each sprint offers an opporuntity to re-prioritise and alter tasks based upon the progress and findings of the previous sprint.

5.1   Project Management

5.2   Contributions and Reflections

6   Related Work

References

[1] The Guardian. 2015. *Future of food: how we cook.* `https://www.theguardian.com/technology/2015/sep/13/future-of-food-how-we-cook`

[2] Paul Hudak. Domain Specific Languages. Department of Computer Science, Yale University, December 15, 1997.

[3] Michael Snoynman. O'Reilly Webcast: Designing Domain Specific Languages with Haskell. January 4, 2013. `https://www.youtube.com/watch?v=8k_SU1t50M8`

[4] Simon Peyton Jones, Microsoft Research, Cambridge. Jean-Marc Eber, LexiFi Technologies, Paris. Julian Seward, University of Glasgow. Composing contracts: an adventure in financial engineering. August 17, 2000.