The Tosser: Language Specification

Anna Owens and Emma Neil

Fall 2022

1 Introduction

We recently came across Joshua McFadden and Martha Holmberg's award-winning veggie-focused cookbook entitled *Six Seasons: A New Way with Vegetables*. The book divides the year into 6 growing seasons (three of which are in the summer). Each section features recipes that use vegetables and fresh herbs exclusively from that season. McFadden's motivation was founded on several guiding principles: first, vegetables take on a more complex and sweeter flavor, and simply better taste at the peak of their growing season. Second, vegetables taste best alongside other vegetables from the same season. Third, it is possible for beginner chefs to learn how to cook to the seasons.

We employ the tenets of the *Six Seasons* cooking philosophy as inspiration for a programming language, grounded in the idea that beginner chefs do not necessarily have to learn how to cook for the seasons through trial and error. Instead, they could write a simple program to generate a scrumptious, seasonal recipe that fits their needs. In the final version of our language, we hope to offer the possibility for our users to enter information about the season, dish type, and ingredients to include or exclude from their recipe.

Our project stands out from other sources of recipes in a few important ways. First, and most obviously, it is opinionated in the style of recipes that it generates, choosing exclusively seasonal (to New England) vegetables and combining flavors according to the recipes in *Six Seasons*. Second, it offers the possibility of generating several recipes in a row, if the first recipe isn't to a user's taste.

2 Design Principles

Our language was designed with the following principles in mind:

Simplicity It is easy to use because it requires so little input. It also produces output that is simple to understand because it takes the form of a list of ingredients which is familiar to anyone who has seen a recipe.
Support for abstraction Users have to know remarkably little about the rules that make the recipe generator function. In particular, they do not need to know which ingredients are seasonal or which combination would taste best. All they need to know is the season and whether they want a warm salad/grain bowl or a cold salad.
Maintainability and Growth We chose to create general types (detailed below) that were flexible enough to enable us to add particular ingredients as we expand the types of dishes a user can program. This flexibility is further achieved by the choice to input new ingredients via a CSV file rather than hardcode them.
Regularity Our parser expects inputs in a very specific order (attribute, season, dish type, restrictions) and recognizes a bounded number of input options for each "slot" in the program. While specific, it is not complicated and allows users to quickly learn and efficiently generate recipes without errors.

3 Examples

Example 1
Input:

```
    dotnet run spring salad without nuts, cheese
```

Output:

```
Recipe will include one spring green, 2 spring vegetables, and a dressing.
One example output could be:

    Here is your recipe for: spring salad without nuts, cheese
    Your recipe is:
    0.250 Bunches of Spinach
    1.000 Artichoke
    4.000 Ounces of Sugar Snap Peas
    0.125 Cups of Citrus Vinaigrette
```

Example 2
Input:

```
    dotnet run warm fall salad without (broccoli) and with (chickpeas)
```

Output:

```
Recipe will include one grain, 1 fall vegetables (none of which are broccoli), 1 fall legum
a dressing.  One example output could be:

    Here is your recipe for: warm fall salad without (broccoli) and with (chickpeas)
    Your recipe is:
    0.500 Cups of Quinoa
    4.000 Ounces of Carrot
    8.000 Ounces of Chickpeas
    0.500 Cups of White Onion
    0.500 Cups of Ricotta
    0.500 Cups of Roasted Almonds
    0.500 Tablespoons of Fresh Sage
    0.500 Tablespoons of Fresh Oregano
    0.125 Cups of Pine Nut Vinaigrette
```

Example 3
Input:

```
    dotnet run summer salad
```

Output:

```
Recipe will include one summer green, 2 summer vegetables, a nut, a cheese and
a dressing.  One example output could be:

    Here is your recipe for: summer salad
    Your recipe is:
    0.250 Bunches of Spinach
    1.000 Bell Pepper
    4.000 Ounces of Broccoli
    0.500 Cups of Ricotta
    0.500 Cups of Roasted Peanuts
    0.125 Cups of Pancetta Vinaigrette
```

4 Language Concepts

The core concepts a user needs to understand our language is a basic knowledge of seasons and ingredients. Our language produces recipes for dishes based on seasons, which dictate which vegetables and fruits are best, and dish type, currently only salad. There are four basic season: fall, winter, spring, and summer. In theory, the user would use the current season. Since the recipe generator works for all seasons, and because modern grocery stores generally provide many fruits and vegetables year round, a user could theoretically input any season.

The second concept a user needs to understand is a dish type. Currently, our dish type is a salad, but the language is built to be able to expand this to later include soups, entrees, and side dishes. A user should have a basic understanding of these dish types, as well as a basic understanding of how to put together salads, as the program does not have recipe instructions. However, even if a user does not have any cooking experience, it would only hinder the food they make, not their ability to use the program. Further implementation will have us adding more elements/ingredients to each dish type, more attributes other than temperature to describe dishes, and more dish types.

With seasons and dishes being our key ideas (primitives), combining them is easy. The user must simply pick the dish they would like, and input their season.

5 Grammar

```
<expr> := <Attribute><ws><Dish>
<Attribute> := <Temperature> | (<Attribute>, <Attribute>) | <NoAttribute>
<ws> := " "
<Temperature> := "warm"<ws>  | "cold"<ws>
<NoAttribute> := ""
<Dish> := (<Season>, <DishType>, <Exception>)
<Season> := "fall"<ws> | "winter"<ws> | "spring"<ws> | "summer"<ws>
<DishType> := "salad"<ws> | "salad"
<Exception> := SoftCore(<Flag>, <Name>) | HardCore(<SoftCore>, <SoftCore>)
| <NoException>
<NoException> := ""
<Flag> := "with"<ws> | "without"<ws>
<Name> := <Category> | <Word> | <NoName> | <Name>(<Comma><ws><Name>)+
<Word> := <Letter>+
<Letter> := 'a' | 'b' | ... | 'z'
<Comma> := ','
<NoName> := ""
<Category> := "greens" | "vegetables" | "fruit" | "cheese" | "nuts" | "dressing"
| "spice" | "legumes"
```

6 Semantics

Our programming language is designed to generate seasonal recipes with New England fruits and vegetables. The minimal version parses a simple string that users can enter via the command line. It expects a single input consisting of (at minimum) a season and a dish type, and outputs a list of ingredients for a dish of the correct type that belongs to that season. The semantics are trivially simple because it only expects these two inputs. In order to accomplish this parsing and evaluation, we define a set of primitive types and a set of types that combine those primitives.

6.1 Primitives

Category describes the relevant food group to which an Ingredient belongs. Each ingredient belongs to one of the following categories: Green, Vegetable, Fruit, Cheese, Nut, Spice, Legumes, Onion, Herb, and Dressing.

Season refers to the season of an Ingredient or recipe. The options (unsurprisingly) are: Fall, Winter, Summer, and Spring.

Dish Type refers to the dish category of a recipe. Currently, users can only choose to generate a Salad dish type.

Temperature describes the temperature of a dish. There are two constructors for this type: **Warm** or **Cold**.

6.2 Abstractions

We use higher level types in order to combine our primitives in ways that feel meaningful as well as abstract away some of the complexity of the language to make it more user friendly.

Attribute describes a variation on the expected dish type. Users may enter one or more attributes. Currently, the only attribute associated with a salad is Temperature (Warm or Cold).

Exception refers to a combination of ingredients or categories to include or exclude from a recipe. Exception types are made up of either a single exception (a **SoftCore** type), or multiple exceptions (a **HardCore** type, which accepts two SoftCores). Each SoftCore accepts a flag (**Include** or **Exclude**), and a **Name**, which is either a StrName (string) or Category.

Dish consists of a Season, Dish Type, and Exception.

Recipe is the highest-level type in our programming language. It combines an Attribute and a Dish.

7 Remaining work

Our language is built for expansion, so we have many different directions we could go with this program. We have defined expansion in two different ways: first, additions to the program that fit with our current AST (without adding new types), and second, changes to the program that expand our AST.

For the first category, we could define a set of ingredient rules (i.e. flavor combinations that work well together), so that instead of our recipe generator being so random, it actually produces combinations that would taste good. However, it is worth noting that one of our key principles is that in-season ingredients taste better together, so our current recipe generations are not entirely random. In order to add more dish types, we will also need to continue adding ingredients to the CSV to broaden our recipe options. With more ingredient options, we could expand our definition of warm/cold salads to have more ingredients.

Changes that would expand our AST start with including new dish types such as soup, entree, or side dishes. We would have to define exactly what these mean, which is difficult because in the regular world, many different dishes fit under the category of entree or side dish. We also may divide our summer season into three categories: early summer, mid summer, and late summer, based on the cookbook we are using for inspiration. Because summer is such a good season for crops, and because these crops differ across the different stages of summer, this would allow for more specific seasonal dishes. We also are looking into adding more attributes other than temperature so that users could get more specific forms of dishes. An example of this would be ”roasted”, or even, a specific serving size.