SOUR CREAM:
Toward Semantic Processing of Recipes

Dan Tasse and Noah A. Smith

CMU-LTI-08-005

Language Technologies Institute
School of Computer Science
Carnegie Mellon University
5000 Forbes Ave., Pittsburgh, PA 15213
www.lti.cs.cmu.edu

SOUR CREAM: Toward Semantic Processing of Recipes

Dan Tasse and Noah A. Smith
School of Computer Science
Carnegie Mellon University
dtasse@andrew.cmu.edu, nasmith@cs.cmu.edu

May 2008

1 Introduction

We present preliminary work on SOUR CREAM (System to Organize and Understand Recipes, Capacitating Relatively Exciting Applications Meanwhile). The aim of this project is to develop new techniques for semantic parsing by focusing on the domain of cooking recipes. This report details the MILK meaning representation language and CURD, a database of recipes annotated in the MILK language. We also detail preliminary efforts at semantic processing using this dataset.

2 MILK: Minimal Instruction Language for the Kitchen

In this section we present MILK, the Minimal Instruction Language for the Kitchen. In designing this language, we aimed to create a concise, yet complete, set of instructions that represent the actions demanded by imperative statements in recipes. MILK is based on first-order logic, but there is a notion of temporal order and creation/deletion of ingredients. It will provide a basic machine-readable target language for our parsing efforts; the MILK framework should allow a variety of useful applications. We have aimed for a medium-grained representation: a MILK translation of a recipe cannot identify in detail each action that occurs during cooking, but it will offer some useful information about each step in the recipe.

2.1 State

The MILK language includes three primitive types:

- `ingredient` Any food ingredient that goes into the recipe. Can be created and deleted.
- `tool` Any non-food tool, container, or cooking utensil. Can be created; cannot be deleted.
- `string` An alphanumeric string that helps the recipes to be human-readable or to provide extra information that cannot otherwise be represented in MILK.
- MILK also allows the type `ingredient set`, which is denoted using curly braces:

  ```
  {ing0, ing1, ing2}
  ```

  This is purely syntactic sugar. Any operation which takes as input an `ingredient set` could instead take two parameters at a time. For example, instead of:

```
combine({ing0, ing1, ing2}, ing3, "all the ingredients", "")
```

we could write:

```
combine(ing0, ing1, ing3, "the first two ingredients", "")
combine(ing3, ing2, ing4, "all the ingredients", "")
```

A state in the world can be described as a tuple:

$$< I, T, S, I_d, T_d, C > \tag{1}$$

where I is a set of ingredients, T is a set of tools, S is a set of strings, $I_d : I \times S$ is a relation between ingredients and their descriptions, $T_d : T \times S$ is a relation between tools and their descriptions, and $C : T \times I$ is a relation describing which tools contain which ingredients. Note that the MILK framework permits multiple descriptions for an ingredient or tool.

2.1.1 Start State

At the beginning of every recipe, there are no ingredients, tools, or strings. That is, the sets I, T, and S, and the relations I_d, T_d, and C are all empty.

2.1.2 Final State

At the end of a recipe, there may be tools in T (and this may be a valid list of which dishes need to be washed!). However, all ingredients in I should eventually be served or otherwise deleted. We will nonetheless consider a parse valid if it ends in a state with some ingredients remaining, as not all recipes end with an instruction to serve them.

2.2 Actions

MILK includes 12 actions. They correspond roughly to void functions with side effects. In this section, the notation "the ingredient (`ing`, "desc") is created" means that:

- The ingredient `ing` is created in I.
- The string "desc" is created in S.
- The pair (`ing`, "desc") is added to I_d.

Similarly, "the ingredient (`ing`, "desc") is deleted" (or even just "the ingredient `ing` is deleted") means that:

- The ingredient `ing` is deleted from I.
- The string "desc" is deleted from S.
- The pair (`ing`, "desc") is deleted from I_d.

The same is true for tools (using T and T_d instead of I and I_d).

2.2.1 Entity creation

Two actions create objects:

- `create_ing(ingredient i, string description)`
  Creates a new ingredient with the given name and description. `description` cannot be null or empty. For example:

  ```
  create_ing(ing0, "1 cup flour")
  ```

  results in the creation of the ingredient (`ing0`, "1 cup flour").

- `create_tool(tool t, string description)`
  Creates a new tool with the given name and description. `description` cannot be null or empty. For example:

  ```
  create_tool(t0, "blender")
  ```

  results in the creation of the tool (`t0`, "blender").

2.2.2 Combining and separating

Combining and separating ingredients are both important actions, and often make up the bulk of a recipe.

- `combine(ingredient set ins, ingredient out, string outdesc, string manner)`
  Combines the ingredients in the list `ins`, creating the new ingredient (`out`, outdesc). Only `manner` can be null or empty; all other parameters must be specified. For example:

  ```
  create_ing(ing0, "strawberries")
  create_ing(ing1, "blueberries")
  combine({ing0, ing1}, ing2, "mixed fruit", "toss")
  ```

  results in the deletion of `ing0` and `ing1` and the creation of the ingredient (`ing2`, "mixed fruit"). The instruction "toss" describes how they should be combined; it does not affect the state. Another example:

  ```
  create_ing(ing0, "1 cup flour")
  create_ing(ing1, "2 cups water")
  combine({ing0, ing1}, ing2, "dough", "")
  ```

  results in the creation of the ingredient (`ing2`, "dough"). The way in which the ingredients are combined is unspecified.

- `separate(ingredient in, ingredient out1, string out1desc, ingredient out2, string out2name, string manner)`
  Separates the ingredient `in` into `out1` and `out2`. This deletes `in` and creates `out1` and `out2`. Only `manner` can be null or empty; all other parameters must be specified. For example:

  ```
  create_ing(ing0, "trail mix")
  separate(ing0, ing1, "raisins", ing2, "peanuts and almonds", "")
  ```

  results in the deletion of `ing0` and the creation of (`ing1`, "raisins") and (`ing2`, "peanuts and almonds").

2.2.3 Location

Two actions alter the location of ingredients (which tool they are in).

- `put(ingredient i, tool t)`
  This represents placing an ingredient into a tool. No ingredients or tools are created or deleted. Neither `i` nor `t` can be null. For example:

  ```
  create_ing(ing0, "1 lb. ground beef")
  create_tool(t0, "saucepan")
  put(ing0, t0)
  ```

  results in the creation of `ing0`, and `t0`. The pair (`t0`, `ing0`) is added to C.

- `remove(ingredient i, tool t)`
  This represents removing an ingredient from a tool. No ingredients or tools are created or deleted. Neither `i` nor `t` can be null. Before this is called, `i` must be in `t`; that is, the pair (`i`, `t`) must be in C. For example:

  ```
  create_ing(ing0, "1 lb. ground beef")
  create_tool(t0, "saucepan")
  put(ing0, t0)
  remove(ing0, t0)
  ```

  results in the creation of `ing0` and `t0`. Note that if the tool did not contain the ingredient, this will raise an error. For example:

  ```
  create_ing(ing0, "cake batter")
  create_tool(t0, "cake pan")
  remove(ing0, t0)
  ```

  will cause an error because `t0` does not contain `ing0`.

2.2.4 Altering ingredients

Four actions may be used to change an ingredient. They all have the same list of arguments: they all can use a tool, they all create a new ingredient, and they all allow a string to be used to describe how this action is done.

- `cut(ingredient in, tool t, ingredient out, string outdesc, string manner)`
  Represents cutting an ingredient into pieces. `in` is deleted and (`out`, outdesc) is created. `t` and `manner` can be null. For example:

  ```
  create_ing(ing0, "1 onion")
  cut(ing0, , ing1, "chopped onion", "chop into small pieces")
  ```

  will result in the creation of (`ing0`, "chopped onion"). Another example:

  ```
  create_ing(ing0, "8 oz. cheddar cheese")
  create_tool(t0, "cheese grater")
  cut(ing0, t0, ing1, "grated cheese", "")
  ```

results in the creation of (`ing0`, "grated cheese").

- `mix(ingredient in, tool t, ingredient out, string outdesc, string manner)` Represents mixing or stirring an ingredient using `t`. `in` is deleted and (`out`, outdesc) is created. `t` and `manner` can be null. For example:

```
create_ing(ing0, "1 jar of tomato sauce")
mix(ing0, ing1, "stirred tomato sauce",
    "stir with a wooden spoon until the consistency is uniform")
```

  Note that this is distinct from the `combine` operation, which merely entails putting ingredients together. Recipes often call for combining ingredients and then mixing them, as shown in this example:

```
create_ing(ing0, "1 cup flour")
create_ing(ing0, "1/2 cup sugar")
combine({ing0, ing1}, ing2, "flour and sugar", "")
mix(ing2, ing3, "mixed flour and sugar",
    "stir together until uniform")
```

- `cook(ingredient in, tool t, ingredient out, string outdesc, string manner)` Represents cooking or heating an ingredient in any way using `t`. `in` is deleted and (`out`, outdesc) is created. `t` and `manner` can be null. For example:

```
create_ing(ing0, "1 cup water")
create_tool(t0, "tea kettle")
cook(ing0, t0, ing1, "hot water", "boil")
```

  will result in water boiled in a tea kettle.

- `do(ingredient in, tool t, ingredient out, string outdesc, string manner)` Represents any action done to an ingredient. `t` and `manner` can be null. This is a very open-ended action, as it is intended to encompass all actions that do not fit into any other category. It could be used for every action, as here:

```
create_ing(ing0, "3 carrots")
do(ing0, , ing1, "chopped carrots", "chop")
```

  Such use is discouraged, however, as it does not deliver as much information. (the example of chopping carrots is better suited by a `cut` action.) `do` is best reserved for actions that cannot be described any other way. For example, when making dumplings or pierogies:

```
create_ing(ing0, "dough")
do(ing0, , ing1, "dumplings", "fold into dumpling shapes")
```

2.2.5 Serving ingredients

One instruction is used at the end of a recipe on every remaining ingredient. It represents the declaration that the ingredient should be served. After an ingredient is served, it cannot be used again.

- `serve(ingredient ing, string manner)`
  Represents serving an ingredient. In a valid recipe, this should be the final instruction. `ing` is deleted. `manner` can be null. For example:

  ```
  create_ing(ing0, "store-bought pie")
  serve(ing0, "with whipped cream")
  ```

2.2.6 Other actions

Three actions do not change the state of the world, but we feel that they represent important parts of a recipe. We have included them in case future users would like to expand by including a more detailed notion of time, state of an ingredient, or state of a tool.

- `set(tool t, string setting)`
  Represents setting a tool on a given setting, or altering a tool in any way. The state of the world is not changed because (in most recipes, at least) tools do not change state. Neither parameter can be null. For example:

  ```
  create_tool(t0, "oven")
  set(t0, "350 degrees")
  ```

  Another example:

  ```
  create_tool(t0, "sauce pan")
  set(t0, "cover")
  ```

- `leave(ingredient ing, string manner)`
  Represents leaving an ingredient alone, usually to cool or to set aside for later use. The state of the world is not changed. `manner` can be null. For example:

  ```
  create_ing(ing0, "hot stew")
  leave(ing0, "let cool")
  ```

- `chefcheck(ingredient ing, string condition)` Represents the chef pausing to check that a condition holds true. The state of the world is not changed. For example, a recipe might say ”saute onions until transparent.” This ”until” condition could be modeled by a `chefcheck`, as shown:

  ```
  create_ing(ing0, "onions")
  create_tool(t0, "saute pan")
  cook(ing0, t0, ing1, "sauteed onions", "saute")
  chefcheck(ing1, "transparent")
  ```

3 CURD: Carnegie Mellon University Recipe Database

We have coordinated an effort by 18 annotators to translate 300 recipes into the MILK language. Annotators used a Java-based GUI tool that we developed.

The data are represented in XML files, with each line containing the original recipe text and the corresponding MILK instruction. If a line corresponds to multiple MILK instructions, it is included as multiple lines.

An additional 350 unannotated recipes are included in the database. As they are assigned to annotators in alphabetical order, the recipes we have annotated come from the first half of the alphabet, so to ensure a random sample, it would be very helpful to have the rest of these recipes finished.

4 Preliminary Work on Semantic Parsing

We began this project with a goal to create a semantic parser for recipes. We have made two attempts so far. We begin with the simple tasks of predicting the sequence of operations to perform, without predicting the arguments to the first-order terms corresponding to those operations.

4.1 Naive Bayes

We first aimed to create a Naive Bayes classifier, which would read in a sentence of text from a recipe and classify, first, how many instructions that line produced, and second, which instructions they are. For example, ”Chop the onions and add them to the stew” would produce two instructions: a `cut` and a `combine`.

4.1.1 Features

We used the following features: occurrence of each word, length of the sentence, and the number of times the word ”and” appears. If we had spent more time on the Naive Bayes model, we might have made a better effort to identify more informative features, but we decided our time was better spent trying a different model.

4.1.2 Results

We used two measures of accuracy: did the parser correctly identify the number of instructions in each recipe sentence, and did it identify the correct series of instructions?
A simple baseline approach for the first measure, labeling every sentence as producing one instruction, would have yielded about 70% accuracy. Unfortunately, our model did not surpass that measure. We only achieved about 65% accuracy in identifying the number of instructions, and about 40% accuracy in identifying the series of instructions. When we removed the ”create_ing” steps (which could easily be done by parsing the ingredient list separately from the instructions) accuracy dropped to about 30% and 5%, respectively.

4.2 Weighted FST

Because of the poor results with a simple Naive Bayes model, we moved on to a more complex model, implemented with a weighted finite state transducer. We aimed to translate words into B-I-O tags to mark the beginning and end of instructions. So, to use our earlier example, ”cut the onions and add them to the stew” would become ”Bcut Icut Icut O Bcombine Icombine Icombine Icombine Icombine”.

4.2.1 The FST’s

This FST model consists of a start (or ”O”) state and a state for each action (”CUT”, ”COMBINE”, etc). The ”inputs” to the FST model the B-I-O sequence (for example, there is an edge from O to CUT with input string ”Bcut”, but no edge from CUT to COMBINE with input string ”Icombine”). The outputs from the

FST model the word sequence; thus, for every legal transition in the B-I-O model, there is an edge for every word seen.

To train, we compose this FST with one for the B-I-O input sequence for a particular sentence and one for the actual sequence of words. We then use the forward-backward algorithm to compute expected counts for all edges in this FST (which correspond to edges in the original FST). Repeat for every training sentence, add the counts together, and renormalize.

We have used the Carmel FST tools to start creating this model, but we have not yet completed an experiment, because of time constraints.

5 Acknowledgments

The recipe annotation effort was supported in part by an undergraduate honors research grant from Boeing and an NSF REU grant. We would like to thank Cinar Sahin for the raw recipe data; Cari Sisson for help testing the annotator tool; our 18 annotators; and Jerry Zhu, Rebecca Hwa, Ric Crabbe, Scott Fahlman, Kevin Gimpel, Shay Cohen, and Dipanjan Das for helpful conversations.