A Wine Pairing Recommendation System

Kia Javanmardian

6.871

May 12, 2005

Table of Contents

Introduction:

Different wines go well with different kinds of food. Many times, however, an individual may be going to a restaurant and not know the ingredients in his meal. Furthermore, he may not know enough about wine to know what to look for when he goes out to eat. This scenario provides an ideal situation for a knowledge based system.

The system presented in this paper pairs wine with a meal. It encapsulates knowledge about food and wine pairings, and it uses additional kinds of knowledge to narrow down food options and make an educated wine recommendation. Overall, the system considers 7 wine possibilities: Merlot, Pinot Noir, Chardonnay, Sauvignon Blanc, Riesling, Cabernet, and Zinfandel. The system also has knowledge pertinent to eight different ethnicities: American, French, Italian, Caribbean, Indian, Middle Eastern, Mexican, and Chinese.

The following subsections will delve into the overall design of our system. They will highlight initial thoughts regarding the overall design of the system and delve into the knowledge encapsulated by the system. Included will be an example of the system in action. The finals sections will focus on what was learned, as well as difficulties uncovered through designing and implementing the system.

Initial Thoughts/Overview:

Initially, a wine pairing system seemed like an ideal domain for a rule based system. When considering a wine, a user takes into account four main factors: kind of meat, kind of sauce, herbs/spices, and a user's personal preferences (dry, sweet). A very simple system could ask the values for each of these categories and match it with an appropriate wine. While such a system would encapsulate knowledge about food and wine pairings, the system would be relatively unknowledgeable. Matching a set ingredients and preferences with different wines uses a shallow and fairly uninteresting set of inferences.

Our system does not ask the ingredients of a person's meal. Instead, it infers the ingredients of a meal through a set of fairly straightforward questions. The inputs to the system are:

- Ethnicity of the restaurant
- Current Season
- User Traits (i.e. health conscious, lactose intolerant)
- User Preference(i.e likes spicy foods, likes savory foods)

The system uses these inputs to infer meal ingredients, and it uses the ingredients to make a wine recommendation. The wine recommendation is a type of wine, but it does not include a year or place of origin.

Figure 1 gives a concrete example of a potential system trace. The trace may not be the exact order in which the final system asks questions, but it should give a fairly accurate example of the system's overall operation.

```
>>System:What kind of meail will you be eating.
<<User:                 Dinner

>>System:  What ethnicity of Restaurant will you be going
to?
<<User:                 Italian

>>System:  Are you health conscious?
<<User:                 No.

>>System:  Are you lactose intolerant?
<<User:                 Yes.

>>System:  Do you like savory foods?
<<User:                 Yes.

>>System:  Are you going on a date?
<<User:                 No.

>>System:  Do you prefer sweetened iced tea over unsweetended iced
tea?
<<User:                 Yes.

>>System:  Be on the lookout for a good Merlot.
```

Figure 1: An example trace of the system.

Knowledge Base/Problem Solving Paradigm Overview

Our system implements a rule based system, and it utilizes JOSHUA. As mentioned before, the initial wine and food pairing problem lends itself well to a rule based system. On a simple level, simple rules regarding meat, sauce type, herbs/spices, and user preferences are fairly straightforward to construct. Our system uses these four categories of rules, but also incorporates another layer of rules that infer the meat, sauce type, and herbs/spices.

Ideally, the system would have incorporated rules and frames. The rule part of the system would use user input to infer ingredients of the meal. The frame portion would then take the ingredients and match it to a number of predetermined frames. Each wine would have one or more frame representations with different slot values representing different ingredients. Frames would be ideal for matching, because, conceptually frames do a better job of capturing ingredient to wine relationships. Rules can be written to perform this task, but the many permutations for ingredient to wine matches become an issue. This is discussed in further detail in the problems section.

The main rules of the system are split up into 5 categories:

- Rules inferring the kind of meat
- Rules inferring the kind of sauce
- Rules inferring the herbs used
- Rules inferring user preferences
- Rules matching ingredients with a particular wine

Detailed Example

The trace given in the previous section asks the ethnicity of the restaurant. After getting the ethnicity of the meal, the system asks whether the user will be having dessert. If the user is having dessert, the system defaults to the only dessert wine, Riesling. If the user is not having a dessert the system assumes he is having a main meal. When inferring the kind of meat, the ethnicity gives the initial set of possible meats. For example, Indian food does not allow for beef, while Italian food, in our system, ignores pork, because beef is the predominant meat. (A very sweeping generalization, a discussion of assumptions is discussed later) After the initial set of meats is inferred from the ethnicity, a number of other questions help to narrow the set of possibilities down to one meat. The system accomplishes this by asking a number of user specific questions.

There are a number of questions the system uses to narrow down the list of possible meats. If we assume the ethnicity is Italian (consistent with figure 1), the possible meat options are seafood/poultry and beef. If a user is health conscious, he will probably prefer a lean meat, poultry/seafood, over beef. To pick between poultry and seafood, the system will ask the user whether he lives near the coast. If he does, the system assumes that seafood is fresh and that the user will choose seafood over poultry. If he does not live on the coast, the system assumes that the user will choose poultry. In Figure 1, the user said he was not health conscious. Therefore, the user assumes that he will be eating beef. Figure 2 illustrates the thought process that goes into inferring the meat.

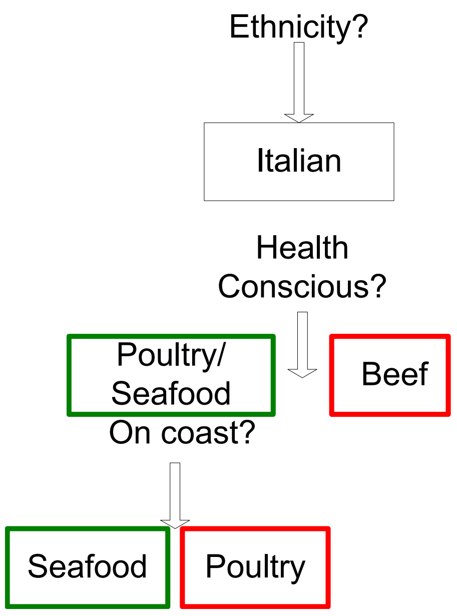

Figure 2: In the context of Italian, the rule linkage diagram for inferring the correct meat.

Inferring the sauce base is similar to inferring the meat. Ethnicity establishes the initial sauce possibilities. For example, Italian food incorporates both tomato and cream based sauces. American food can use BBQ sauce, and Indian food can incorporate a yogurt/cream sauce. User characteristic and other questions narrow down the sauce possibilities. Italian food will be used once again to illustrate the sauce inference process.

The system assumes that Italian food uses either a cream or tomato based sauce. To narrow it down to one sauce base, the system asks whether the user is lactose intolerant. If the user is lactose intolerant, the system will assume that he is having a tomato based sauce. If he is not lactose intolerant, the system will ask whether it is summer. If it is summer, the system assumed a tomato based sauce, since tomatoes will be in season. If it isn't summer the system assumes a cream based sauce. The user input in Figure 1 indicates a lactose intolerant individual. As a result, the system will assume a tomato based sauce. Figure 3 illustrates the reasoning used to infer the sauce base.

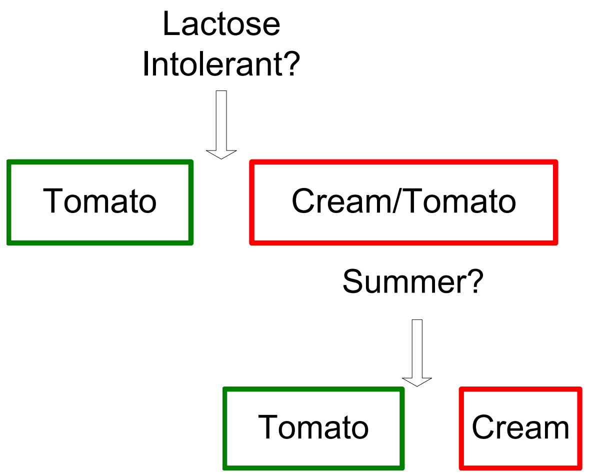

Figure 3: The rule linkage diagram for selecting the sauce base (in Italian context).

Ethnicity also plays a role in inferring herbs and spices. Ethnicity gives the most prominent herbs and spices used in a particular ethnicity of cooking. For example, the system assumes the prominent herbs in Italian cooking are basil and oregano. For Mexican food, the system assumes chili, and for Indian food the system assumes curry. The system includes spices and herbs based on a user's taste preferences. It does not distinguish between herbs in a specific ethnic group. It merely infers whether the ethnically derived herbs will play a prominent role in the user's meal. Italian food will illustrate this concept.

The system assumes that Italian cooking using predominantly uses basil and oregano. To decide whether or not the user's meal will contain these herbs, the system asks whether the user enjoys savory foods. If the user enjoys savory foods, the system will assume the inclusion of basil and oregano. If the user does not enjoy savory foods, the system does not include basil and oregano in the ingredients of the meal. The system will also ask the user whether he is going on a date. If he is, the system will assume the use of garlic, since Italian food uses a lot of garlic. If he isn't going on a date, garlic will not be included. The user input in Figure 1 indicates the user enjoys savory foods and is not going on a date. The system uses this input to infer basil, oregano, and garlic will likely be used in the user's meal. Figure 4 demonstrates the inference problem involved in determining herbs and spices.

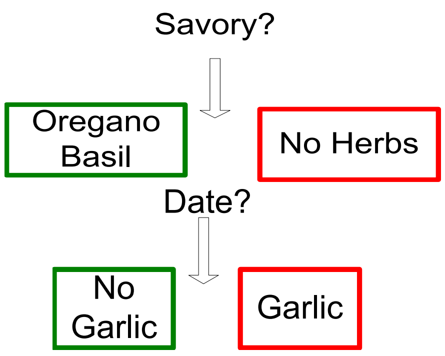

Figure 4 In the Italian domain, the process by which herbs and spices are determined.

A user's personal tastes also play a role in the wine pairing process. The system focuses on whether the user would most likely enjoy a sweet or dry wine. While there are a number of characteristics that describe a given wine, the system focused on sweet vs. dry. Focusing on more characteristics would have greatly increased the overall complexity of the rules.

In order to determine whether the user would prefer a sweet or dry wine, the system asks the user whether he prefers sweetened or unsweetened iced tea. Sweetened iced tea indicates a preference for sweeter wine, while a response of unsweetened iced tea indicates a preference for a dryer wine. In our example, the user has a preference fo sweeter wine. Figure 5 illustrates the process by which the system picks between a sweeter or dryer preference.

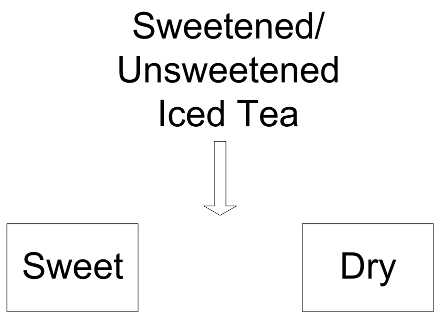

Figure 5: The system chooses between preferences for dry or sweet wine.

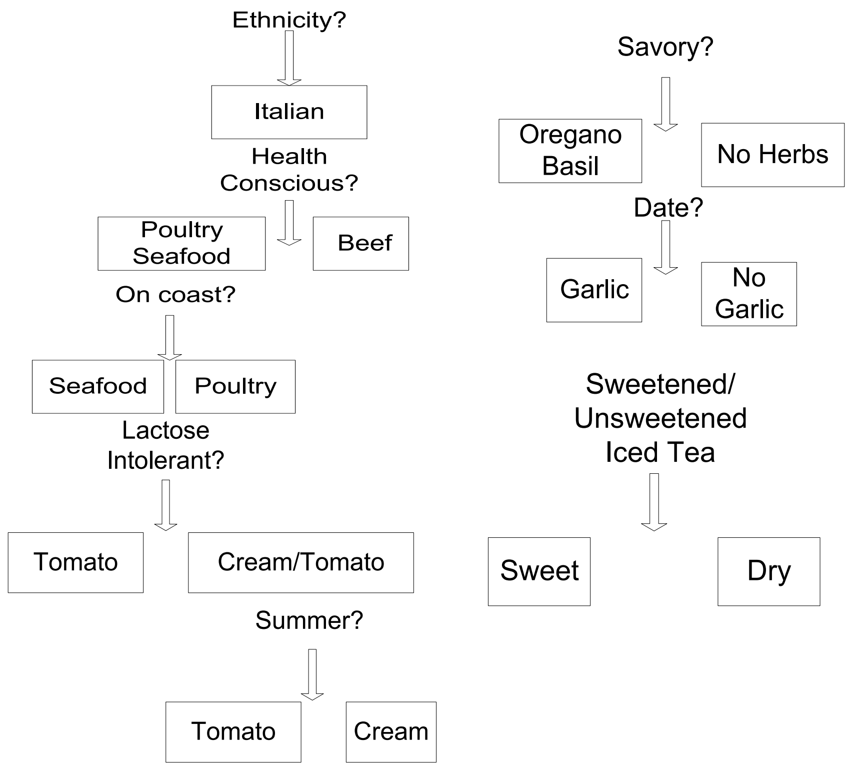

Figure 6: The overall knowledge base for the Italian restaurant example.

In our example, the inferred meat was beef. The inferred sauce was tomato based. The herbs and spices used were oregano, basil, and garlic. The system also inferred that the user had a preference for a sweeter kind of wine. Matching these ingredients and preferences to a particular wine is the next step. The ingredients and preferences inferred in this example match well with a merlot. To make this match, the system has a rule matching beef, tomatoes, basil, oregano, garlic, and sweet wine with a merlot.

Backward Chaining

The previous section outlined the main knowledge base of the system. Overall, there were 5 main groups of knowledge. The example that was presented, however, left out some important characteristics of our system. Most importantly, the example does not illustrate the manner in which the system uses backward chaining rules. While the system does use some forward chaining rules, the overall design and implementation relied upon backward chaining.

The system starts with the rules relating food ingredients to a particular wine. The ingredient-to-wine rules have most priority, and will fire before any of the other rules. The system tries to match the antecedents of these rules by firing different rules governing the meat, sauce, herbs, etc. Almost every ingredient to wine rule has some reference to a particular meat. Therefore, rules governing meat inference will almost always be fired.

Backward chaining also makes the system more flexible. When constructing the system, the biggest challenge was inferring the ingredients of a user's meal. We tried to extract as much information as possible by asking a number of different questions. Some inferred ingredients, however, were not present in any of the ingredient to wine pairing rules. We decided to keep these irrelevant rules, because a more thorough ingredient to wine knowledge base may incorporate some of the extraneously inferred ingredients. Garlic for example, was initially not even considered in ingredient to wine rules. It was later added to some of these rules as we increased our ingredient to wine knowledge base. Since irrelevant rules are not fired in a backward chaining system, irrelevant rules that may become relevant can be captured without any penalty to the system's overall functionality.

Weights

Weights are not incorporated into level of the backward chaining process. All rules assume a certainty of one, unless they are ingredient to wine pairing rules. This makes a very sweeping assumption. However, the mechanism by which JOSHUA used certainty factors made it difficult to correctly assign weights to each rule. Instead, the wine pairing rules are given a weight based on the criteria that made the rule fire. For example, there are a number of rules that result in a Merlot recommendation. Some of these rules have more conditions than other, and therefore, receive a higher certainty factor in our system. This situation is further discussed in the *JOSHUA Difficulties* section and is illustrated by Figures 8 and 9 of this document.

Domain Specific Questions

The system asks domain specific questions. The example did not illustrate this point, but it is an important aspect of the system. Once an ethnicity is selected, the system knows what questions are relevant. For example, if the ethnicity is Italian, the system will not ask the user whether he likes spicy food. Similarly, if the ethnicity is Chinese, the system will not ask the user if he is lactose intolerant. The system's rules are constructed in a manner that avoids irrelevant questions.

Once the ethnicity is selected rules unrelated to that ethnicity will not be fired by the system. Many of the rules specific to a particular ethnicity are written in an ethnicity specific manner. A rule governing Indian food illustrates this idea. If the ethnicity is Indian and the user likes spicy food, the system infers the use of curry. If the ethnicity is Italian, this rule will not be fired, because it requires the ethnicity be Indian. By specifying the preconditions necessary for the rule to be relevant, the system avoids asking irrelevant questions.

Avoiding irrelevant questions is also the result of backward chaining. If, for example, the only wine that considers garlic has already been ruled out, the system will not fire rules inferring garlic use. This is a direct result of backward chaining. Since none of the wines being considered contain a reference to garlic, no rules will be fired that make conclusions regarding garlic use. A forward chaining rule would automatically fire all the rules whenever the antecedents were satisfied—whether or not the rule was relevant. (Assuming ethnicity was the first question being asked)

System Responses to "I don't Know"

The system deals with scenarios in which the user does not know the answer to a particular question. To cope with this challenge, the system incorporates another layer of knowledge. For example, a user may not know whether he likes savory food. The system tries to solve this problem by asking a series of questions. From these questions the system will try to infer whether or not the user enjoys savory foods. For the example of savory foods, the system will ask the questions shown in Figure 7.

```
>>System:      Do you like savory foods?
<<User:        I don't know

>>System:      Do you have an herb garden?
<<User:        No.

>>System:      Do you have over 10 spices in your spice cabinet?
<<User:        No.

>>System:      Do you like steak sauce?
<<User:        No.
```

Figure 7 The system asks the user a number of questions to infer whether or not he likes savory foods.

Since the user does not have an herb garden, ten spices in his spice cabinet, and does not like steak sauce, the system concludes that the user does not like savory foods. Dealing with "I don't know" requires additional knowledge. Under a specific ethnicity, the system knows the ingredients implicated by savory food, and it also knows criteria that imply a user's affinity for savory flavors. Rules determining a user's like or dislike of savory foods enables the system to perform and additional layer of inference.

Additional rules are in place for determining if a user is on the coast, health conscious, has exotic tastes, is lactose intolerance, or if a restaurant is expensive. To determine if a user is on the coast, the system may ask whether or not he frequently goes boating or to the beach. To determine whether the user is health conscious, the system may ask whether the user has been dieting or exercises regularly. For exotic tastes, the system may ask a user whether he goes hunting or eats lots of ethnic foods. A complete listing of the different "I don't know" rules can be found in the appendix of this document.

JOSHUA Difficulties

A number of difficulties arose during the construction of this system. Many of the difficulties were JOSHUA specific, although some were a result of the overall scope and design of the system.

JOSHUA had a tendency to misfire rules. One reason came from a bug in the JOSHUA system. JOSHUA incorrectly evaluates all the predicates of an "or" statement even if a predicate is already found to be true. Another source of this problem involved forward chaining rules. Some of our system's rules were forward chaining and triggered whenever the antecedents of the rule were satisfied. This was done mainly for efficiency sake. For example, when the user specifies the ethnicity of his meal, the system automatically fires a number of rules that narrow down the possible meats, spices, sauces, etc. However, if a rule was currently being fired and backward chaining, any forward chaining rules that were fired during this process did not register its conclusions with the system. The system only became aware of the results once the rule that had initiated the backward chaining process had exited.

One of the pinot noir rules illustrates this point nicely. In this particular rule the system tries to find out if chicken is part of the meal. In doing so the system asks the user if he is health conscious, and the user says no. The system then realizes that the predominant meat in the dish is beef. The meat being beef fires a forward chaining rule that says the current meat is not chicken. However, the system does not recognize that this rule fired until after the pinot noir rule is done executing. Therefore, even though the system determines the meat is beef, it ends up asking whether the user is eating chicken. To combat this problem, we implemented a number of backward chaining rules that would rule out different meats, once one had been selected. Doing this, however, exposed another problem with JOSHUA.

JOSHUA did not allow backward chaining rules to make more than one conclusion at a time. Therefore, a rule with more than one conclusion had to be written as a set of multiple rules, each having only one conclusion. This wasn't a major problem, but it was fairly cumbersome.

Permutation Problem

Another major difficulty involved the different permutations of the ingredient to wine rules. A Merlot, for example, could be well suited to a particular combination of meat, sauce, and spices. However, not having a particular match with the sauce or spices does not mean that a Merlot would not go well with the meal. The system had to account for this scenario by implementing looser rules that would also recommend a Merlot. It had to know that a Merlot would go well with beef but with less certainty than a meal containing beef, oregano, and tomatoes. This idea was touched upon earlier. Figure 8 partially illustrates this idea.

```
(defrule riesling-1 (:backward :certainty 1.0 :importance 92)
      if    [and  [or [poultry ?user yes]
                        [shellfish ?user yes]]
                        [has-cheese ?user yes]
                        [sauce-type ?user light]
                        [or   [uses-dill ?user yes]
                              [uses-sage ?user sage]
                              [uses-clove ?user clove]
                              [uses-ginger ?user ginger]]
                        [prefers-sweet-wines ?user yes]]
            then [wine-to-drink ?user riesling])
```

Figure 8: An ingredient to wine pairing rule that returns Riesling.

The rule above requires the meal to have poultry or shellfish, cheese, dill or sage or clove or ginger. The user must also prefer a sweeter wine over a dry wine. Another rule for Riesling is shown if Figure 9.

```
(defrule riesling-1 (:backward :certainty .8 :importance 92)
      if    [and  [or [poultry ?user yes]
                        [shellfish ?user yes]]
                        [has-cheese ?user yes]
                        [sauce-type ?user light]
                        [or   [uses-dill ?user yes]
                              [uses-sage ?user sage]
                              [uses-clove ?user clove]
                              [uses-ginger ?user ginger]]]
            then [wine-to-drink ?user riesling])
```

Figure 9: A less stringent rule that pairs Riesling with meal ingredients

This rule is a less stringent version of the rule in Figure 8. The system accounts for this difference by assigning less of a certainty factor to the rule, but a problem clearly emerges. Different permutations of a single rule have to be implemented to properly represent wine pairing possibilities. A more complex the knowledge base between ingredients and wines, leads to more possible permutations. This was one of the reasons why herbs were clumped together in a single or statement. Differentiating between them would have required many more permutations of a single rule. If a rule has *n* preconditions, for example, the permutations of the different preconditions are 2^n. This was one of the major challenges that we faced during implementation. Furthermore, this led us to believe that a frame based system would better capture the relationships between different ingredients-to-wine pairings. This idea is further discussed in the *Conclusions* section.

What Went Well/What didn't go well

Overall, the system was able to come up with an educated suggestion for wine, based on a users input. While our approach was fairly simple, we met our objectives within the scope of our project. Namely, we were able to make a wine recommendation without having the user tell us about the ingredients found in his meal. On the flipside, our recommendations were based on a number of sweeping generalizations. As a result, our system is not able to give very exact recommendations. It is possible that the suggested wine will not match a user's meal. That is ok, however, and is further addressed in the *Conclusions* section.

If the user answered yes to questions that inferred new ingredients, the system did a fairly good job of coming up with a wine recommendation. If the user answered no to such questions, the system had a hard time coming up with a good recommendation, because it was unable to infer more ingredients. For example, answering no to spicy food tells the system that curry or chili (depending on domain) is not used, but it does give any idea of what is used.

The nature of our system also made very big assumptions, and it didn't allow for much variety. For example, it doesn't allow for savory Indian food. By limiting the scope of the kinds of foods within an ethnic domain, the system was able to use the ethnicity to make inferences. If the system were actually true to the variety of foods in an ethnic group, it would be unable to use ethnicity as a way of differentiating between different food types. Mexican food, within the context of our system, illustrates this idea. Mexican food is known for being spicy, having cheese, etc. However, many Mexican dishes are savory and use seafood. Had the system accounted for this, the ethnicity, Mexican, would have told us very little about a meal in relation to other ethnicities. Our system is built upon the stereotype of Mexican food being spicy, using pork or beef, and using a cheese. Although inaccurate, without making this assumption the system would be at a loss when it comes to inferring meal properties from an ethnicity.

Conclusions

While our system can not always deliver very exact recommendations, the system was good at making a recommendation based on the information it is given. Further, this brings up an important point regarding knowledge based systems, in general. The recommendation of a knowledge based system is only as good as the information that it is given. Since the system tries to infer ingredients from external user input, the recommendations it gives are not as good as those given by a system which asks the actual ingredients. If the main goal had been providing an accurate wine recommendation, a database matching ingredients to wines may have done a better job. However, a database would not have nearly as much "knowledge" as our system, and it would infer almost nothing.

We also learned about the role of scope in designing a knowledge based system. Whenever a system infers something from user inputs, it is generally required to make assumptions. In the case of our system, these assumptions dealt with generalizations regarding ethnicities. These assumptions included Mexican food being predominantly spicy rather than savory. This also included the assumption that savory Italian food implies the user of oregano and basil. While these assumptions are not accurate representations of the world, they are necessary for our system to make inferences from answers to very general question.

We also came to the conclusion that a frame based system would have worked better for the portion of the system that matched ingredients with a wine. While rules did a good job of coming up with the ingredients, rules had a hard time dealing with the different permutations of ingredients to a specific wine. A frame system designed for ingredient to wine matches may be better suited for the task.

If a frame system could differentiate between different ingredients by assigning weights to them, a match to a particular wine could be reached by adding the weights of the different matching factors. In this way, more matching factors would give a higher score. JOSHUA's certainty procedure did not really allow us to assign different weights to different factors. Our system mimicked the process of adding different weights, by assigning a higher certainty to a rule that had more conditions. For example, the rule in Figure 8 had a higher certainty than the one in Figure 9. The more conditions a rule had, the greater the match to a wine, and therefore, the rule got a higher score.

Overall, designing this system was a very useful exercise and brought to light many of the concepts discussed in class. It demonstrated how a knowledge based system can actually encapsulate knowledge and make inferences. It also showed the limitations of knowledge based systems and the assumptions that must be made when designing them.