Approximating Knowledge of Cooking in High-Order Functions, a Case Study of Froglingo

Kevin Xu, Jingsong Zhang, Shelby Gao

Bigravity Business Software LLC
2306 Johnson Circle, Bridgewater, New Jersey, United States
{kevin, jingsong, shelby}@froglingo.com

Abstract. Without accurately representing the knowledge of cooking, a computer would never react as if it were a professional chef. Accumulating the knowledge into a computer presentation, maintaining the accuracy of the presentation, and offering practically valuable advice face many challenges in the current technologies. Froglingo is a computer system that uniformly represents both business data and business logic as high-order functions. It has a set of built-in operators stemming from the relationships among the high-order functions. As a case study in this paper, we use Frolingo to approximate the knowledge of cooking and to approximate the reasoning on the knowledge.

Keywords: High-order Function, Ordering Relation, Query, Similarity, Adaptation, Approximation.

1 Introduction

To be a valuable recipe advisor, a system shall have a data presentation semantically approximating the knowledge of cooking including adaptation processes. In other words, the system allows users to accumulate data that approaches closely to the knowledge of cooking, and thereafter to accurately offer advice reflecting the managed data.

To implement such a system, the authors suggest a data model that generally and therefore uniformly represents knowledge. Only with the generality, we have a chance to unlimitedly accumulate data that approaches the knowledge more closely; and the mission would not be hindered by any pre-mature assumptions of data presentation.

Modeling knowledge in high-order functions offers an opportunity. A collection of high-order functions embed rich relationships among them [6]. In addition, a class of total recursive functions, in which high-order functions are the sole members, is the upper bound of what a computer can do practically. (We attempt to exclude those partial recursive functions which don't terminate on some inputs and therefore are not desired in software practice [8 and 11]).

In this paper, we use Frolingo to approximate the knowledge of cooking. Froglingo has a data model, called the EP (Enterprise-Participant) data model. The EP data model is equivalent to a class of total recursive functions [9]. Therefore it mathematically defines the entire semantic space for software applications. The variables and the sequential terms in Froglingo, beyond the EP data model, allow one to construct business logic, i.e., infinite data [10]. The addition of the new concepts

doesn't alter the semantic space defined by the EP data model. Instead, it allows one to stuff the infinite semantic space in finite expressions. As a result, the relationships among high-order functions are preserved and thereafter the built-in operators introduced by the EP data model can be equally applicable to the data expressed in variables [6]. The unified semantic space for both business data and business logic makes the architecture of Froglingo a monolith [7]. Froglingo reaches the greatest possible generality in uniformly representing both business data and business logic including knowledge.

The system, "Chef Froglingo", implemented for the 2010 Computer Cooking Contest (CCC), has the following objectives: 1) A consistent method of representing the knowledge of cooking including adaptation; 2) The ability of accepting a format of the inputs producible by the customers who don't know the knowledge representation of system; 3) A reasonable precision of answering customer queries as if it were a professional chef.

In Section 2, we briefly outline the main concepts of Froglingo. For a detailed introduction of Froglingo syntax, readers may reference the user's guide [10]. In Sections 3, we give a Froglingo presentation of the primary cooking concepts such as ingredient, dish origin, recipe, and menu. At the same time, we discuss our strategy of parsing the XML document file, the Recipe Book from CCC, containing a set of recipes. (A sample XML block for a recipe is attached in Appendix A for reference.) In Section 4, we describe how customer queries are expressed and answered by a Froglingo built-in operator. In Section 5, we describe the implementation of an adaptation task, and therefore demonstrate that the system can approach the accuracy of a professional chef in responses to future events as close as the volume of the data approximating the knowledge of cooking is allowed. In Section 6, we make a few remarks about system architecture, related work and future work.

2 Froglingo

In traditional data models, an entity is either dependent on one and only one other entity, or independent from the rest of the world. The functional dependency in relational data model and the child-parent relationships in hierarchical data model are the typical examples. This restriction, however, doesn't reflect the complexities of the real world that are manageable by using a computer. The EP (Enterprise-Participant) data model suggests that if an entity is dependent on others, it precisely depends on two other entities. Drawing the terminologies from the structure of an organization or a party, one depended entity was called enterprise (such as organization and party), the other called participant (such as employee and party participant), and the dependent entity called participation. An enterprise consists of multiple participations. Determined by its enterprise and its participant, a participation yields a value, and this value in turn is another enterprise.

Specifically, a Froglingo database is a set of assignments, an assignment has an assignee and possibly an assigner, and both assignee and assigner are terms. The concept of term is essential. A term is a constant, an identifier, a variable, or a pair of parenthesized terms. A few sample terms are `3.14`, `$x`, `"A String"`, `mike`, `(mike`

`salary`), and `((a b) (c d))`. A sample database is: {`mike salary = 3.14; fac 0 = 1; fac $n = ($n * (fac ($n - 1))); ((a b) (c d)) = something; x y z;`}.

A term is called a sub-term of itself. When a term consists of a pair of two other terms, it is called an application, where the first element is the function, and the second the argument. Given a sub-term of a term, recursively, an inner sub-term of the sub-term is also a sub-term of the given term. For example, `a`, `c`, `(c d)`, and `((a b) (c d))` are sub-terms of the term `((a b) (c d))`. If the right sub-term of an application is not another application, the parentheses surrounding the application don't have to be written. For example, `((a b) (c d))` is equivalent to `a b (c d)`. In this article, we alternatively use the phrase "plus-term" in the place of the function of an application to avoid any confusion with the word "function" in generic term.

In terms of ordering relations, we call an application *neutrally* depends on a sub-terms, i.e., either its function, its argument, or any inner sub-term, in contrast to the fact that the function and the argument play slightly different roles in determining the application. For example, `a b (c d)` neutrally depends on `c`. Because of assignment, two or more terms can be derived to be equal in a given database. Given an application and its equal peers in a database, we say that the application or any one of its peers is functionally derivable from the function of the application, argumentatively derivable from the argument, and *neutrally* derivable from a sub-term of the application. Given the sample database discussed earlier, for example, both `a b (c d)` and `something` are neutrally derivable from c.

3 Knowledge of Cooking

During the design of the knowledge representation, we first consider of arranging the knowledge of cooking in certain orders that are available among high-order functions. The aim is to leverage Froglingo built-in operators stemming from the orders in support of the queries expressible from customers. Secondly, we consider the challenges in retrieving data from the given Recipe Book, a loosely structured text file. In this section, we introduce a Froglingo presentation of preliminary concepts in the knowledge of cooking. The query and adaptation processes are discussed separately in Sections 4 and 5.

3.1 Ingredient

An ingredient may have an ingredient type, and an ingredient type may have its parent type. Sometimes, an ingredient may belong to multiple types. For example, chicken wing has the type chicken, and chicken has the type meat (or poultry precisely). Chicken also belongs to the type broth for the ingredient chicken broth. Here are a few examples in Froglingo terms:

```
meat beef corned;
meat beef (short loin);
meat chicken;
vegetable artichoke;
vegetable (Chinese artichoke);
soda (baking powder);
broth (meat chicken);
broth vegetable;
```

To collect all the ingredients, we first inventory a list of preliminary ingredients such as meat, vegetable, nut, alcohol, grain, sauce, herb, and spice. To obtain granular ingredients appeared in Recipe Book, e.g., cilantro and all-purpose flour, we collect all the possible names from an ingredient dictionary [3], and classify each of them as one of the preliminary ingredients by parsing the body of the ingredient dictionary.

3.2 Preparation Method and Step

Preparation method is an important piece of information in a recipe. It may determine the type of dish. Taking it into consideration in our data presentation helps the case study more closely modeling the interactions between customers and wait staffs. Like ingredients, it participates in preparation steps. For example: `bake`, `grill`, `steam`, and `stir fry` are preparation methods.

A recipe normally involves multiple preparation methods and therefore a sequence of preparation steps, and each step may have its sub steps. To avoid the difficulty of parsing the multiple steps in the Recipe Book, we identify one preparation method for a recipe. If there are multiple methods appeared between a preparation block (i.e., a pair of <PR> and </PR>), we choose only one by pre-defining weights for all the methods. For example, if both `grill` and `marinate` appear in a recipe preparation block, we will choose `grill` because `grill` is assigned with a heavier weight.

The cooking temperatures and the cooking time periods of preparation methods are the factors considered when the weights are assigned.

3.3 Origin

Many ingredients and dishes originate from certain cultures. For example, curry is generally regarded as an Indian ingredient, wok as an Asian equipment, and a Fajita beef as a Mexican dish. Therefore, a recipe may have multiple origins. To support this feature, we inventory a list of food origins. Here are a few sample terms inventoried in Froglingo terms:

```
Asian Chinese Szechuan;
French Dijon;
Irish Kilkenny;
```

The complete list of origins is to be identified through a world atlas.

When the ingredient dictionary is parsed (as discussed in Section 3.1), and when an XML block between <RECIPE> and </RECIPE> is parsed, we search the key

words from the origin list, and tag ingredients and preparation steps if a key word is found.

3.4 Dish Type

The food described by a recipe can be served as an appetizer, a main course, or a dessert. It also can be simultaneously served for two or more dish moments. Instead of explicitly labeling them for the purpose of determining dish moments, we derive dish moments according to the ingredients and cooking methods of recipes. For example, cafe, tea, sugar, chocolate, and refrigerator most likely imply dessert; and soup, salad, fry, and the rest of non-sweet foods can imply appetizer.

Appetizers and desserts are normally further classified to different types, such as cake, cookie, ice cream, and pizza. We observe that recipe titles sometimes embed this information. The Chef Froglingo uses the embedded information to identify dish types.

We also allow customers to express key words appeared in the title of recipes, e.g., Mom, and Carne Asada. It facilitates customers to make requests easier.

3.5 Synonym

Ingredients may have multiple names. We identify them when the ingredient dictionary is processed (discussed in Section 3.1). Here are a few sample pairs of equivalent names in Froglingo assignments:

```
lichee = lichi;
chile = hot pepper;
```

3.6 Recipe

We discussed many concepts in the earlier sub-sections from which our simplified version of the concept recipe is assembled. To Froglingo, however, all of them are indiscriminately stored as data, i.e., high-order functions. Here we give the database for a few sample recipes.

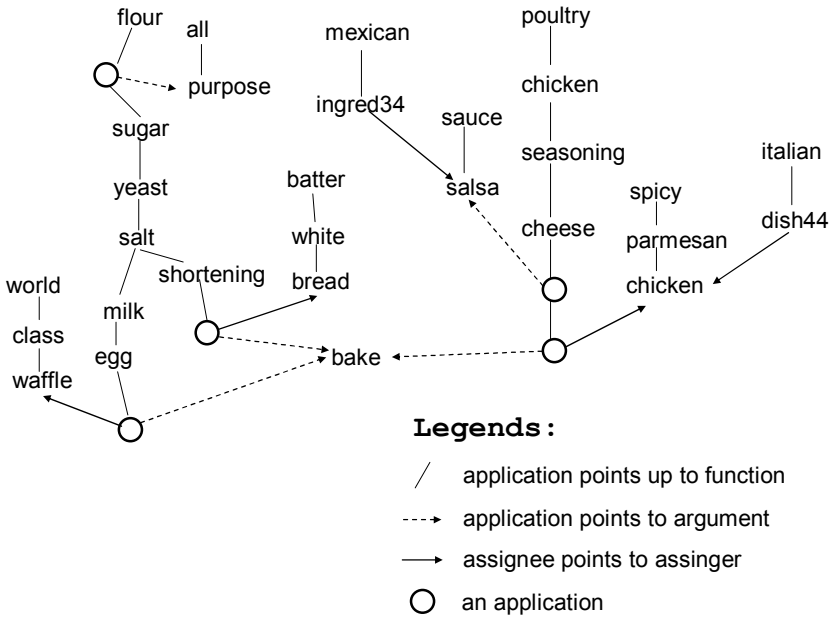
```
(flour (all purpose)) sugar yeast salt milk egg bake
     = world class waffle;
(flour (all purpose)) sugar yeast salt shortening bake
     = batter white bread;
(meat chicken) cheese seasoning (sauce salsa) bake
     = spicy parmesan chicken;
mexican ingred34 = sauce salsa;
Italian dish44 = spicy parmesan chicken;
```

To better understand the knowledge representation that is arranged in orders among high-order functions, we provide a graphical view of the database.

To support the queries from customers, we keep the text message from “Recipe Book” for each recipe. Here is an example:

```
spicy parmesan chicken title = "Spicy Parmesan Chicken";
spicy parmesan chicken body  = "...";
```
(The texts in the XML block <PR>)

Here the terms `title` and `body` are the key words indicating that the text strings of the recipe title and body follow as assigners.



3.7 Menu

A term in Froglingo can be used to express a menu too. Here is a sample expression for a dinner:

```
dinner (Potato Salad 2) (Spicy Parmesan Chicken) (Sorry Cake);
```

However, enumerating all the possible dinner choices is not preferred. We choose to have variables for the set of dinners producible from dishes. The following sample expression is equivalent to a set of dinners including the one described earlier:

```
dinner $s:[$s {=+ salad or $s {=+ soup]
      $m: [$m {=+ chicken or $m {=+ marinate]
      $d: [$m {=+ cake];
```

4 Query

As readers might have realized, the Froglingo expressions for recipes, as the examples demonstrated in Section 3, embed all the relevant information including ingredients, preparation methods, and cultures. In addition, this information is arranged in certain

ordering relations, i.e., the dependent relations and pre-ordering relations. Given an entity, e.g., all-purpose flour, salt, bake, and Chinese, one can find all the dishes that include the entity by navigating the ordering relations. When a customer gives a list of entities, the same process is repeated for one entity. At the end, the remaining recipes are those satisfying the initial request. This process is done by the built-in operator `(=`, a pre-ordering relation in Froglingo (called neutrally derivable in this article). Here is a sample query expression retrieving the recipes that have all-purpose flour as an ingredient and that use the preparation method bake:

```
select $dish title, $dish body where
      $dish (= flour (all purpose) and
      $dish (= bake and
      $dish title != null;
```

In the expression, we use the clause `$dish title != null` to only retrieve those terms representing recipes because not all the terms in the database have correspondences of recipes.

It is clear that the recipes "World Class Waffles" and "Batter White Bread" will be retrieved for this query expression.

To support negation, we simply use the negation operator available in Froglingo `not`. When the boolean expression `not $dish (= egg` is added into the `select` operation above, the query will only return the recipe "Batter White Bread".

The dietary practices of vegetarian, nut-free, and no alcohol are translated into the following expressions:

```
not $dish (= meat;
not $dish (= nut;
not $dish (= alcohol;
```

where all the meats have the ingredient type `meat`, all the nuts have the type `nut`, and all the alcohol beverages have the type `alcohol`.

What we have done so far is eventually to support a user interface such that the customers who don't know the system representation of recipes can easily express their requests, and the system can precisely process the requests. The resulting system allows users to enter a list of phrases or words delimited by a comma, and each phrase

or word can be preceded with the word "no" for negation. See a sample screenshot of the interface above.

In the sample query string in the screenshot, the system doesn't break the phrase "all-purpose flour" into individual words, but translate it to the Froglingo term `flour (all purpose)`. In other words, customers are required to be precise on the phrases representing ingredients, preparation methods (such as stir fry), and recipe titles if they want to be specific. If a customer is not sure about the exact spelling of a phrase, he or she should enter individual words divided by commas. Individual words lead to more recipes in return than phrases do.

The same interface is equally applicable to dinners that have variables in their expressions. When customers enter a string: "dinner, no soup, chicken", for example, the system will retrieve all the dinner choices, each of which will include a salad as the starter, a main dish with chicken, and an ice cream as the dessert.

5 Adaptation

If we view the knowledge of cooking discussed above as experience, or precisely historical events, the concept of adaptation in the field of Case-Based Reasoning refers to a systematic ability of proactively planning future events that may not repeat the past experience [5]. Adaptation may mean various systematic abilities. The one required in the 2010 CCC requests a system to response reasonably when a customer wants a specific recipe and certain ingredients for the recipe are not available. The reasonability may mean different things from time to time and from person to person. When there is not a grill available, for example, a recipe which requires fresh salmon and a grill may be desired to be adapted to a recipe with a broiler some times, and to a recipe of a Sushi at another occasion.

For the 2010 CCC, what we choose as a reasonable response is to adapt some existing and similar recipes to replace those wanted but not available. Further, we rule that two applications in a Froglingo database are similar if they share the same function, i.e., the plus-term. This rule applies to all the cooking related entities including ingredients, preparation methods, preparation steps, and origins. Given the sample ingredients in Section 3.1, for example, `meat beef corned` and `meat beef (short loin)` are similar because they share the same plus-term `meat beef`. Similarly, the sample preparation steps `(flour (all purpose)) sugar yeast salt milk` in Section 6 is similar to `(flour (all purpose)) sugar yeast salt shortening`.

Given a recipe and a set of excluded cooking entities, i.e., a set of entities that are required by the recipe but not available, we attempt to identify alternative recipe(s), as a solution for the adaptation task of the 2010 CCC, by taking the following steps:

1). Identify the set of non-excluded entities required by the recipe, and retrieve those recipes that are neutrally derivable from the non-excluded entities.

2). If there is no recipe returning from Step 1, we relax the constraints by taking a non-excluded entity as an excluded entity, and rerun Step 1.

3). Taking the returning recipes from Step 1 as the initial input, we select those neutrally derivable from the plus-terms of the excluded entities, but filter out those

recipes that are neutrally derivable from the excluded entities. The outcome would be those similar recipes, but not the recipes requiring the excluded entities.

The interface introduced in Section 4 is also used for the adaptation task. For example, the request strings "World Class Waffle, no egg" is for the query: "I want World Class Waffles, but I don't have egg. Please show me the similar recipes". This query would return a set of recipes including Batter White Bread, but not World Class Waffle.

When taking a member out of the set of non-excluded entities in Step 2, we choose the one that is less important to the recipe. We rule again that the preparation method of a recipe is the least important, and an ingredient toward the end of the ingredient list in a recipe is less important than a one closer to the beginning.

The relaxation of query constraints in Step 2 is the initial step of the approximation in searching for similar recipes, and the additional constraints based on similar entities in Step 3 is the second step approaching closer to the most similar recipes.

6 Remarks

Reasoning on the knowledge of cooking has been identified as a typical domain in the field of case-based reasoning in [2]. Under the benchmark of CCC, many systems were implemented for this domain. The tools used for the reasoning in the implementation were based on traditional technologies such as the propositional logic for WikiTAAABLE [1], and Java for CookIIS [4].

Aimed to be a true recipe advisor, Chef Froglingo represents knowledge of cooking in high-order functions, and uses a neutral derivative relation, i.e., (=, to approximate the adaptation processes of substituting a desired but not available dish with the most similar ones. The real cooking knowledge is much more complicated than what we have discussed in this article. But this approach can reach our expectation if we accumulate as much knowledge of cooking, and develop as many adaptation tasks as our business needs. The knowledge accumulation process is at multiple dimensions, such as the dimension of nutrition categories, e.g., protein and fat levels, the dimension of additional ingredient attributes, e.g., volume and texture, and the dimension of more complicated preparation steps, e.g., sub steps and cooking equipment.

Chef Froglingo, available at http://www.froglingo.com/ccc/index.html, is implemented in Froglingo, a single tool for business data, business logic, and web server. Therefore, we also easily offer a web page allowing users to update recipes.

References

1 F. Badra, J. Cojan, A. Cordier, J. Lieber, T. Meilender, A. Millle, P. Molli, E. Nauer, A. Napoli, H. Skaf-Molli, and Y. Toussaint. "Knowledge Acquisition and Discovery for the Textual Case-Based Cooking System WIKIAAABLE". 8th International Conference on Case-Based Reasoning – ICCBR 2009, Workshop Proceedings 2009, Page 249 - 258.

2 K. J. Hammond. “CHEF: A model of case-based planning.” AAAI Proceedings of the 5th National Conference on Artificial Intelligence, page 267-271. Morgan Kaufmann, 1986.
3 S. T. Herbst, “Food Lover’s Companion, The (Barron’s Cooking Guide) 2nd Edition”, available at: http://www.epicurious.com/tools/fooddictionary.
4 N. Ihle, R. Newo, A. Hanft, K. Bach, M. Reichle. “CookIIS – A Case-Based Recipe Advisor”. 8th International Conference on Case-Based Reasoning – ICCBR 2009, Workshop Proceedings 2009.
5 D. Leake. “Case-Based Reasoning: Experience, Lessons, and Future Directions”. Menlo Park: AAAI Press/MIT Press, 1996.
6 K. H. Xu, J. Zhang, S. Gao. “High-Order Functions and their Ordering Relations”. The Fifth International Conference on Digital Information Management, 2010.
7 K. H. Xu, J. Zhang, S. Gao. “Froglingo, A Monolithic Alternative to DBMS, Programming Language, Web Server, and File System”. The Fifth International Conference on Evaluation of Novel Approaches to Software Engineering, 2010.
8 K. H. Xu, J. Zhang, S. Gao. “An Assessment on the Easiness of Computer Languages”. To appear in the Journal of Information Technology Review, 2010.
9 K. H. Xu, J. Zhang, S. Gao, R. R. McKeown. “Let a Data Model be a Class of Total Recursive Functions”. The 2010 International Conference on Theoretical and Mathematical Foundations of Computer Science (TMFCS-10), 2010.
10 K. H. Xu, J. Zhang. “A User’s Guide to Froglingo, An Alternative to DBMS, Programming Language, Web Server, and File System, Release 1.0, January 2010”. Available at the website: http://www.froglingo.com/ FrogUserGuide10.doc.
11 K. H. Xu, J. Zhang, S. Gao. “Assessing Easiness with Froglingo”. The Second International Conference on the Application of Digital Information and Web Technologies, 2009, page 847 - 849.

Appendix A: A Sample XML Block from Recipe Book

<RECIPE>
<TI>Spicy Parmesan Chicken</TI>
<IN>2 Whole chicken breasts, split and fat removed -or-</IN>
<IN>1 Broiler chicken, cut up and skin removed</IN>
<IN>1/4 c Grated Parmesan cheese</IN>
<IN>1 ts Italian seasoning</IN>
<IN>1/2 c Mozzarella cheese, (2 ounces)</IN>
<IN>3/4 c Salsa</IN>
<PR>Preheat oven to 375 degrees. Place chicken, bone-side down, in a 9x13-inch baking pan. Combine Parmesan cheese and Italian seasoning and sprinkle over chicken. Bake for 15 minutes if using two breasts, 45 minutes for a broiler chicken. Sprinkle with mozzarella cheese and bake 1 more minute. Serve with salsa.
</PR>
</RECIPE>