University of Camerino

Science and Technologies

Master Degree in Computer Science (LM-18 Class)
Knowledge Engineering and Business Intelligence

Personalized Wine Menu

Students
Piermichele Rosati
Student ID 124176

Marco Serenelli
Student ID 124081

Supervisors
Prof. Knut Hinkelmann

Prof. Holger Wache

A.Y. 2022/2023

Contents

1 Introduction **5**
1.1 Domain 5
1.2 Our Menu and Wine Pairing 6

2 Decision Model and Notation **11**
2.1 Camunda 11
2.2 Our Decision Model 11
2.3 Input and Output Data 12
2.4 Decision Tables and FEEL Expressions 13
2.5 Input/Output Example 23

3 Rule-Based Systems **25**
3.1 Prolog 25
3.2 Implementation 26
3.3 Query Example 38

4 Ontology Engineering **39**
4.1 Protégé 39
4.2 Our Ontology 40

5 Graphical Modelling Language **49**
5.1 Adoxx 49
5.2 Wine Decision Implementation 49

6 Testing **55**

7 Conclusions **65**
7.1 Piermichele Rosati's Conclusion 65
7.2 Marco Serenelli's Conclusion 70

1. Introduction

1.1 Domain

Nowadays, technology is constantly advancing and many sectors in the market world are adapting to it, trying to make the most of it to increase their business. If we think about the food service field, almost every restaurant in the world now has a digital menu that can be scanned via QR code. This functionality is very convenient both in terms of cost savings on the printing of a paper menu and in terms of managing any changes to that menu by the restaurant manager. While there are several advantages for operators, there may not be for customers. They scan the menu via QR code but may be equipped with a phone with a small screen size, in which case reading the menu becomes highly inconvenient especially if the menu is extensive with a wine section.

An additional problem to this concerns the possibility of expressing preferences by users who might be interested in a specific grape or a specific quality of wine, e.g. a user might prefer only white wines and therefore a system to find all dishes paired with this set of wines would be very useful.

In this context, the meal is a decisive factor. In particular, red wines go well with meat and white wines with fish, but we may also have exceptions such as white wines with chicken or rabbit.

The objective of this project is to define and build a knowledge base on wines and to implement a system capable of selecting wines to match the selected meal and possibly the preferences expressed by the user.

In order to achieve our goal we did a lot of research on wines in order to understand how best to pair them with specific meals. This way, we have defined our knowledge base and have defined different knowledge-based solutions using *Camunda*, *Prolog* and *Protégé* to help with the wine selection task. And lastly, we have created a graphical representation using *ADOxx*.

Following, in Chapter 2 we will discuss the decision model and notation, and how we have modelled our system using Camunda.

In Chapter 3 we will show a different approach using, rule-based systems, specifically, Prolog.

While in Chapter 4 we will explain how we have created our model via the use of ontology engineering in Protégé.

Then Chapter 5 we will go over our graphical representation of the model using ADOxx.

In Chapter 6 we will show various testing examples that we have used to test our solutions.

Finally Chapter 7 contains the conclusion of Piermichele Rosati and Marco Serenelli respectively.

1.2 Our Menu and Wine Pairing

Our menu includes 18 wines, 3 wine colors (red, white, rosé), 9 wine categories (i.e. 2 wines in each category) and 6 meals. Since wine is made by one or more grapes and in a particular region and state, we decided to consider these aspects for user preferences, along with dryness, taste and the tannin level. In an effort to simplify our knowledge base, we intentionally omitted certain factors related to wines, such as ignoring the alcohol content, in order to minimize the overall complexity.
Regarding how to pair a wine with a meal, we have followed the scheme and suggestions within the "Food and Wine pairing" guide [3].
Each meal consists of a varying number of ingredients and each ingredient is associated with a strong and/or weak ideal wine category. To determine the strong and weak ideal wine categories for each meal, we selected the strong and weak ideal wines for each ingredient within that meal and found their intersection. The resulting intersection represents the weak or strong ideal wine category for the meal. In the case where one ingredient had a specific wine category classified as strong and another ingredient had the same specific wine category classified as weak, we have determined that their intersection would yield a strong ideal wine category. For simplicity, we decided to represent also the preparation of a plate such as grilled or fried as an ingredient.

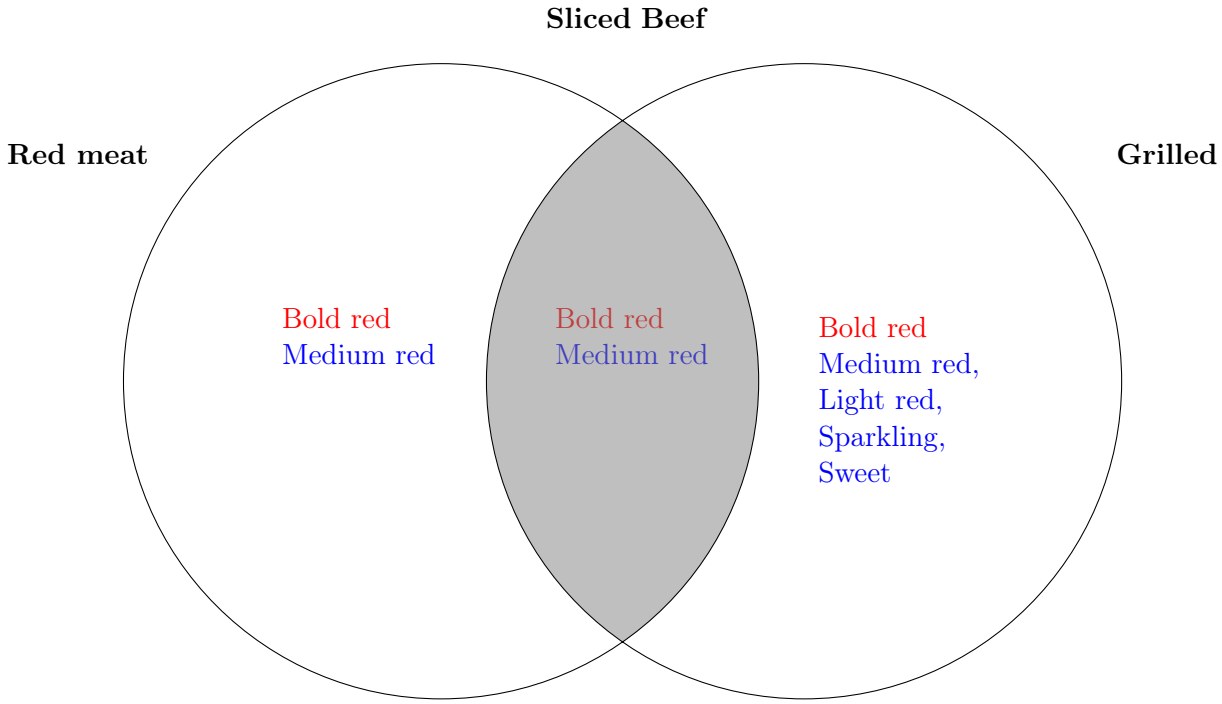


Figure 1.1: Intersection between the ideal wine categories of ingredients Red Meat and Grilled, which are present in the Sliced Beef Meal. In red are shown the strong ideal wine categories and in blue the weaks

In the following tables, firstly we will illustrate the ingredients that we used in our knowledge-based along with their Strong and Weak ideal wine categories, and then the meals with their respective ingredients together with their already-computed strong and weak ideal wine categories.

Ingredient	**Strong Ideal Category**	**Weak Ideal Category**
Red Meat	Bold red	Medium red
Grilled	Bold red	Medium red, Light red, Sparkling, Sweet White
Fish	Light white	Rich white, Sparkling
Herbs	Light white	Rich white, Rose, Light red, Medium Red
Saulted or fried	Light red	Rose, Rich white, Light white, Sparkling
White starches	-	Bold red, Medium red, Light red, Rose, Rich white, Light white, Sweet white, Dessert
Root vegetables & squash	Rose	Rich white, Sweet white
Soft cheese & cream	Light red, Rich white	Medium red, Rose, Light white, Sparkling, Dessert, Sweet white
Poultry	Light red, Rich white	Medium red, Rose, Light white, Sparkling
Roasted	Bold red	Medium red, Light red, Rose
Pork	Medium red	Sparkling, Rose, Bold red
Black pepper	Bold red	Medium red
Alliums	Medium red	Bold red, Light red, Rose, Rich white, Light white, Sparkling, Sweet white
Red pepper	Medium red	Bold red, Rose, Light white, Sparkling, Sweet white
Grilled	Bold red	Medium red, Light red, Sparkling, Sweet white
Smoked	Medium red	Bold red, Light red, Rose, Sparkling, Dessert
Fruit and berries	Sweet white	Sparkling, Dessert

Table 1.1: Ingredients with their own strong and weak ideal wine categories

Meal	**Ingredient**	**Ideal Category**
Sliced Beef	Red Meat, Grilled	Strong: Bold Red Weak: Medium red
Fried bass with herbs	Fish, Herbs, Saulted or fried	Strong: Light white Weak: Rich white
Pumpkin risotto	White starches, Root vegetables & squash, Soft cheese & cream	Strong: Rich white, Rose Weak: Sweet
Roast chicken with herbs	Poultry, Herbs, Roasted	Strong: Light red Weak: Medium red, Rose
Pulled pork	Pork, Black pepper, White starches, Alliums, Red pepper, Grilled, Smoked	Strong: Medium red, Bold red
Strawberry Cheesecake	Fruit and berries	Strong: Sweet White Weak: Sparkling, Dessert

Table 1.2: Meal with ingredients and their strong and weak ideal wine categories

The following tables illustrate our wine knowledge base, each table represents a category of wines with their grape, region and country along with its taste.

Wine	**Grapes**	**Region (Country)**	**Taste**
Brunello di Montalcino	Brunello, Sangiovese	Tuscany (Italy)	Bold, Tannic, Dry
Juline Châteauneuf-du-Pape	Grenache, Syrah	Vaucluse (France)	Bold, Tannic, Dry

Table 1.3: Bold Red Wines

Wine	**Grapes**	**Region (Country)**	**Taste**
Case Basse Sangiovese	Sangiovese	Tuscany (Italy)	Bold, Tannic, Dry
Pomerol Château de Salesl	Cabernet Franc, Cabernet Sauvignon, Merlot	Bordeaux (France)	Bold, Tannic, Dry

Table 1.4: Medium Red Wines

Wine	**Grapes**	**Region (Country)**	**Taste**
Aragone	Carignan	Aragona (Spain)	Bold, Tannic, Dry
Romanée-Saint	Pinot Noir	Vosne-Romanée (France)	Bold, Tannic, Dry

Table 1.5: Light Red Wines

Wine	**Grapes**	**Region (Country)**	**Taste**
Garrus Rosé	Vermentino	Chateau d'Esclans (France)	Light, Tannic, Dry
Marsannay	Chardonnay	Burgundy (France)	Light, Tannic, Not Dry

Table 1.6: Rosé Wines

Wine	**Grapes**	**Region (Country)**	**Taste**
Corton-Charlemagne Grand Cru	Chardonnay	Burgundy (France)	Light, Tannic, Dry
Hermitage AOC	Marsanne	Rhône (France)	Bold, Tannic, Dry

Table 1.7: Rich White Wines

Wine	**Grapes**	**Region (Country)**	**Taste**
Sauternes	Chardonnay	Bordeaux (France)	Bold, Less Tannic, Not Dry
Trebbiano d'Abruzzo	Trebbiano	Abruzzi (Italy)	Bold, Less Tannic, Dry

Table 1.8: Light White Wines

Wine	**Grapes**	**Region (Country)**	**Taste**
Coteau de la Beylesse	Marsanne	Rhône (France)	Bold, Tannic, Dry
Cremant d'Alsace	Crémant	Anjou-Saumur (France)	Bold, Less Tannic, Dry

Table 1.9: Sparkling Wines

Wine	**Grapes**	**Region (Country)**	**Taste**
G-Max Reisling	Riesling	Rheinhessen (Germany)	Bold, Less Tannic, Dry
Vajra Moscato d'Asti	Moscato	Piemonte (Italy)	Light, Less Tannic, Dry

Table 1.10: Sweet Wines

Wine	**Grapes**	**Region (Country)**	**Taste**
Fino	Palomino	Andalusia (Spain)	Bold, Less Tannic, Dry
Occhio di Pernice	Sangiovese, Syrah	Tuscany (Italy)	Light, Tannic, Not Dry

Table 1.11: Dessert Wines

2. Decision Model and Notation

In business analysis, the **Decision Model and Notation (DMN)** [6] is a standard published by the **Object Management Group** [4]. This approach is used in business for describing and modelling repeatable decisions within companies in such a way that decision models can be interchangeable between them. This standard allows the company to define and create a modelling notation in order to take decisions that will support decision management and business rules.

DMN Elements

The DMN standard consists of the following four elements:

- **Decision Requirements Diagrams**: these diagrams are used to show how the elements of decision-making are linked into a dependency network;
- **Decision tables** [7]: are a concise visual representation for specifying which actions to perform depending on given conditions;
- **Business Context**: for the decision of the company;
- **Friendly Enough Expression Language (FEEL)**: is a language used to evaluate expressions in decision tables.

2.1 Camunda

Camunda [2] is an open-source platform designed in Java, that enables workflow and process automation for the management of companies of any size. It is used for creating decision models, more in general to create BPMN process diagrams and DMN decision tables. Moreover, it provides REST APIs for developers who do not use Java but at the same time want to use Camunda's functionality. We used Camunda in order to define our knowledge base about wines as well as to define decision-making processes and rules that align with our objectives.

2.2 Our Decision Model

As we can see from Figure 2.1, our model's decision-making process starts by taking the meal as the only required input, along with any additional user preferences regarding the wine, which may include or exclude color, dryness, tannin level, taste, grape, country, and region.

Once the input data is obtained, the decision tables will apply decision logic on this input to compute the output: the list of wines associated with the meal given as input with the preferences expressed by the user.

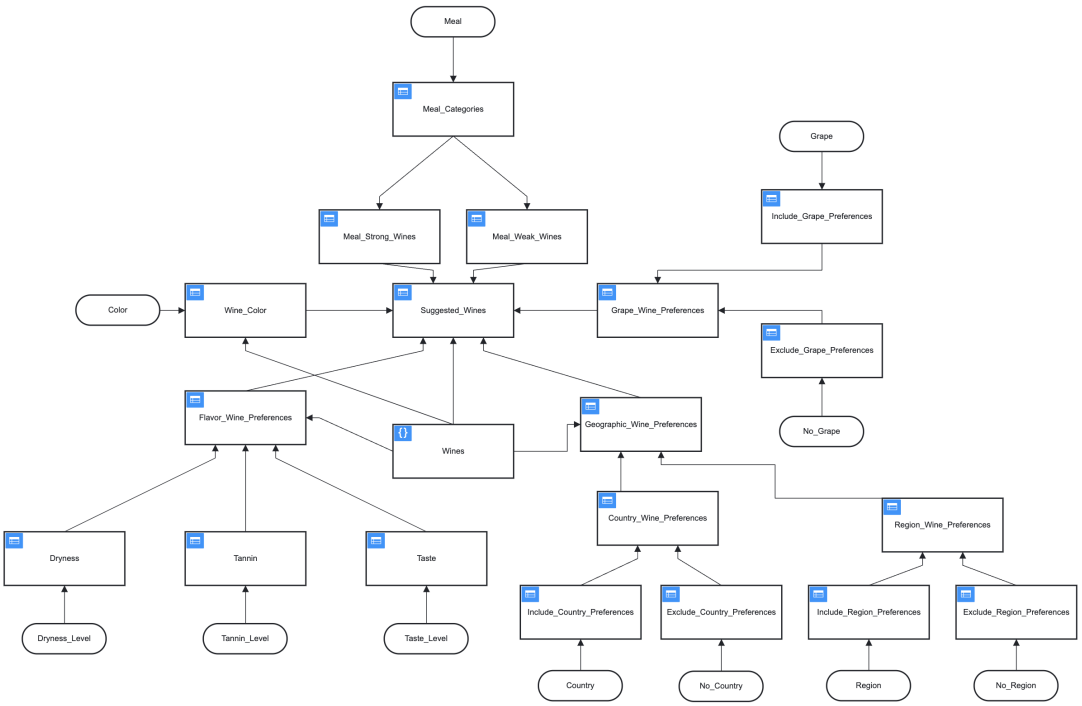


Figure 2.1: Our Model in Camunda

To define our knowledge base we have used several decision tables. Going into more detail, our initial approach involved specifying the knowledge base through a few tables, each containing a substantial number of inputs and a large number of rows. However, this led to maintainability and management problems with the tables due to their size. To address this, we opted to divide the responsibility for each input context by creating additional tables, effectively splitting the responsibility of each input into separate tables.

2.3 Input and Output Data

Referring to Figure 2.1, we can see that we have the following tables taking a single input data:

- *Wine_ Color*: takes as input the **Color**, i.e. whether the user wants a red (or not red), white (or not white) or rosé (or not rosé) wine and outputs the **wine list with the specified wine color**;
- *Dryness*: takes as input the **Dryness_Level**, i.e. whether the user wants a dry wine or not and outputs the **wine list with the specified dryness**;
- *Tannin*: takes as input the **Tannin_Level**, i.e. whether the user wants a tannic or less tannic wine and outputs the **wine list with the specified tannin level**;

- *Taste*: takes as input the **Taste_Level**, i.e. whether the user wants a bold or light wine and outputs the **wine list with the specified taste level**;

- *Include_Country_Preferences*: takes as input the **Country** which represents the state of wine production to be included and outputs the **wine list associated with the specified country**;

- *Exclude_Country_Preferences*: takes as input the **No_Country** which represents the state of wine production to be excluded and outputs the **wine list associated with the specified country**;

- *Include_Region_Preferences*: takes as input the **Region**, which represents region of wine production to be included and outputs the **wine list associated with the specified region**;

- *Exclude_Region_Preferences*: takes as input the **No_Region**, which represents region of wine production to be excluded and outputs the **wine list not associated with the specified region**;

- *Meal_Categories*: takes as input the **Meal**, which is one of the meals on the menu and produces 3 outputs: the **list of ingredients** that that meal consist of, the list of wine categories matched to that specific meal with a **strong recommendation** and the list of wine categories matched to that specific meal with a **weak recommendation**;

- *Meal_Strong_Wines*: takes as input the **list of strong wine categories matched to that specific meal** and outputs the **list of wines associated with the strong categories** contained therein;

- *Meal_Weak_Wines*: takes as input the **list of weak wine categories matched to that specific meal** and outputs the **list of wines associated with the weak categories** contained therein;

- *Include_Grape_Preferences*: takes as input the **Grape**, which represents the grape of wine production to be included and outputs the **wine list composed of the specified grape**;

- *Exclude_Grape_Preferences*: takes as input the **No_Grape**, which represents the grape of wine production to be excluded and outputs the **wine list not composed of the specified grape**;

2.4 Decision Tables and FEEL Expressions

To simplify our process, we have developed a literal expression that encapsulates the wine list in the form of a string.

Figure 2.2: Wines Literal Expression

For the following tables, what we associated each row with the corresponding wine. As we can see, in Figure 2.3, depending on the user's preference about the wine dryness (**dry** or **not dry**), there are all the rows of dry wines and non-dry wines.

Dryness — Hit Policy: Collect

	When	Then
	Dryness_Level "dry","not dry"	Dryness_Wines "Brunello di Montalcino","Juline...
1	"dry"	"Brunello di Montalcino"
2	"dry"	"Juline Châteauneuf-du-Pape"
3	"dry"	"Pomerol Château de Salesl"
4	"dry"	"Case Basse Sangiovese"
5	"dry"	"Romanée-Saint"
6	"dry"	"Aragone"
7	"dry"	"Garrus Rosè"
8	"dry"	"Coteau de la Beylesse"
9	"dry"	"Hermitage AOC"
10	"dry"	"Trebbiano d'Abruzzo"
11	"dry"	"Cremant d'Alsace"
12	"dry"	"Coteau de la Beylesse"
13	"dry"	"G-Max Reisling"
14	"dry"	"Vajra Mosacto d'Asti"
15	"dry"	"Fino"
16	"not dry"	"Marsannay"
17	"not dry"	"Sauternes"
18	"not dry"	"Occhio di pernice"
19	""	""

Figure 2.3: Dryness Dec. Table

The same reasoning was applied for the tables Tannin (**tannic** or **less tannic**) and Taste (**bold** or **light**):

Tannin — Hit Policy: Collect

	When: Tannin_Level ("tannic","less tannic")	Then: Tannin_Level_Wines ("Brunello di Montalcino","Juline...)
1	"tannic"	"Brunello di Montalcino"
2	"tannic"	"Juline Châteauneuf-du-Pape"
3	"tannic"	"Pomerol Château de Salesl"
4	"tannic"	"Case Basse Sangiovese"
5	"tannic"	"Romanée-Saint"
6	"tannic"	"Aragone"
7	"tannic"	"Marsannay"
8	"tannic"	"Garrus Rosè"
9	"tannic"	"Corton-Charlemagne Grand Cru"
10	"tannic"	"Hermitage AOC"
11	"tannic"	"Coteau de la Beylesse"
12	"tannic"	"Occhio di pernice"
13	"less tannic"	"Sauternes"
14	"less tannic"	"Trebbiano d'Abruzzo"
15	"less tannic"	"Cremant d'Alsace"
16	"less tannic"	"G-Max Reisling"
17	"less tannic"	"Vajra Mosacto d'Asti"
18	"less tannic"	"Fino"
19	""	""

Figure 2.4: Tannin Dec. Table

Taste — Hit Policy: Collect

	When: Taste_Level ("bold","light"	Then: Taste_Wines ("Brunello di Montalcino","Juline...
1	"bold"	"Brunello di Montalcino"
2	"bold"	"Juline Châteauneuf-du-Pape"
3	"bold"	"Pomerol Château de Salesl"
4	"bold"	"Case Basse Sangiovese"
5	"bold"	"Romanée-Saint"
6	"bold"	"Aragone"
7	"bold"	"Hermitage AOC"
8	"bold"	"Sauternes"
9	"bold"	"Trebbiano d'Abruzzo"
10	"bold"	"Cremant d'Alsace"
11	"bold"	"Coteau de la Beylesse"
12	"bold"	"G-Max Reisling"
13	"bold"	"Fino"
14	"light"	"Marsannay"
15	"light"	"Garrus Rosè"
16	"light"	"Corton-Charlemagne Grand Cru"
17	"light"	"Vajra Mosacto d'Asti"
18	"light"	"Occhio di pernice"
19	""	""

Figure 2.5: Taste Dec. Table

For the geographical preference, we defined four countries in our knowledge base: *France*, *Germany*, *Italy* and *Spain*. In this case, we apply the same reasoning as before, associating each country with the respected wine produced there. As the possibility of inclusion and exclusion is granted for the country, the following 2 tables were defined:

Include_Country_Preferences — Hit Policy: Collect

	When: Country ("italy","france","germany","spain"	Then: Include_Country_Preferences ("Brunello di Montalcino","Juline...
1	"italy"	"Brunello di Montalcino"
2	"italy"	"Case Basse Sangiovese"
3	"italy"	"Trebbiano d'Abruzzo"
4	"italy"	"Vajra Mosacto d'Asti"
5	"italy"	"Occhio di pernice"
6	"france"	"Juline Châteauneuf-du-Pape"
7	"france"	"Pomerol Château de Salesl"
8	"france"	"Romanée-Saint"
9	"france"	"Marsannay"
10	"france"	"Garrus Rosè"
11	"france"	"Corton-Charlemagne Grand Cru"
12	"france"	"Hermitage AOC"
13	"france"	"Sauternes"
14	"france"	"Cremant d'Alsace"
15	"france"	"Coteau de la Beylesse"
16	"germany"	"G-Max Reisling"
17	"spain"	"Aragone"
18	"spain"	"Fino"
19	""	""

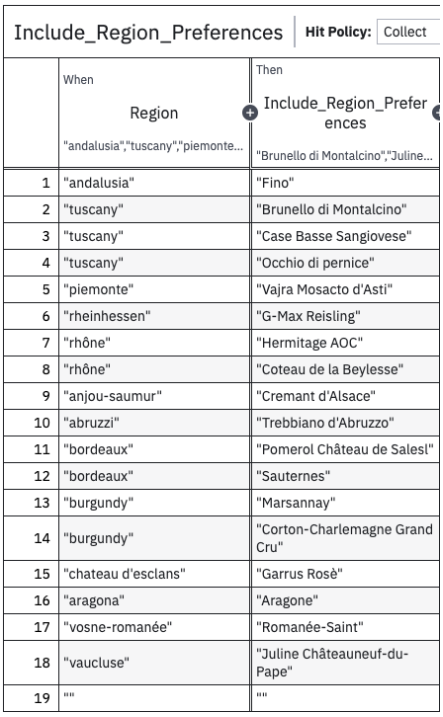
Figure 2.6: Include Country Dec. Table

Exclude_Country_Preferences — Hit Policy: Collect

	When: No_Country ("italy","france","germany","spain"	Then: Exclude_Country_Preferences ("Brunello di Montalcino","Juline...
1	"italy"	"Brunello di Montalcino"
2	"italy"	"Case Basse Sangiovese"
3	"italy"	"Trebbiano d'Abruzzo"
4	"italy"	"Vajra Mosacto d'Asti"
5	"italy"	"Occhio di pernice"
6	"france"	"Juline Châteauneuf-du-Pape"
7	"france"	"Pomerol Château de Salesl"
8	"france"	"Romanée-Saint"
9	"france"	"Marsannay"
10	"france"	"Garrus Rosè"
11	"france"	"Corton-Charlemagne Grand Cru"
12	"france"	"Hermitage AOC"
13	"france"	"Sauternes"
14	"france"	"Cremant d'Alsace"
15	"france"	"Coteau de la Beylesse"
16	"germany"	"G-Max Reisling"
17	"spain"	"Aragone"
18	"spain"	"Fino"
19	""	""

Figure 2.7: Exclude Country Dec. Table

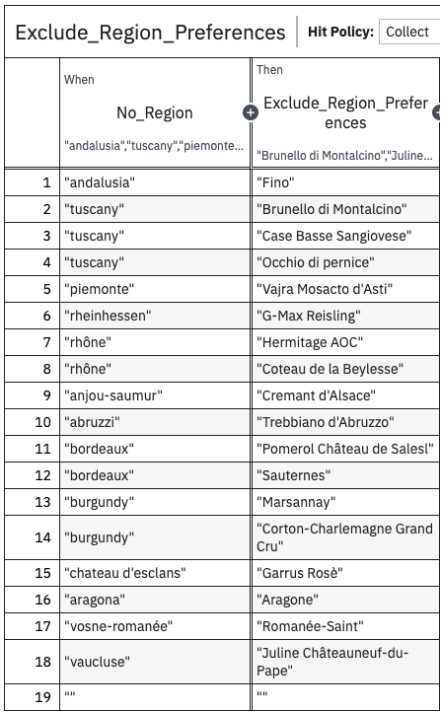
This reasoning was further utilized to include and exclude regions:



Include_Region_Preferences | Hit Policy: Collect

	When	Then
	Region	Include_Region_Preferences
	"andalusia","tuscany","piemonte...	"Brunello di Montalcino","Juline...
1	"andalusia"	"Fino"
2	"tuscany"	"Brunello di Montalcino"
3	"tuscany"	"Case Basse Sangiovese"
4	"tuscany"	"Occhio di pernice"
5	"piemonte"	"Vajra Mosacto d'Asti"
6	"rheinhessen"	"G-Max Reisling"
7	"rhône"	"Hermitage AOC"
8	"rhône"	"Coteau de la Beylesse"
9	"anjou-saumur"	"Cremant d'Alsace"
10	"abruzzi"	"Trebbiano d'Abruzzo"
11	"bordeaux"	"Pomerol Château de Salesl"
12	"bordeaux"	"Sauternes"
13	"burgundy"	"Marsannay"
14	"burgundy"	"Corton-Charlemagne Grand Cru"
15	"chateau d'esclans"	"Garrus Rosè"
16	"aragona"	"Aragone"
17	"vosne-romanée"	"Romanée-Saint"
18	"vaucluse"	"Juline Châteauneuf-du-Pape"
19	""	""

Figure 2.8: Include Region Dec. Table



Exclude_Region_Preferences | Hit Policy: Collect

	When	Then
	No_Region	Exclude_Region_Preferences
	"andalusia","tuscany","piemonte...	"Brunello di Montalcino","Juline...
1	"andalusia"	"Fino"
2	"tuscany"	"Brunello di Montalcino"
3	"tuscany"	"Case Basse Sangiovese"
4	"tuscany"	"Occhio di pernice"
5	"piemonte"	"Vajra Mosacto d'Asti"
6	"rheinhessen"	"G-Max Reisling"
7	"rhône"	"Hermitage AOC"
8	"rhône"	"Coteau de la Beylesse"
9	"anjou-saumur"	"Cremant d'Alsace"
10	"abruzzi"	"Trebbiano d'Abruzzo"
11	"bordeaux"	"Pomerol Château de Salesl"
12	"bordeaux"	"Sauternes"
13	"burgundy"	"Marsannay"
14	"burgundy"	"Corton-Charlemagne Grand Cru"
15	"chateau d'esclans"	"Garrus Rosè"
16	"aragona"	"Aragone"
17	"vosne-romanée"	"Romanée-Saint"
18	"vaucluse"	"Juline Châteauneuf-du-Pape"
19	""	""

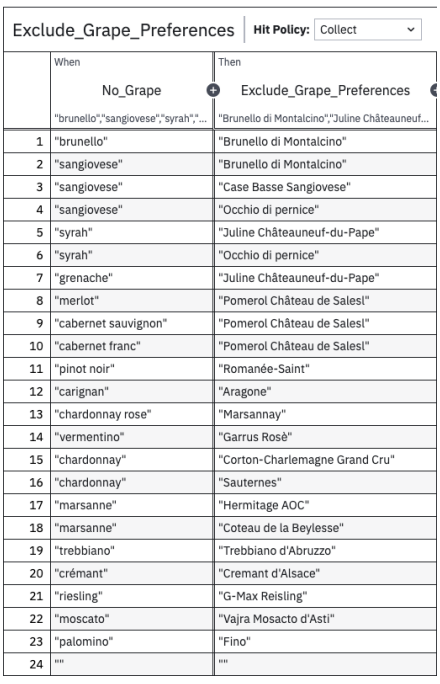
Figure 2.9: Exclude Region Dec. Table

The same logic for the inclusion and exclusion was also applied to the preference of grapes. The only difference is that, as wine can be made from one or more grapes, in our knowledge base, we have associated a grape with multiple wines.

Include_Grape_Preferences | Hit Policy: Collect

	When	Then
	Grape	Include_Grape_Preferences
	"brunello","sangiovese","syrah","...	"Brunello di Montalcino","Juline Châteauneuf...
1	"brunello"	"Brunello di Montalcino"
2	"sangiovese"	"Brunello di Montalcino"
3	"sangiovese"	"Case Basse Sangiovese"
4	"sangiovese"	"Occhio di pernice"
5	"syrah"	"Juline Châteauneuf-du-Pape"
6	"syrah"	"Occhio di pernice"
7	"grenache"	"Juline Châteauneuf-du-Pape"
8	"merlot"	"Pomerol Château de Salesl"
9	"cabernet sauvignon"	"Pomerol Château de Salesl"
10	"cabernet franc"	"Pomerol Château de Salesl"
11	"pinot noir"	"Romanée-Saint"
12	"carignan"	"Aragone"
13	"chardonnay rose"	"Marsannay"
14	"vermentino"	"Garrus Rosè"
15	"chardonnay"	"Corton-Charlemagne Grand Cru"
16	"chardonnay"	"Sauternes"
17	"marsanne"	"Hermitage AOC"
18	"marsanne"	"Coteau de la Beylesse"
19	"trebbiano"	"Trebbiano d'Abruzzo"
20	"crémant"	"Cremant d'Alsace"
21	"riesling"	"G-Max Reisling"
22	"moscato"	"Vajra Mosacto d'Asti"
23	"palomino"	"Fino"
24	""	""

Figure 2.10: Include Grape Dec. Table



Exclude_Grape_Preferences | Hit Policy: Collect

	When	Then
	No_Grape	Exclude_Grape_Preferences
	"brunello","sangiovese","syrah","...	"Brunello di Montalcino","Juline Châteauneuf...
1	"brunello"	"Brunello di Montalcino"
2	"sangiovese"	"Brunello di Montalcino"
3	"sangiovese"	"Case Basse Sangiovese"
4	"sangiovese"	"Occhio di pernice"
5	"syrah"	"Juline Châteauneuf-du-Pape"
6	"syrah"	"Occhio di pernice"
7	"grenache"	"Juline Châteauneuf-du-Pape"
8	"merlot"	"Pomerol Château de Salesl"
9	"cabernet sauvignon"	"Pomerol Château de Salesl"
10	"cabernet franc"	"Pomerol Château de Salesl"
11	"pinot noir"	"Romanée-Saint"
12	"carignan"	"Aragone"
13	"chardonnay rose"	"Marsannay"
14	"vermentino"	"Garrus Rosè"
15	"chardonnay"	"Corton-Charlemagne Grand Cru"
16	"chardonnay"	"Sauternes"
17	"marsanne"	"Hermitage AOC"
18	"marsanne"	"Coteau de la Beylesse"
19	"trebbiano"	"Trebbiano d'Abruzzo"
20	"crémant"	"Cremant d'Alsace"
21	"riesling"	"G-Max Reisling"
22	"moscato"	"Vajra Mosacto d'Asti"
23	"palomino"	"Fino"
24	""	""

Figure 2.11: Exclude Grape Dec. Table

Regarding the color of the wine, since there are three colors in our knowledge base: *red*, *white* and *rosé*, we have associated each color with the list of wines of that color. Moreover, we allow to negate the input taking "not" and the color that user wants to exclude, for example "not red". In this decision table, the **"Collect"** hit policy was selected, since the output will be different for each color but in the case of negation more than one row could be selected as output.

Wine_Color | **Hit Policy:** Collect

	When	Then
	Color "red","not red","white","not white","rose","not rose"	Wine_Color
1	"red", "not white", "not rose"	["Brunello di Montalcino","Juline Châteauneuf-du-Pape", "Pomerol Château de Salesl", "Case Basse Sangiovese", "Romanée-Saint", "Aragone", "Occhio di pernice"]
2	"rose", "not red", "not white"	["Marsannay", "Garrus Rosè"]
3	"white", "not red", "not rose"	["Corton-Charlemagne Grand Cru", "Hermitage AOC", "Sauternes", "Trebbiano d'Abruzzo", "Cremant d'Alsace", "Coteau de la Beylesse", "G-Max Reisling", "Vajra Mosacto d'Asti", "Fino"]
4	""	Wines

Figure 2.12: Wine Color Dec. Table

For meals we reasoned differently. Since a dish is made up of ingredients, when combined they give the best category of wine for the given dish. In our knowledge base, we have pre-computed the best wine category for each meal, by following the approach explained in Section 1.2. The *Meal_ Categories* table (Fig. 2.13) associates a list of strong and weak wine categories recommendations with each meal and, unlike all the other tables in the model it produces **3 outputs**: the list of ingredients for that meal, a list of strongly recommended wines and a list of weakly recommended wines. Here, the **"Unique"** hit policy is used, since the output will be different for every meal.

Meal_Categories | **Hit Policy:** Unique

	When	Then	And	And
	Meal "sliced beef","pumpkin risotto","...	Ingredients	Strong_Wine_Category	Weak_Wine_Category
1	"sliced beef"	["red meat", "grilled"]	["bold red"]	["medium red"]
2	"fried bass with herbs"	["fish", "herbs", "saulted_or_fried"]	["light white"]	["rich white"]
3	"pumpkin risotto"	["white_starches", "root_vegetables_and_squash", "soft_cheese_and_cream"]	["rich white","rose"]	["sweet white"]
4	"roast chicken with herbs"	["poultry", "herbs", "roasted"]	["light red"]	["medium red","rose"]
5	"pulled pork"	["pork", "black_pepper", "white_starches", "alliums", "red_pepper", "grilled", "smoked"]	["bold red","medium red"]	[]
6	"strawberry cheesecake"	["fruit_and_berries"]	["sweet white"]	["sparkling","dessert"]

Figure 2.13: Meal Categories Dec. Table

We can see that in every illustrated table, except for *Meal_ Categories* (Fig. 2.13), the **"Collect"** hit policy was selected, since the expected output in all these decision tables is the list of wines associated with the specified preference.

In all the other decision tables, we have utilized **FEEL expressions**, which were very useful for drastically reducing the number of rows on each table, and for obtaining exceptional results by leveraging on the language's constructs.

Meal_Strong_Wines and Meal_Weak_Wines

In the tables *Meal_ Strong_ Wines* and *Meal_ Weak_ Wines* we used the function **list contains(list, element)** which returns true if the given list contains the element, false otherwise. This function was very useful for filtering the list of wine categories taken as **input** (which is also the double output of *Meal_ Categories*).

On both of these tables, each wine category is associated with the corresponding wine, and the **output** of these tables is the list of wines with a strong recommendation for *Meal_ Strong_ Wines* and the list of wines with a weak recommendation for *Meal_ Weak_ Wines*.

Meal_Strong_Wines | **Hit Policy:** Collect

	When	Then
	Strong_Wine_Category	Meal_Strong_Wines
		"Brunello di Montalcino","Juline...
1	list contains(Strong_Wine_Category, "bold red")	"Brunello di Montalcino"
2	list contains(Strong_Wine_Category, "bold red")	"Juline Châteauneuf-du-Pape"
3	list contains(Strong_Wine_Category, "medium red")	"Pomerol Château de Salesl"
4	list contains(Strong_Wine_Category, "medium red")	"Case Basse Sangiovese"
5	list contains(Strong_Wine_Category, "light red")	"Romanée-Saint"
6	list contains(Strong_Wine_Category, "light red")	"Aragone"
7	list contains(Strong_Wine_Category, "rose")	"Marsannay"
8	list contains(Strong_Wine_Category, "rose")	"Garrus Rosè"
9	list contains(Strong_Wine_Category, "rich white")	"Corton-Charlemagne Grand Cru"
10	list contains(Strong_Wine_Category, "rich white")	"Hermitage AOC"
11	list contains(Strong_Wine_Category, "light white")	"Sauternes"
12	list contains(Strong_Wine_Category, "light white")	"Trebbiano d'Abruzzo"

Figure 2.14: Strong Wines Dec. Table

Meal_Weak_Wines | **Hit Policy:** Collect

	When	Then
	Weak_Wine_Category	Meal_Weak_Wines
		"Brunello di Montalcino","Juline...
1	list contains(Weak_Wine_Category, "bold red")	"Brunello di Montalcino"
2	list contains(Weak_Wine_Category, "bold red")	"Juline Châteauneuf-du-Pape"
3	list contains(Weak_Wine_Category, "medium red")	"Pomerol Château de Salesl"
4	list contains(Weak_Wine_Category, "medium red")	"Case Basse Sangiovese"
5	list contains(Weak_Wine_Category, "light red")	"Romanée-Saint"
6	list contains(Weak_Wine_Category, "light red")	"Aragone"
7	list contains(Weak_Wine_Category, "rose")	"Marsannay"
8	list contains(Weak_Wine_Category, "rose")	"Garrus Rosè"
9	list contains(Weak_Wine_Category, "rich white")	"Corton-Charlemagne Grand Cru"
10	list contains(Weak_Wine_Category, "rich white")	"Hermitage AOC"
11	list contains(Weak_Wine_Category, "light white")	"Sauternes"
12	list contains(Weak_Wine_Category, "light white")	"Trebbiano d'Abruzzo"

Figure 2.15: Weak Wines Dec. Table

The next tables are the only ones to use the **"Unique"** hit policy, since unlike the others they contain only one row, and therefore although the output is always a list of wines,

this list is produced using the **set intersection rule** between the input lists.

Flavor_Wine_Preferences

Since the user can express preferences or not, e.g. if we focus on wine flavour preferences, the user can select one as all or none. We have defined the intersection of the Dryness list, the Tannin list and the Taste list as the output rule. The wine list, given as the **flavour final output**, will be exactly the intersection of these 3 input lists (Figure 2.16). A special matter to consider is when a list is empty, in this case, the final intersection will not be an empty list but the intersection of that list(s) and the *Wines* list. An empty final intersection is also In the case of a non-existent combination of preferences, there is the possibility that the final intersection is empty (see Chapter 6).

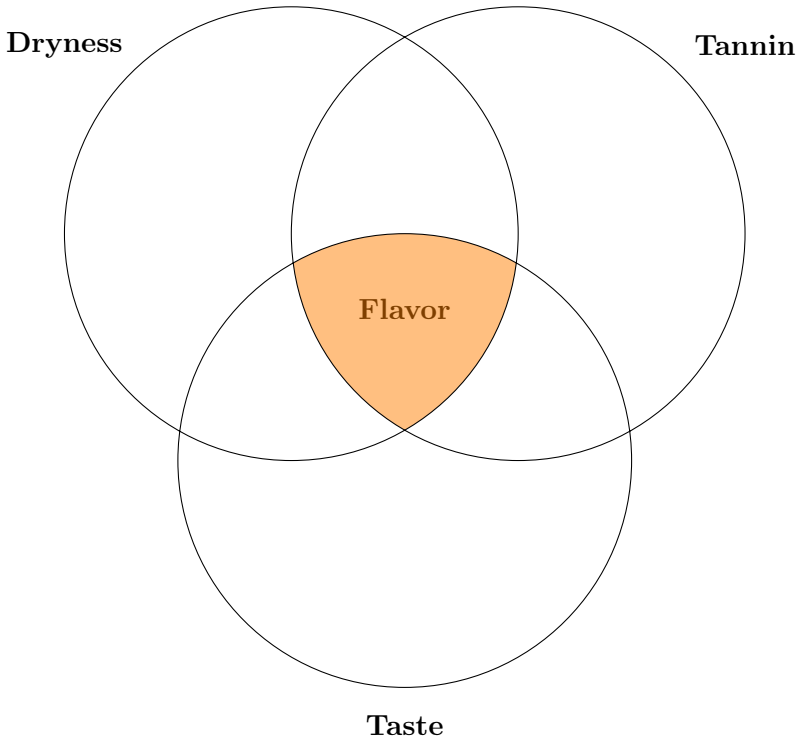


Figure 2.16: Intersection between Dryness, Tannin and Taste input lists

Unfortunately, FEEL does not provide a function that determines common elements between several lists, i.e. an intersection between lists. Because of this, what we did was to define the following FEEL expression:

if(((**count**($Dryness\_Wines$) = 1) **and** ($Dryness\_Wines[1] = ""$)) **and**
((**count**($Tannin\_Level\_Wines$) = 1) **and** ($Tannin\_Level\_Wines[1] = ""$)) **and**
((**count**($Taste\_Wines$) = 1) **and** ($Taste\_Wines[1] = ""$))) **then** $Wines$
else
($Dryness\_Wines$[**list contains**($Wines, item$)][**if**((**count**($Tannin\_Level\_Wines$) =
1) **and** ($Tannin\_Level\_Wines[1] = ""$)) **then list contains**($Wines, item$)
else list contains($Tannin\_Level\_Wines, item$)])
[**if**((**count**($Taste\_Wines$) = 1) **and** ($Taste\_Wines[1] = ""$))
then list contains($Wines, item$) **else list contains**($Taste\_Wines, item$)]

The above may sound like a complicated expression but it is actually simpler than it seems. The reasoning is that if the 3 lists are all **simultaneously** empty (they all contain only the empty string ""), then the entire *Wines list* (Fig. 2.2) is given as output. The empty list is checked with the **count** function, which checks if the size of the list is 1 and then if the first element of it is the empty string "". Otherwise, if the 3 lists are all not empty, a **filter** is applied.

As we have only checked if the lists are all **simultaneously** empty, the **filter** starts by checking if the *Dryness_ Wines list* is not empty, before intersecting the wines contained in it with the ones in *Wines list*. After this step, we also have to check if *Tannin_ Level_ Wines list* is empty before intersecting it with the output produced by the "intermediate" intersection between *Dryness_ Wines list* and *Wines list*. The same reasoning is used for the last list, resulting in the intersection of the three lists.

Flavor_Wine_Preferences | Hit Policy: Unique

	When Dryness_Wines string	And Tannin_Level_Wines string	And Taste_Wines string	Then Flavor_Wine_Preferences
1	-	-	-	if (((count(Dryness_Wines) = 1) and (Dryness_Wines[1] = "")) and ((count(Tannin_Level_Wines) = 1) and (Tannin_Level_Wines[1] = "")) and ((count(Taste_Wines) = 1) and (Taste_Wines[1] = ""))) then Wines else (Dryness_Wines[list contains(Wines,item)][if((count(Tannin_Level_Wines) = 1) and (Tannin_Level_Wines[1] = "")) then list contains(Wines,item) else list contains(Tannin_Level_Wines,item)])[if((count(Taste_Wines) = 1) and (Taste_Wines[1] = "")) then list contains(Wines,item) else list contains(Taste_Wines,item)]

Figure 2.17: Flavor Wine Preferences Dec. Table

Since the expression explained above is excessively long, and hard to read, the alternative we devised and implemented is the following:

Flavor_Wine_Preferences | Hit Policy: Unique

	When Dryness_Wines string	And Tannin_Level_Wines string	And Taste_Wines string	Then Flavor_Wine_Preferences	Annotations
1	(count(Dryness_Wines) >= 1) and (Dryness_Wines[1] != "")	(count(Tannin_Level_Wines) >= 1) and (Tannin_Level_Wines[1] != "")	(count(Taste_Wines) >= 1) and (Taste_Wines[1] != "")	(Wines[list contains(Dryness_Wines,item)][list contains(Tannin_Level_Wines,item)])[list contains(Taste_Wines,item)]	include, include, include
2	(count(Dryness_Wines) >= 1) and (Dryness_Wines[1] != "")	(count(Tannin_Level_Wines) = 1) and (Tannin_Level_Wines[1] = "")	(count(Taste_Wines) >= 1) and (Taste_Wines[1] != "")	Wines[list contains(Dryness_Wines,item)][list contains(Taste_Wines,item)]	include, none, include
3	(count(Dryness_Wines) >= 1) and (Dryness_Wines[1] != "")	(count(Tannin_Level_Wines) >= 1) and (Tannin_Level_Wines[1] != "")	(count(Taste_Wines) = 1) and (Taste_Wines[1] = "")	Wines[list contains(Dryness_Wines,item)][list contains(Tannin_Level_Wines,item)]	include, include, none
4	(count(Dryness_Wines) = 1) and (Dryness_Wines[1] = "")	(count(Tannin_Level_Wines) >= 1) and (Tannin_Level_Wines[1] != "")	(count(Taste_Wines) >= 1) and (Taste_Wines[1] != "")	Wines[list contains(Tannin_Level_Wines,item)][list contains(Taste_Wines,item)]	none, include, include
5	(count(Dryness_Wines) = 1) and (Dryness_Wines[1] = "")	(count(Tannin_Level_Wines) >= 1) and (Tannin_Level_Wines[1] != "")	(count(Taste_Wines) = 1) and (Taste_Wines[1] = "")	Wines[list contains(Tannin_Level_Wines,item)]	none, include, none
6	(count(Dryness_Wines) = 1) and (Dryness_Wines[1] = "")	(count(Tannin_Level_Wines) = 1) and (Tannin_Level_Wines[1] = "")	(count(Taste_Wines) >= 1) and (Taste_Wines[1] != "")	Wines[list contains(Taste_Wines,item)]	none, none, include
7	(count(Dryness_Wines) >= 1) and (Dryness_Wines[1] != "")	(count(Tannin_Level_Wines) = 1) and (Tannin_Level_Wines[1] = "")	(count(Taste_Wines) = 1) and (Taste_Wines[1] = "")	Wines[list contains(Dryness_Wines,item)]	include, none, none
8	(count(Dryness_Wines) = 1) and (Dryness_Wines[1] = "")	(count(Tannin_Level_Wines) = 1) and (Tannin_Level_Wines[1] = "")	(count(Taste_Wines) = 1) and (Taste_Wines[1] = "")	Wines	none, none, none

Figure 2.18: Splitted Flavor Wine Dec. Table

In this case, the approach remains consistent, but it involves considering all possible combinations of the three cases, based on whether the user provides that specific input or not.

Country_Wine_Preferences

The exact same reasoning applied to *Flavor_ Wine_ Preferences* was applied for the inclusion and exclusion of countries considering the 4 cases in the following table:

Country_Wine_Preferences | Hit Policy: Unique

	When: Include_Country_Preferences (string)	And: Exclude_Country_Preferences (string)	Then: Country_Wine_Preferences	Annotations
1	(count(Include_Country_Preferences) = 1) and (Include_Country_Preferences[1] = "")	(count(Exclude_Country_Preferences) = 1) and (Exclude_Country_Preferences[1] = "")	Wines	no include, no exclude
2	(count(Include_Country_Preferences) = 1) and (Include_Country_Preferences[1] = "")	(count(Exclude_Country_Preferences) >= 1) and (Exclude_Country_Preferences[1] != "")	Wines[not (list contains (Exclude_Country_Preferences,item))]	no include, exclude
3	(count(Include_Country_Preferences) >= 1) and (Include_Country_Preferences[1] != "")	(count(Exclude_Country_Preferences) = 1) and (Exclude_Country_Preferences[1] = "")	Wines[list contains(Include_Country_Preferences,item)]	include, no exclude
4	(count(Include_Country_Preferences) >= 1) and (Include_Country_Preferences[1] != "")	(count(Exclude_Country_Preferences) >= 1) and (Exclude_Country_Preferences[1] != "")	Wines[list contains (Include_Country_Preferences,item)] [not(list contains(Exclude_Country_Preferences, item))]	include, exclude

Figure 2.19: Country Wine Dec. Table

The final output for the countries will be the list of wines associated with the countries to be included and those to be excluded.

Region_Wine_Preferences

The same reasoning is applied for the inclusion and exclusion of regions also considering the 4 cases in the following table:

Region_Wine_Preferences | Hit Policy: Unique

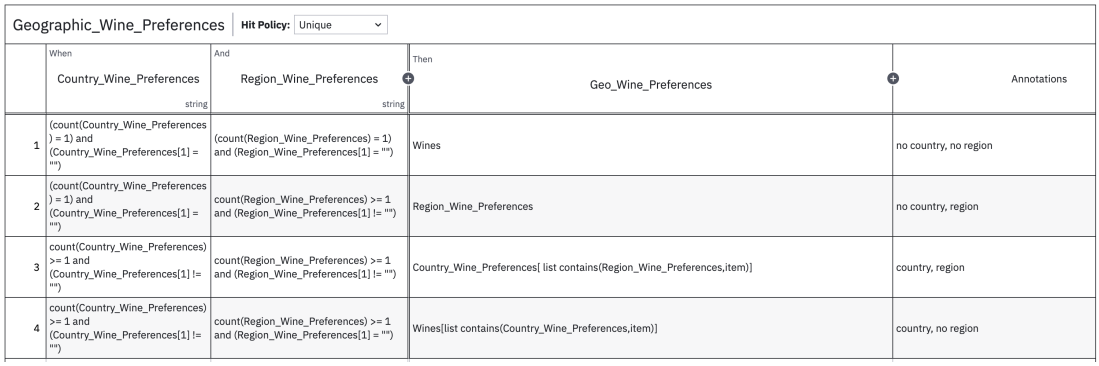
	When: Include_Region_Preferences (string)	And: Exclude_Region_Preferences (string)	Then: Region_Wine_Preferences	Annotations
1	(count(Include_Region_Preferences) = 1) and (Include_Region_Preferences[1] = "")	(count(Exclude_Region_Preferences) = 1) and (Exclude_Region_Preferences[1] = "")	Wines	no include, no exclude
2	(count(Include_Region_Preferences) = 1) and (Include_Region_Preferences[1] = "")	(count(Exclude_Region_Preferences) >= 1) and (Exclude_Region_Preferences[1] != "")	Wines[not(list contains(Exclude_Region_Preferences,item))]	no include, exclude
3	(count(Include_Region_Preferences) >= 1) and (Include_Region_Preferences[1] != "")	(count(Exclude_Region_Preferences) = 1) and (Exclude_Region_Preferences[1] = "")	Wines[list contains(Include_Region_Preferences,item)]	include, no exclude
4	(count(Include_Region_Preferences) >= 1) and (Include_Region_Preferences[1] != "")	(count(Exclude_Region_Preferences) >= 1) and (Exclude_Region_Preferences[1] != "")	Wines[list contains(Include_Region_Preferences,item)][not(list contains(Exclude_Region_Preferences,item))]	include, exclude

Figure 2.20: Region Wine Dec. Table

The final output for the regions will be the list of wines associated with the regions to be included and those to be excluded.

Geographic_Wine_Preferences

For the geographical preferences, we applied the intersection rule between the *Country_ Wine_ Preferences*'s output and the *Region_ Wine_ Preferences*' output.



Geographic_Wine_Preferences | Hit Policy: Unique

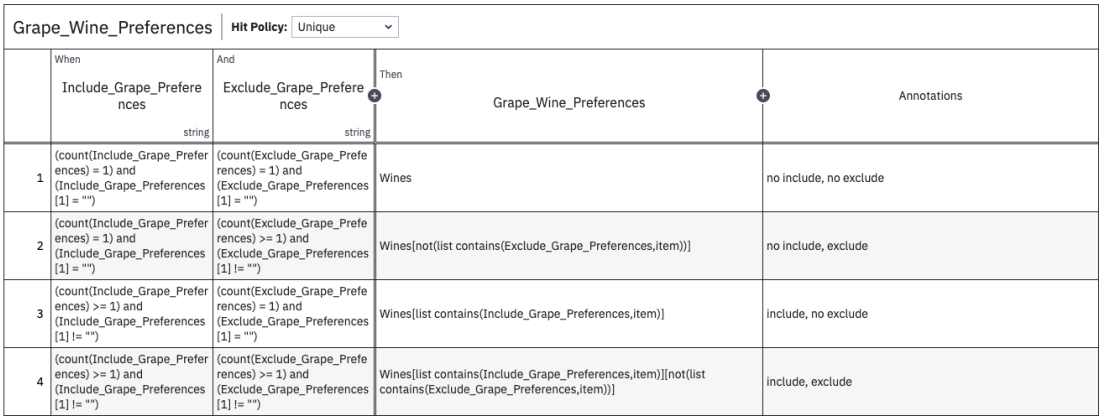
	When: Country_Wine_Preferences (string)	And: Region_Wine_Preferences (string)	Then: Geo_Wine_Preferences	Annotations
1	(count(Country_Wine_Preferences) = 1) and (Country_Wine_Preferences[1] = "")	(count(Region_Wine_Preferences) = 1) and (Region_Wine_Preferences[1] = "")	Wines	no country, no region
2	(count(Country_Wine_Preferences) = 1) and (Country_Wine_Preferences[1] = "")	count(Region_Wine_Preferences) >= 1 and (Region_Wine_Preferences[1] != "")	Region_Wine_Preferences	no country, region
3	count(Country_Wine_Preferences) >= 1 and (Country_Wine_Preferences[1] != "")	count(Region_Wine_Preferences) >= 1 and (Region_Wine_Preferences[1] != "")	Country_Wine_Preferences[list contains(Region_Wine_Preferences,item)]	country, region
4	count(Country_Wine_Preferences) >= 1 and (Country_Wine_Preferences[1] != "")	count(Region_Wine_Preferences) >= 1 and (Region_Wine_Preferences[1] = "")	Wines[list contains(Country_Wine_Preferences,item)]	country, no region

Figure 2.21: Geographic Wine Preferences Dec. Table

In our model, it is not allowed include a region which does not belong to a country but this is allowed for the exclusion.

Grape_Wine_Preferences

Also for the grape preferences, the intersection between the list of wines associated with the grape to be included and the list of wines associated with the grape to be excluded, was applied:



Grape_Wine_Preferences | Hit Policy: Unique

	When: Include_Grape_Preferences (string)	And: Exclude_Grape_Preferences (string)	Then: Grape_Wine_Preferences	Annotations
1	(count(Include_Grape_Preferences) = 1) and (Include_Grape_Preferences[1] = "")	(count(Exclude_Grape_Preferences) = 1) and (Exclude_Grape_Preferences[1] = "")	Wines	no include, no exclude
2	(count(Include_Grape_Preferences) = 1) and (Include_Grape_Preferences[1] = "")	(count(Exclude_Grape_Preferences) >= 1) and (Exclude_Grape_Preferences[1] != "")	Wines[not(list contains(Exclude_Grape_Preferences,item))]	no include, exclude
3	(count(Include_Grape_Preferences) >= 1) and (Include_Grape_Preferences[1] != "")	(count(Exclude_Grape_Preferences) = 1) and (Exclude_Grape_Preferences[1] = "")	Wines[list contains(Include_Grape_Preferences,item)]	include, no exclude
4	(count(Include_Grape_Preferences) >= 1) and (Include_Grape_Preferences[1] != "")	(count(Exclude_Grape_Preferences) >= 1) and (Exclude_Grape_Preferences[1] != "")	Wines[list contains(Include_Grape_Preferences,item)][not(list contains(Exclude_Grape_Preferences,item))]	include, exclude

Figure 2.22: Grape Wine Preferences Dec. Table

Suggested_Wines

The last but most important decision table is **Suggested_Wines**, which is responsible for determining the list of wines that will be matched with all the user preferences that were previously described.

Intersection rule is also applied to this table, but in a slightly different way to the previous tables:

$$
\begin{aligned}
&((\ \textbf{union}\ (\ Meal\_Strong\_Wines, Meal\_Weak\_Wines\ ))[\\
&\textbf{list contains}(Grape\_Wine\_Preferences, item)]\\
&[\textbf{list contains}(Flavor\_Wine\_Preferences, item)])\\
&[\textbf{list contains}(Geo\_Wine\_Preferences, item)]\\
&[\textbf{list contains}(flatten(Wine\_Color), item)]
\end{aligned}
$$

This expression, starts by computing the **union** between the list of wines with strong and weak recommendations (*Meal_ Strong_ Wines* and *Meal_ Weak_ Wines*), with the aim of selecting all wines from these 2 lists. The expression continues by applying our intersection rule between the union just obtained and the remaining lists: the list of wines associated with grape preference (*Grape_ Wine_ Preferences*), the list of wines associated with taste preference (*Flavor_ Wine_ Preferences*), the list of wines associated with geographical preference (*Geo_ Wine_ Preferences*) and finally the list of wines associated with color preference (*Wine_ Color*), in particular we use the flatten function because the result may be a list with nested list, so since flatten returns a list that includes all elements of the given list without nested lists and we simply need a list of wines, we used it for this reason.

We notice that, unlike other expressions, we do not need to consider multiple cases or check for empty lists in this particular situation. This is because we know that we will not have any empty lists according to how we defined the output of the previous tables when the user does not express any preferences, all wines are "paired" and consequently the *Wines* list is output.

In the second row, there is the print of the error when the user wants to include a region that does not belong to a country or vice versa.

Suggested_Wines | Hit Policy: Unique

	When	And	And	And	And	And	Then
	Meal_Strong_Wines	Meal_Weak_Wines	Grape_Wine_Preferences	Flavor_Wine_Preferences	Geo_Wine_Preferences	Wine_Color	Suggested_Wines
1	-	-	-	-	count(Geo_Wine_Preferences) != 0	-	((union(Meal_Strong_Wines,Meal_Weak_Wines))[list contains(Grape_Wine_Preferences,item)][list contains(Flavor_Wine_Preferences,item)])[list contains(Geo_Wine_Preferences,item)][list contains(flatten(Wine_Color),item)]
2	-	-	-	-	count(Geo_Wine_Preferences) = 0	-	"ERROR: Country and Region don't match!"

Figure 2.23: Suggested Wines Dec. Table

2.5 Input/Output Example

An example using the Camunda online simulator is shown below in Figure 2.24. In this case, the user wants to discover the possible wines to pair with the pulled pork meal. Additionally, he just wants to see, all wines that have Sangiovese as a grape. We can clearly see all the output lists produced by the decision tables on the right-hand side of the figure, where the final list is *Suggested_ Wines* and in this case we can notice that the pulled pork has only strong wine recommendations.

Figure 2.24: I/O Camunda Example

3. Rule-Based Systems

In Computer Science, a **Rule-Based System** [12] is a system used to store and manipulate knowledge to interpret information in a useful way.
This type of system is highly utilised in Artificial Intelligence and its domains, such as Machine Learning. The power of a rule-based system is the domain-specific expert system that uses rules to make inferences or choices. Rule-based programming attempts to derive execution instructions from a set of source data and rules, and thus unlike other programming paradigms, this is a more elastic and unconstrained approach.

Rule-Based System Elements

In general, the elements of a rule-based system are the following:

- **Rule Base**: the list of rules specific to their knowledge base;
- **Semantic Reasoner**: infers information or takes actions based on the interaction between the input and the rule base;
- **Temporary working memory**: for data storing;
- **UI**: to send input and receive output.

3.1 Prolog

Logic programming utilizes logic as a declarative representation language and backward chaining as an inference rule. **Prolog** [10] is a logic programming language associated with artificial intelligence and computational linguistics, the strength of this language lies in the **logic**. Prolog syntax provides:

- **Symbols**: **,** (and) **;** (or) **:-** (if) **not** (not);
- **Variables**: a variable in Prolog is a string of letters, digits, and underscores (_) beginning either with a capital letter or with an underscore;
- **Facts**: a fact is a predicate expression that makes a declarative statement about the problem domain;
- **Rules**: a rule is a predicate expression that uses logical implication (:-) to describe a relationship among facts. Thus a Prolog rule takes the form:

$$left\_hand\_side \textbf{ :- } right\_hand\_side \textbf{ .}$$

- **Queries**: the Prolog interpreter answers queries on the facts and rules represented in its database. By asking a question, Prolog is asked whether it can prove that the question is true. If the answer is "yes", Prolog answers "yes" and displays all the links between the variables made to obtain the answer. If it cannot prove that the query is true, it answers "no".;

Furthermore, in Prolog, the technique of **backtracking** is employed. Backtracking is a process where Prolog examines the truthfulness of various predicates by evaluating their correctness. Essentially, the Prolog interpreter explores different potential paths through backtracking until it discovers a valid route that reaches the final destination. Moreover, there are also a some **built-in predicates**, i.e. native rules made available by Prolog, such as the "*length(?List, ?Length)*" which allows to get the length dimension *Length* of a list *List*. We used some of these built-in predicates for different reasons, in particular to determine the wine categories associated with each meal.

As Prolog is so powerful that it allows defining one's knowledge base in just a few lines, we used it to create our rule-based system regarding wines and food pairing.

3.2 Implementation

As our Prolog code is several hundred lines long, for reasons of space and readability, the entire code will not be reproduced within this chapter. The complete code is illustrated at the end of the report.

Since in Prolog, the order in which facts and predicates are defined is important, we initially defined some **list operations** useful for grape rules and for obtaining wine categories from meal ingredients:

```
% list operations
% * list membership (utility for grape inclusion/exclusion)
is_in_list(H,[H|_]).
is_in_list(H,[_|T]) :- is_in_list(H,T).
list_intersection([], _, []).
list_intersection([H|T], L2, [H|L1]) :- is_in_list(H,L2), !, list_intersection(T,L2,L1).
list_intersection([_|T],L2,L1) :- list_intersection(T,L2,L1).
% * concatenation between lists including duplicates (utility for meal derivation from ingredients)
list_concat([],L,L).
list_concat([X1|L1],L2,[X1|L3]) :- list_concat(L1,L2,L3).
% * count list occurrencies (utility for meal derivation from ingredients)
list_count([],_H,0). %_H to suppress Singleton Warning
list_count([H|T],H,C):- list_count(T,H,O), C is 1+O.
list_count([H1|T],H,C):- H1\=H, list_count(T,H,C).
list_countall(L,H,C) :-
   is_in_list(H,L),
   list_count(L,H,C).
```

Listing 3.1: List operations

Then we defined **countries**, **regions**:

```
% countries
country(france).
country(germany).
country(italy).
country(spain).

% regions
region(abruzzi).
region(anjou-saumur).
region(aragona).
region(andalusia).
region(bordeaux).
region(burgundy).
region(chateau_d_esclans).
region(piemonte).
region(rheinhessen).
region(rhone).
region(tuscany).
region(vaucluse).
region(vosne_romanee).
```

Listing 3.2: Countries and regions

and the rule **region_of** to define that the region is part of that specific state:

```
% country regions
region_of(france, anjou-saumur).
region_of(france, bordeaux).
region_of(france, burgundy).
region_of(france, chateau_d_esclans).
region_of(france, rhone).
region_of(france, vaucluse).
region_of(france, vosne_romanee).
region_of(germany, rheinhessen).
region_of(italy, abruzzi).
region_of(italy, friuli).
region_of(italy, piemonte).
region_of(italy, tuscany).
region_of(spain, andalusia).
region_of(spain, aragona).
```

Listing 3.3: Regions of country

The facts about the **grape** follow:

```
% grapes
grape(brunello).
grape(cabernet_franc).
grape(cabernet_sauvignon).
grape(carignan).
grape(chardonnay).
grape(chardonnay_rose).
grape(cremant).
grape(grenache).
grape(marsanne).
grape(merlot).
grape(moscato).
grape(palomino).
grape(pinot_noir).
grape(riesling).
grape(sangiovese).
grape(syrah).
grape(trebbiano).
grape(vermentino).
```

Listing 3.4: Grapes

As mentioned in the previous chapters, we have defined 3 wine colors and the following 9 **wine categories**:

```
% wine colors
wine_color(red).
wine_color(rose).
wine_color(white).

% wine categories
wine_category(bold_red).
wine_category(medium_red).
wine_category(light_red).
wine_category(rose_).
wine_category(rich_white).
wine_category(light_white).
wine_category(sparkling).
wine_category(sweet_white).
wine_category(dessert).
```

Listing 3.5: Wine colors and categories

And we associate the wine color to each wine category:

```
% wine color categories
wine_color_of(bold_red, red).
wine_color_of(medium_red, red).
wine_color_of(light_red, red).
wine_color_of(rose_, rose).
wine_color_of(rich_white, white).
wine_color_of(light_white, white).
wine_color_of(sparkling, white).
wine_color_of(sweet_white, white).
wine_color_of(dessert, red).
```

Listing 3.6: Wine color of wine categories

With regard to **taste**, **tannin** and **dryness level**, we have come up with the following facts:

```
% tastes
taste(bold).
taste(light).

% tannin levels
tannin_level(tannic).
tannin_level(less_tannic).

% dryness
dry_level(dry).
dry_level(not_dry).
```

Listing 3.7: Flavor wine preferences

As with some previous facts, those of **wines** are as follows:

```
% wines
wine(aragone).
wine(brunello_di_montalcino).
wine(case_basse_sangiovese).
wine(corton_charlemagne_grand_cru).
wine(coteau_de_la_beylesse).
wine(cremant_d_alsace).
wine(fino).
wine(g_max_reisling).
wine(garrus_rose).
wine(hermitage_aoc).
wine(juline_chateauneuf_du_pape).
wine(marsannay).
wine(occhio_di_pernice).
wine(pomerol_chateau_de_salesl).
wine(romanee_saint).
wine(sauternes).
wine(trebbiano_d_abruzzo).
wine(vajra_mosacto_d_asti).
```

Listing 3.8: Wines

Since a wine consists of one or more grapes, we have defined the grapes_of rule. The

logic is that a wine has a **list of grapes** associated with it:

```
% wine grapes
grapes_of(aragone, [carignan]).
grapes_of(brunello_di_montalcino, [brunello, sangiovese]).
grapes_of(case_basse_sangiovese, [sangiovese]).
grapes_of(corton_charlemagne_grand_cru, [chardonnay]).
grapes_of(coteau_de_la_beylesse, [marsanne]).
grapes_of(cremant_d_alsace, [cremant]).
grapes_of(fino, [palomino]).
grapes_of(g_max_reisling, [riesling]).
grapes_of(garrus_rose, [vermentino]).
grapes_of(hermitage_aoc, [marsanne]).
grapes_of(juline_chateauneuf_du_pape, [grenache, syrah]).
grapes_of(marsannay, [chardonnay_rose]).
grapes_of(occhio_di_pernice, [sangiovese, syrah]).
grapes_of(pomerol_chateau_de_salesl, [cabernet_franc, cabernet_sauvignon, merlot]).
grapes_of(romanee_saint, [pinot_noir]).
grapes_of(sauternes, [chardonnay]).
grapes_of(trebbiano_d_abruzzo, [trebbiano]).
grapes_of(vajra_mosacto_d_asti, [moscato]).
```

Listing 3.9: Grapes of wines

While to associate the **region of the wine's origin**, we defined the wine_from rule, which allows for logically inferring the country associated with that region:

```
% wine regions
wine_from(aragona, aragone).
wine_from(tuscany, brunello_di_montalcino).
wine_from(tuscany, case_basse_sangiovese).
wine_from(burgundy, corton_charlemagne_grand_cru).
wine_from(rhone, coteau_de_la_beylesse).
wine_from(anjou-saumur, cremant_d_alsace).
wine_from(andalusia, fino).
wine_from(rheinhessen, g_max_reisling).
wine_from(chateau_d_esclans, garrus_rose).
wine_from(rhone, hermitage_aoc).
wine_from(vaucluse, juline_chateauneuf_du_pape).
wine_from(burgundy, marsannay).
wine_from(tuscany, occhio_di_pernice).
wine_from(bordeaux, pomerol_chateau_de_salesl).
wine_from(vosne_romanee, romanee_saint).
wine_from(bordeaux, sauternes).
wine_from(abruzzi, trebbiano_d_abruzzo).
wine_from(piemonte, vajra_mosacto_d_asti).
```

Listing 3.10: Wine's origin

As it can be guessed, we have associated the **wine categories** with the wines in the same way:

```
% wine categories
category_of(brunello_di_montalcino, bold_red).
category_of(juline_chateauneuf_du_pape, bold_red).
category_of(case_basse_sangiovese, medium_red).
category_of(pomerol_chateau_de_salesl, medium_red).
category_of(aragone, light_red).
category_of(romanee_saint, light_red).
category_of(garrus_rose, rose_).
category_of(marsannay, rose_).
category_of(corton_charlemagne_grand_cru, rich_white).
category_of(hermitage_aoc, rich_white).
category_of(sauternes, light_white).
category_of(trebbiano_d_abruzzo, light_white).
category_of(coteau_de_la_beylesse, sparkling).
category_of(cremant_d_alsace, sparkling).
category_of(g_max_reisling, sweet_white).
category_of(vajra_mosacto_d_asti, sweet_white).
category_of(fino, dessert).
category_of(occhio_di_pernice, dessert).
```

Listing 3.11: Wine categories

We also applied the same reasoning for the **wine's flavour**:

```
% wine tastes
taste_of(aragone, bold).
taste_of(brunello_di_montalcino, bold).
taste_of(case_basse_sangiovese, bold).
taste_of(corton_charlemagne_grand_cru, light).
```

Listing 3.12: Wine taste

```
% wine tannin levels
tannin_of(aragone, tannic).
tannin_of(brunello_di_montalcino, tannic).
tannin_of(case_basse_sangiovese, tannic).
tannin_of(corton_charlemagne_grand_cru, tannic).
```

Listing 3.13: Wine tannin level

```
% wine drynesses
dryness_of(aragone, dry).
dryness_of(brunello_di_montalcino, dry).
dryness_of(case_basse_sangiovese, dry).
dryness_of(corton_charlemagne_grand_cru, dry).
```

Listing 3.14: Wine dryness

Unlike other knowledge bases, here in Prolog we have defined all the **ingredients** we use to create our dishes:

```
% ingredients
ingredient(fish).
ingredient(herbs).
ingredient(pork).
ingredient(saulted_or_fried).
ingredient(black_pepper).
ingredient(white_starches).
ingredient(alliums).
ingredient(red_pepper).
ingredient(grilled).
ingredient(smoked).
ingredient(red_meat).
ingredient(root_vegetables_and_squash).
ingredient(soft_cheese_and_cream).
ingredient(poultry).
ingredient(roasted).
ingredient(fruit_and_berries).
```

Listing 3.15: Ingredients

While the **meals** are the following:

```
% meals
meal(fried_bass_with_herbs).
meal(pulled_pork).
meal(pumpkin_risotto).
meal(roast_chicken_with_herbs).
meal(sliced_beef).
meal(strawberry_cheesecake).
meal(pumpkin_risotto).
```

Listing 3.16: Meals

Then each dish was associated with its own **list of ingredients**:

```
% meal ingredients
ingredient_of(fried_bass_with_herbs, [fish, herbs, saulted_or_fried]).
ingredient_of(pulled_pork, [pork, black_pepper, white_starches, alliums, red_pepper, grilled, smoked]).
ingredient_of(sliced_beef, [red_meat, grilled]).
ingredient_of(pumpkin_risotto, [white_starches, root_vegetables_and_squash, soft_cheese_and_cream]).
ingredient_of(roast_chicken_with_herbs, [poultry, herbs, roasted]).
ingredient_of(strawberry_cheesecake, [fruit_and_berries]).
```

Listing 3.17: Ingredients of each meal

In order to identify **strong and weak recommendations** for wines, we have defined a rule in which the ideal wine category is paired with the ingredient. In this way, it is easy to infer the strong and the weak wine categories of each ingredient via the **strong_category_of** (Fig. 3.18) and **weak_category_of** (3.19) rules. Consequently, once the category of wine is inferred, it is simple to infer the strong and the weak wine via the **category_of** rule (Fig. 3.11). So, we have defined the following rules:

```
% ingredients strong categories
strong_category_of(fish, light_white).
strong_category_of(herbs, light_white).
strong_category_of(saulted_or_fried, light_red).
strong_category_of(black_pepper, bold_red).
strong_category_of(pork, medium_red).
strong_category_of(red_pepper, medium_red).
strong_category_of(alliums, medium_red).
strong_category_of(grilled, bold_red).
```

Listing 3.18: Strong categories of ingredients

```
% ingredients weak categories
weak_category_of(fish, rich_white).
weak_category_of(fish, sparkling).
weak_category_of(herbs, rich_white).
weak_category_of(herbs, rose_).
weak_category_of(herbs, light_red).
weak_category_of(herbs, medium_red).
weak_category_of(saulted_or_fried, rose_).
weak_category_of(saulted_or_fried, rich_white).
weak_category_of(saulted_or_fried, light_white).
```

Listing 3.19: Weak categories of ingredients

In order to make the final rule **suggested_wines** work, we had to define these subsequent rules.

Rules: wine_checking

To infer the W wine, we need this checking rule which works by taking IC included country, EC excluded country, IR included region, ER excluded region, $ICOL$ included color, $ECOL$ excluded color and flavour (T taste, TL tannin level and D dryness level):

```
% rules
wine_checking(W,IC,EC,IR,ER,ICOL,ECOL,T,TL,D) :- wine(W),
   region_belongs(IR,IC), exclude_wine_by_country(W,EC), exclude_wine_by_region(W,ER),
   wine_from(IR,W),
   wine_category(CAT), wine_color_of(CAT,ICOL),
   ICOL \== ECOL, category_of(W,CAT),
   taste_of(W,T), tannin_of(W,TL), dryness_of(W,D).
```

Listing 3.20: Wine checking

Rules: is_ideal_strong/weak_category

As we have defined our previous facts and predicates, we can infer a strong (or weak) C wine category paired with a M meal: it is possible to infer C getting all the I ingredients of M and checking if the ING ingredient belonging to I has a strong (or weak) C wine category. In this context, we defined the following 2 rules:

```
ideal_strong_category(M,C) :-
   ingredient_of(M,I),
   is_in_list(ING,I),
   strong_category_of(ING,C).

ideal_weak_category(M,C) :-
   ingredient_of(M,I),
   is_in_list(ING,I),
   weak_category_of(ING,C).
```

Listing 3.21: Strong/weak wine category

Rules: get_strongs/weaks

A built-in predicate that has been very useful to us is "**bagof**", which produces a list in which can be contained repetitions of the instances that satisfy a predicate. In our case, we used bagof to get the R list of all strong (or weak) categories associated with each ingredient in our M meal, including repeated categories. The duplicate categories are crucial to the reasoning applied to determine the strong and weak categories.

```
get_strongs(M,R) :- bagof(
          X,
          ideal_strong_category(M,X),
          R).

get_weaks(M,R) :- bagof(
          X,
          ideal_weak_category(M,X),
          R).
```

Listing 3.22: Strong/weak wine categories using bagof

Rules: ideal_strong/weak_categories

To determine strong categories of a M meal we have 2 cases:

1. all the ingredients have that C strong category;
2. there are ingredients that have C weak category but at least one has C as strong category.

Unfortunately, it is not so obvious to get the list of strong categories easily in Prolog but we still put our following theory into practice:

- first of all we get the *R1* strong categories and the *R2* weak categories of M through **get_strongs(M,R1)** and **get_weaks(M,R2)** rules;
- then we make the R concatenation between these 2 lists keeping duplicate categories using the **list_concat(R1,R2,R)**;
- once we have R, we check if each CAT category belongs to the *R1* strong wine categories and to R;
- at this point we use **list_countall(R,CAT,O)** in order to calculate the O number of CAT occurrences in R;

- we also have to calculate the L number of ingredients of M and if O is grather or equal then L then we add CAT to C list using **setof**.

Obviously at first glance this seems like complicated reasoning, but just think that to determine strong categories, it is sufficient that the total number of strong categories associated with each ingredient is greater than or equal to the number of ingredients. To count in this way, the use of **bagof** is crucial, especially to determine strong categories in case 2: it allows us to maintain the list of all wine categories associated with each ingredient keeping duplicates and precisely with these duplicated categories we can count all weak and strong categories and classify as strong the category that has all weak and at least one strong.

```
ideal_strong_categories(M,C) :-
   setof(CAT,
       (get_strongs(M,R1),
            get_weaks(M,R2),
            list_concat(R1,R2,R),
          wine_category(CAT),
          is_in_list(CAT,R1), % check if is a strong category
          is_in_list(CAT,R), % and is in concatenation list
          list_countall(R,CAT,O),
          ingredient_of(M,I),
          length(I,L), O >= L),
       C).
```

Listing 3.23: Strong wine categories of a meal

To determine weak categories the same reasoning is done but only checking that all ingredients have that C category as weak:

```
ideal_weak_categories(M,C) :-
   setof(CAT,
       (get_weaks(M,R),
          wine_category(CAT), is_in_list(CAT,R),
          list_countall(R,CAT,L),
          ingredient_of(M,I),
          length(I,L)),
       C).
```

Listing 3.24: Weak wine categories of a meal

The result of these two rules will be the C list of all wine categories associated with that M meal, or rather two lists: one for strong and one for weak wine categories. In both rules the **setof** rule is used to remove duplicate categories from C, since we do not need them from now on.

Rules: is_ideal_strong/weak_wine

As we have defined our previous facts and predicates, we can infer a strong (or weak) W wine paired with a M meal: it is possible to infer the ideal wine through its CAT category, specifically the category that belongs to the C list of all categories. In this context, we defined the following 2 rules:

```
is_ideal_strong_wine(W,M) :- wine_category(CAT), ideal_strong_categories(M,C),
   is_in_list(CAT,C), category_of(W,CAT).

is_ideal_weak_wine(W,M) :- wine_category(CAT), ideal_weak_categories(M,C),
   is_in_list(CAT,C), category_of(W,CAT).
```

Listing 3.25: Ideal strong/weak wine

Rules: region_belongs

This rule allows to infer all R regions that belong to the C country:

```
region_belongs(R,C) :- country(C), region(R), region_of(C,R).
```

Listing 3.26: Region belongs to a country

In other words, this rule allows us to include the specified region and country.

Rules: exclude_wine_by_country

To exclude wines by country, we check that the EC country to be excluded is different from all C countries:

```
exclude_wine_by_country(W,EC) :- wine_from(R,W), region_belongs(R,C),
   EC \== C.
```

Listing 3.27: Exclude wine by country

Rules: exclude_wine_by_region

To exclude wines by region, we check that the ER region to be excluded is different from all R regions:

```
exclude_wine_by_region(W,ER) :- wine_from(R,W),
   ER \== R.
```

Listing 3.28: Exclude wine by region

Rules: include/exclude_wine_by_grape

In this system, the user can decide to include or exclude one or more grapes. To allow this preference, the reasoning is as follows.

Once the IG (or EG) list of grapes to be included (or excluded) is taken two events can occur. If it is empty (the user has no grape preference) all W wines with any possible grape (_) are inferred. Otherwise, W wines that have GS grapes contained in the IG (or EG) list are inferred: in other words, if a wine has one or more grapes contained in the list of grapes to be included, it is inferred.

```
include_wine_by_grape(W,IG) :- IG=[] -> (grapes_of(W,_));
   grapes_of(W,GS), is_in_list(G,IG), is_in_list(G,GS).

exclude_wine_by_grape(W,EG) :- grapes_of(W,GS), list_intersection(GS,EG,NW),
   not(is_in_list(G,NW)), is_in_list(G,GS).
```

Listing 3.29: Include/Exclude wine by grapes

Rules: suggested_strong/weak_wines

To calculate which W wines are recommended with the given user preferences, without taking into account if the suggestion is strong or weak, we can use the following rule:

```
suggested_general_wines(W,M,IG,EG,C,EC,R,ER,COL,ECOL,T,TL,D) :-
   wine_checking(W,C,EC,R,ER,COL,ECOL,T,TL,D),
   include_wine_by_grape(W,IG),
   exclude_wine_by_grape(W,EG),
   meal(M).
```

Listing 3.30: Suggested general wines

And to know which wine is strongly recommended or weakly recommended, we simply call the rule **is_strong_ideal_wine** or **is_weak_ideal_wine** immediately after the rule explained above.

```
suggested_strong_wines(STRONG,M,IG,EG,C,EC,R,ER,COL,ECOL,T,TL,D) :-
      suggested_general_wines(STRONG,M,IG,EG,C,EC,R,ER,COL,ECOL,T,TL,D),
   is_ideal_strong_wine(STRONG,M).

suggested_weak_wines(WEAK,M,IG,EG,C,EC,R,ER,COL,ECOL,T,TL,D) :-
      suggested_general_wines(WEAK,M,IG,EG,C,EC,R,ER,COL,ECOL,T,TL,D),
   is_ideal_weak_wine(WEAK,M).
```

Listing 3.31: Suggested strong and weak wines

Rules: suggested_wines

This rule, merges the **is_strong_ideal_wine** and **is_weak_ideal_wine** rules, in order to know which W wine to pair along with the strong or weak recommended category.

```
suggested_wines(STRONG,WEAK,M,IG,EG,IC,EC,IR,ER,ICOL,ECOL,T,TL,D) :-
   setof(STRONG, suggested_strong_wines(STRONG,M,IG,EG,IC,EC,IR,ER,ICOL,ECOL,T,TL,D), STRONG);
   setof(WEAK, suggested_weak_wines(WEAK,M,IG,EG,IC,EC,IR,ER,ICOL,ECOL,T,TL,D), WEAK).
```

Listing 3.32: Suggested wines

Note also here the use of **setof** to avoid multiple outputs of wine categories.

3.3 Query Example

An example of a query in Prolog is shown in Fig. 3.1 where, as in the previous chapter, the user wants to discover the possible wines to pair with the pulled pork meal and that are made by the Sangiovese grape. As we can see in Fig. 3.1, there are 2 strong wines associated with the pulled pork and these both have the sangiovese as grape. Moreover there are not weak wines with sangiovese.

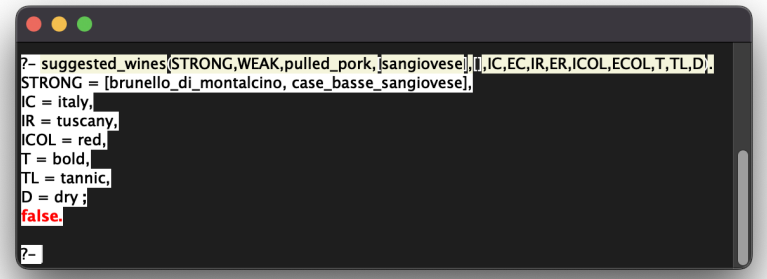


Figure 3.1: Prolog query example

As we can see in Fig. 3.2 without using the **setof** built-in predicate in the *suggested_ wines* rule, "Brunello di Montalcino" output was replicated, this is because the given wine is made from two different grapes (brunello and sangiovese), thus the backtracking of the Prolog's engine shows the output two times (one for each grape). For this reason, we used setof avoiding to repeat a wine in the output.

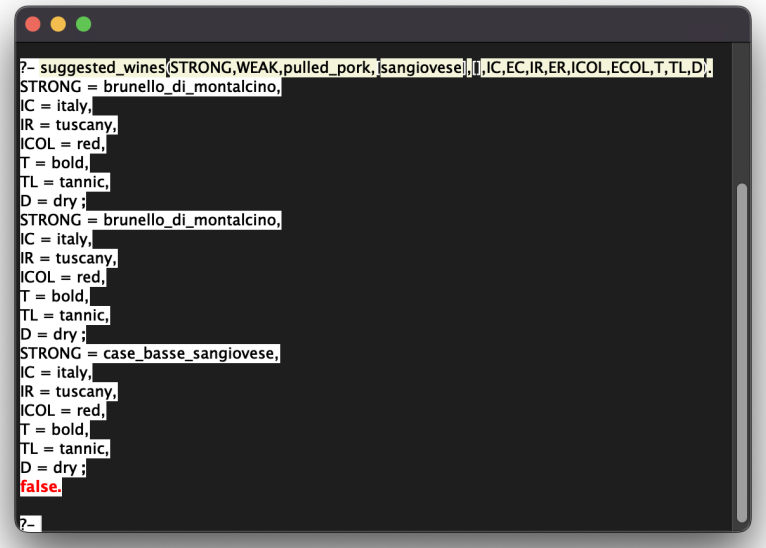


Figure 3.2: Prolog query example without using setof

4. Ontology Engineering

In Computer Science, **ontology engineering** [9] is a field that studies methodologies for creating ontologies, which includes a representation, formal naming and definition of categories, properties and relationships between concepts, data and entities.
Ontologies are represented in different data models including RDF, OWL, Neo4J and more. In order to build our ontology, we have used RDF.

RDF

The **Resource Description Framework (RDF)** [11] is a standard used as a general method for the description and the exchange of graph data.
RDF allows for describing resources used to build knowledge graphs.

A **Knowledge Graph** represents a network of real-world entities like objects, events, situations, or concepts and illustrates the relationship between them.

Knowledge Graph Features

Knowledge graphs merge the features of different data management paradigms:

- **Knowledge base**;
- **Graph**;
- **Database**.

Ontology Elements

In order to create an ontology, the following 4 steps are required:

- the **class definition**;
- the **class hierarchy**;
- **defining properties** and **assigning values** to these properties;
- **creation of instances** and assignment of instance values.

4.1 Protégé

Protégé [5] is a free, open-source ontology editor and a knowledge management system. It allows the definition of **Semantic Web Rule Language (SWRL)** [13] to create

inference rules that can generate new data based on existing ontologies. Furthermore, it supports the **Sparql Protocol And Rdf Query Language (SPARQL)** [14], which is a semantic query language for databases capable of retrieving and manipulating data stored in RDF format.

4.2 Our Ontology

By using **Protégé**, we have defined our, ontology taking into account the main entities, which are *wine*, the wine *category*, the wine *color grape*, the origin of wine production (*country* and *region*), the *meal* and the *meal ingredients*.

Classes

The main entities defined before, are also our main classes. However, we have also defined a hierarchy. This is to allow other resources to be added in the future, such as another possible drink different from the wine, like beer, water and so on.

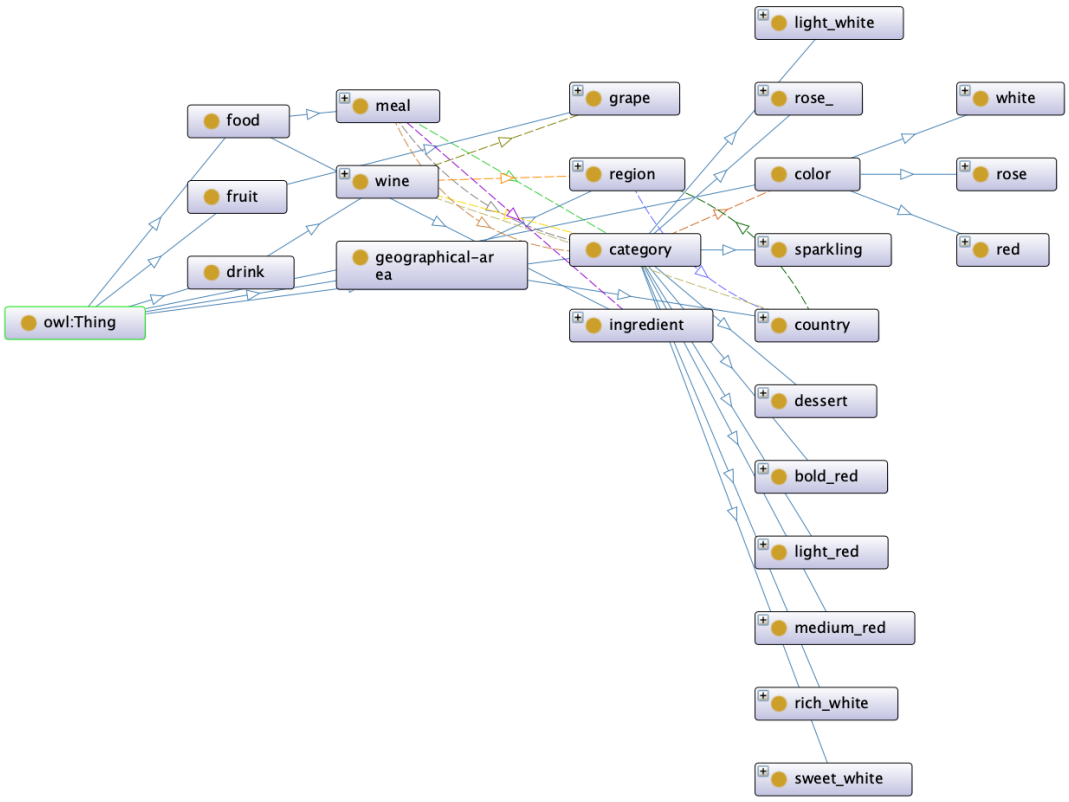


Figure 4.1: Our Ontology Graph in Protégé

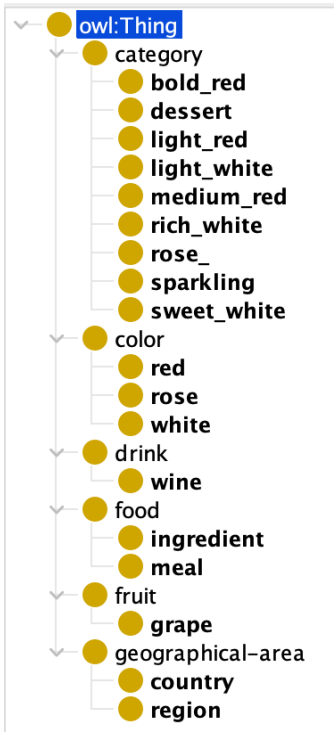


Figure 4.2: Our Model in Protégé

Data Properties

Data Properties represent the attributes of the classes. We defined 3 data properties for the *wine* class: **has_taste** with range {"bold","light"} and two boolean properties **is_dry** and **is_tannic**. While we have defined **recommendation** data property with range {"strong","weak"} for the *category* class.

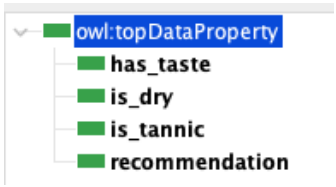


Figure 4.3: Data properties

Object Properties

Object properties are used to represent the relationships between classes. We defined several object properties for the *wine* class:

- **has_category**: to associate a category with wine;
- **has_grape**: to associate one or more grapes with wine;
- **is_wine_from_country**: this is a SWRL inferred by the reasoner, useful to understand the country origin of a wine;
- **is_wine_from_region**: to associate the region origin with wine;

For the *country* class we have defined the **is_country_of** object property to associate a country with one or more regions.
While, for the *region* class, we defined the **is_region_of** object property, to join a region with a specific country.
About wine recommendations, 2 object properties were defined for the *category* class: **is_ideal_strong** and **is_ideal_weak** that both associate a meal with one or more wine categories, one representing a strong and a weak recommendation.
For the *meal* class the **has_color** was defined to associate a wine category to a color, while for the *meal* class we have defined 2 object properties: the **is_ideal_category** object property, to associate the ideal category of a meal and the **has_ingredient** object property, to define which ingredients a meal consists of.

Figure 4.4: Object properties

Individuals

To reduce space, we have omitted the inclusion of all individuals belonging to the 6 main classes in our knowledge base in this report, even though they have been defined.

SW Rules

We have exploited the reasoner provided by Protégé its power by defining the following rules:

S1

$$kebi2023 : is\_region\_of(?r, ?c) \rightarrow kebi2023 : is\_country\_of(?c, ?r)$$

This rule infers which city *?c* has a *?r* region, knowing that a *?r* region belongs to that *?c* city.

S2

$$kebi2023 : is\_wine\_from\_region(?w, ?r) \wedge\ kebi2023 : is\_region\_of(?r, ?c) \rightarrow$$

$$kebi2023 : is\_wine\_from\_country(?w, ?c)$$

This rule infers which wine *?w* comes from country *?c*, knowing that *?w* comes from *?r* region and that *?r* region belongs to *?c* country.

S3

$$kebi2023 : meal(?m) \land\ kebi2023 : is\_ideal\_strong(?m, ?c) \rightarrow$$

$$kebi2023 : recommendation(?c, \text{“}strong\text{”}) \land\ kebi2023 : is\_ideal\_category(?m, ?c)$$

This rule infers which *?m* meal has a strong *?c* category and defines it as an ideal category with a strong recommendation.

S4

$$kebi2023 : meal(?m) \land\ kebi2023 : is\_ideal\_weak(?m, ?c) \rightarrow$$

$$kebi2023 : recommendation(?c, \text{“}weak\text{”}) \land\ kebi2023 : is\_ideal\_category(?m, ?c)$$

This rule infers which *?m* meal has a weak *?c* category and defines it as an ideal category with a weak recommendation.

Unfortunately, due to the way we defined the knowledge base and the way we devised our strong and weak recommendation method, it is not possible in Protégé to define rules for the automatic inference of wine categories from ingredients, but other pairing reasoning can of course be applied.

SPARQL Queries

This section shows some possible queries in our ontology. For convenience, prefixes are defined only on the first query but are present on all of them.

All Wines

```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX kebi: <http://www.semanticweb.org/piermichele.rosati-marco.serenelli/kebi2023#>

SELECT ?wine WHERE {
        ?wine rdf:type kebi:wine.
}
```

Listing 4.1: All wines in the knowledge base

All Meals

```
SELECT ?meal WHERE{
        ?meal rdf:type kebi:meal.
}
```

Listing 4.2: All meals in the knowledge base

All Red Wines

```
SELECT ?w WHERE{
?w kebi:has_category ?c.
?c kebi:has_color kebi:color_red
}
```

Listing 4.3: All red wines

All Rosé Wines

```
SELECT ?w WHERE{
?w kebi:has_category ?c.
?c kebi:has_color kebi:color_rose
}
```

Listing 4.4: All rosé wines

All White Wines

```
SELECT ?w WHERE{
?w kebi:has_category ?c.
?c kebi:has_color kebi:color_white
}
```

Listing 4.5: All white wines

All Dried Wines

```
SELECT ?w WHERE{
        ?w kebi:is_dry TRUE.
}
```

Listing 4.6: All dried wines

All Not Dried Wines

```
SELECT ?w WHERE{
        ?w kebi:is_dry FALSE.
}
```

Listing 4.7: All not dried wines

All Tannic Wines

```
SELECT ?w WHERE{
        ?w kebi:is_tannic TRUE.
}
```

Listing 4.8: All tannic wines

All Not Tannic Wines

```
SELECT ?w WHERE{
        ?w kebi:is_tannic FALSE.
}
```

Listing 4.9: All not tannic wines

All Bold Wines

```
SELECT ?w WHERE{
        ?w kebi:has_taste "bold".
}
```

Listing 4.10: All bold wines

All Light Wines

```
SELECT ?w WHERE{
        ?w kebi:has_taste "light".
}
```

Listing 4.11: All light wines

All regions of All Countries

```
SELECT ?region ?country WHERE {
        ?region kebi:is_region_of ?country.
}
```

Listing 4.12: All regions that belong the corresponding country

All Wines From all Countries

```
SELECT ?w ?c WHERE{
    ?w kebi:is_wine_from_country ?c.
    ?c rdf:type kebi:country.
}
```

Listing 4.13: All wines that belong the corresponding country

All Wines From all Regions

```
SELECT ?w ?r WHERE{
    ?w kebi:is_wine_from_region ?r.
    ?r rdf:type kebi:region.
}
```

Listing 4.14: All wines that belong the corresponding region

All Strong Wines paired with Grilled Bass with Herbs

```
SELECT ?strong WHERE{
    kebi:meal_fried_bass_with_herbs kebi:is_ideal_strong ?c.
    ?strong kebi:has_category ?c.
}
```

Listing 4.15: All strong wines that have a pairing with the Grilled Bass with Herbs

All Weak Wines paired with Grilled Bass with Herbs

```
SELECT ?weak WHERE{
    kebi:meal_fried_bass_with_herbs kebi:is_ideal_weak ?c.
    ?weak kebi:has_category ?c.
}
```

Listing 4.16: All weak wines that have a pairing with the Grilled Bass with Herbs

All Wines paired with Pulled Pork

```
SELECT ?w WHERE{
        kebi:meal_pulled_pork kebi:is_ideal_category ?c.
        ?w kebi:has_category ?c.
}
```

Listing 4.17: All wines that have a pairing with the Pulled Pork

All Wines paired with Pumpkin Risotto

```
SELECT ?w WHERE{
        kebi:meal_pumpkin_risotto kebi:is_ideal_category ?c.
        ?w kebi:has_category ?c.
}
```

Listing 4.18: All wines that have a pairing with the Pumpkin Risotto

All Wines Without Moscato Grape paired with Pumpkin Risotto

The negation is made using **MINUS**, a SPARQL way based on removing matches by evaluating two patterns (from the SPARQL documentation `https://www.w3.org/TR/sparql11-query/#negation`).

```
SELECT ?w WHERE{
    kebi:meal_pumpkin_risotto kebi:is_ideal_category ?c .
    ?w kebi:has_category ?c.
    MINUS {
        ?w kebi:has_grape kebi:grape_moscato.
    }
}
```

Listing 4.19: All wines that have a pairing with the Pumpkin Risotto

All Wines paired with Roast Chicken with Herbs

```
SELECT ?w WHERE{
        kebi:meal_roast_chicken_with_herbs kebi:is_ideal_category ?c.
        ?w kebi:has_category ?c.
}
```

Listing 4.20: All wines that have a pairing with the Roast Chicken with Herbs

All Wines paired with Sliced Beef

```
SELECT ?w WHERE{
        kebi:meal_sliced_beef kebi:is_ideal_category ?c.
        ?w kebi:has_category ?c.
}
```

Listing 4.21: All wines that have a pairing with the Sliced Beef

All Wines paired with Strawberry Cheesecake

```
SELECT ?w WHERE{
        kebi:meal_strawberry_cheesecake kebi:is_ideal_category ?c.
        ?w kebi:has_category ?c.
}
```

Listing 4.22: All wines that have a pairing with the Strawberry Cheesecake

All Not White Weak Wines paired with Strawberry Cheesecake

```
SELECT ?weak WHERE{
        kebi:meal_strawberry_cheesecake kebi:is_ideal_weak ?c .
        ?weak kebi:has_category ?c.
        MINUS {
                ?c kebi:has_color kebi:color_white.
        }
}
```

Listing 4.23: All not white weak wines that have a pairing with the Strawberry Cheesecake

All Strong Wines Not from Italy paired with Fried Bass with Herbs

```
SELECT ?weak WHERE{
        kebi:meal_fried_bass_with_herbs kebi:is_ideal_weak ?c .
        ?weak kebi:has_category ?c.
        MINUS {
                ?weak kebi:is_wine_from_country kebi:country_italy
        }
}
```

Listing 4.24: All strong wines not from Italy that have a pairing with the Fried Bass with Herbs

5. Graphical Modelling Language

A graphical modelling language is a visual system used to represent and describe models, relationships, or concepts using graphical elements. Its purpose is to enable the creation of visual models that effectively capture complex ideas or systems, enhancing understanding, analysis, and communication. Through the utilization of graphical models, we are able to construct a representation of reality known as *conceptual modelling*. To begin with the model creation process, the initial step involves defining the *metamodel*, which describes concepts with attributes and rules that serve as a foundation for representing general knowledge pertaining to the domain. Subsequently, the *modelling language* specifies the notations applicable to the concepts outlined in the metamodel. In the case of graphical modelling, these notations revolve around visualizing the concepts effectively. By employing the modelling language, we can generate the model required to represent the desired real-world domain.

5.1 Adoxx

Adoxx [1] is a metamodelling tool, that enables users to define metamodels and create models, offering multiple features and functionalities for model development and management. The tool is divided into two main components:

- ADOxx Development Toolkit;
- ADOxx Modelling Toolkit.

The *ADOxx Development Toolkit*, functions as a platform for defining and managing modelling languages. The Toolkit allows users to define, create and customize libraries, as well as manage users, models, and components, this way, the modelling languages according to their specific needs and requirements. On the other hand, the *ADOxx Modelling Toolkit* allows users to handle the metamodels created in the development phase and generate models. This toolkit offers a rich set of modelling capabilities, allowing users to create detailed and accurate representations of their systems or processes.

5.2 Wine Decision Implementation

Our goal was to design a graphical modelling language, which allows a chef to represent meals and wines in a graphical way, such that it contains all information relevant for the customers to select according to their preferences. In order to start with the design and achieve the final graphical representation of the wine decision model, we have created a new library using the ADOxx Development Toolkit. Our library had the following specifications:

Username: admin

Password: password

Name: wine Decision Static & wine Decision Dynamic.

The wine decision library was used to create the classes with their own attributes and relations for our model.

Classes and Relations

In our library, we can distinguish two main classes: Meal and Wine.
The **Meal** class is used to represent a meal, its attributes are the *name* (String), and the *ingredients* that compose the meal (Enumeration). To represent the *ingredients* we have decided to utilize different icons to deliver a better graphical experience.

Figure 5.1: All the different Ingredients icon

The final representation of a **meal** class is as shown:

Figure 5.2: Meal class representation

The **Wine** class defines a wine, and its attributes are:

- **Name:** a string representing the name of the wine
- **Taste:** an enumeration, which possible values are Bold/Tannic
- **Tannin level:** an enumeration, which possible values are Tannic / Less tannic

- **Dryness:** an enumeration, which possible values are Dry / Not Dry
- **Category:** an enumeration, which possible values are Bold red/ Medium red / Light red / Rose / Rich white / Light white / Sparkling / Sweet White / Dessert.
- **Country:** an enumeration, which possible values are Italy / Spain / France / Germany. The countries are represented by their own flags.
- **Region:** an enumeration, which possible values are Andalusia / Tuscany / Piemonte / Rheinhessen / Rhône / Anjou-Saumur / Abruzzi / Bordeaux / Burgundy / Chateau d'Esclans / Aragona / Vosne-Romanée / Vaucluse
- **Grapes:** an enumeration with the possibility of choosing multiple values, which are Brunello / Sangiovese / Syrah / Grenache / Merlot / Cabernet Sauvignon / Cabernet Franc / Pinot noir / Carignan / Chardonnay Rose / Vermentino / Chardonnay / Marsanne / Trebbiano / Crémant / Riesling / Moscato / Palomino

We have decided to utilize different icons for each available wine category:

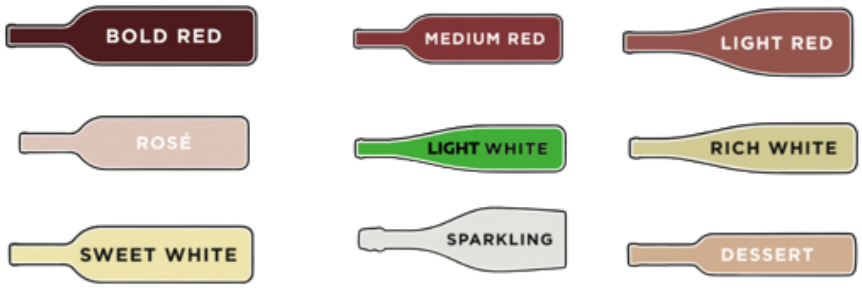


Figure 5.3: Possible wine category icons

We did the same, for visualizing all the possible *countries*:

Figure 5.4: Possible country icons

And also for visualizing the tannin level (tannic, less tannic), the dryness level (dry, not dry) and for a graphical representation of whether the wine as bold or light.

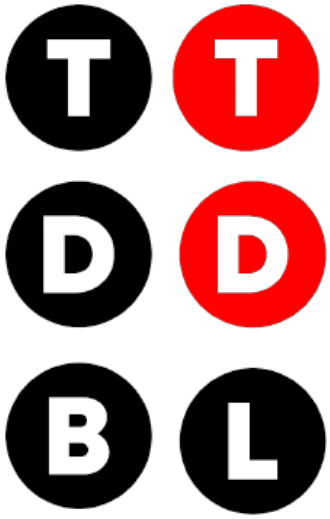


Figure 5.5: Tannic in a *black T*, less tannic in *red T*, followed by, dry in *black D* and less dry in *red D*. Lastly bold is represented with a *black B* and light with a *black L*

The final representation of a **wine** class is as follows:

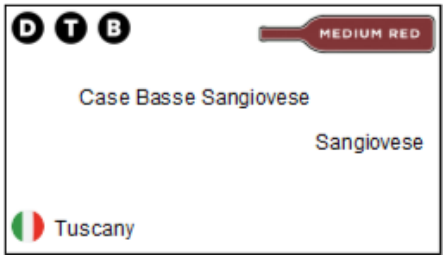


Figure 5.6: Wine class representation

Between the meal and wine class, there can be two different relationships, **Weak Ideal Wine** and **Strong Ideal Wine**. These relationships are used to join the wine with their respective ideal wine, as the relationship between the meal and wine can be either strong or weak, we have decided to use two different associations.

Figure 5.7: Possible country icons

Graphrep

Each class has a Graphrep attribute, which represents a long string capable of containing code. This code is later interpreted as a script by the Graphrep interpreter, allowing for the identification of various elements within it, including style, shape, variable assignment, context, and control. Through modifications made to the Graphrep code, we gain the ability to influence the visual representation of our classes and relationships within the model.

To define the **relationships** between meals and wine, we have used these two Graphrep codes:

```
GRAPHREP

SHADOW off
PEN color:red style:solid w:0.05cm
EDGE

START

MIDDLE
TEXT "Strong" x:0cm y:-0.6cm w:c

END
LINE x1:0.1cm y1:0.0cm x2:-0.2cm y2:0.1cm
LINE x1:0.1cm y1:0.0cm x2:-0.2cm y2:-0.1cm
```

Listing 5.1: Strong Ideal Wine relation Graphrep

```
GRAPHREP

SHADOW off
PEN color:blue style:solid w:0.05cm
EDGE

START

MIDDLE
TEXT "Weak" x:0cm y:-0.6cm w:c

END
LINE x1:0.1cm y1:0.0cm x2:-0.2cm y2:0.1cm
LINE x1:0.1cm y1:0.0cm x2:-0.2cm y2:-0.1cm
```

Listing 5.2: Weak Ideal Wine relation Graphrep

In order to define the **Meal** class we have developed this Graphrep code:

```
GRAPHREP

FILL color:gray
RECTANGLE x: -2cm y:-1cm w:7cm h:2.2cm
ATTR "Name" w:c:2.5cm x:1.5cm y:0.4cm h:c line-break:rigorous

AVAL ingredient_1: "ingredient_1"
AVAL ingredient_2: "ingredient_2"
....
IF (ingredient_1 = "Red meat")
  BITMAP "db:\\red_meat.png" x:-1.9cm y:-0.9cm w:0.7cm h:0.7cm
ELSIF (ingredient_1 = "Grilled")
  BITMAP "db:\\grill.png" x:-1.9cm y:-0.9cm w:0.7cm h:0.7cm
ELSIF (ingredient_1 = "Fish")
  BITMAP "db:\\fish.png" x:-1.9cm y:-0.9cm w:0.7cm h:0.7cm
...
```

Listing 5.3: Meal class Graphrep

For the **Wine** class we have used this Graphrep code:

```
GRAPHREP

FILL color:white
RECTANGLE x: -2cm y:-1cm w:7cm h:4cm
ATTR "Name" x:-0.8cm y:0.5cm h:c line-break:rigorous

AVAL dryness: "Dryness"
AVAL tannin_level: "Tannin Level"
AVAL taste: "Taste"
AVAL country: "Country"
AVAL region: "Region"
AVAL category: "Category"
AVAL grape_1: "Grape_1"
....
IF (dryness = "dry")
  BITMAP "db:\\dry.png" x:-1.9cm y:-0.9cm w:0.5cm h:0.5cm
ELSIF (dryness = "not dry")
  BITMAP "db:\\not_dry.png" x:-1.9cm y:-0.9cm w:0.5cm h:0.5cm
ENDIF
...
```

Listing 5.4: Wine class Graphrep

Final Result

As a final result, we have a graphical representation of the wines with their attributes, the meals with their ingredients, where they are interconnected by relationship defining which are the weak and strong ideal wines for that dish.

6. Testing

In this chapter, we have prepared some testing cases, in order to check how the different tools will evaluate their output.

Test 1

During this test, we will give the following inputs:

- **Meal:** Fried Bass with herbs
- **Grape to exclude:** Chardonnay
- **Tannin Level:** Less tannic

The expected output for this test is: **Trebbiano D'abruzzo**

Camunda

Figure 6.1: Camunda's output for test 1

Prolog

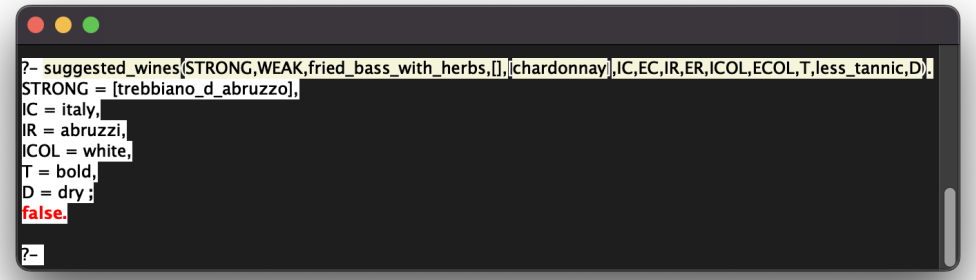


Figure 6.2: Prolog's output for test 1

Protégé

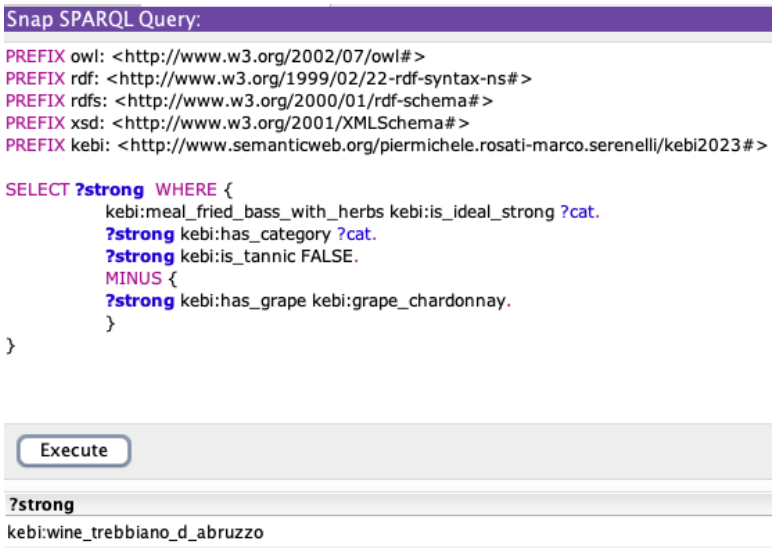


Figure 6.3: Protégé's output for test 1

Test 2

During this test, we will give the following inputs:

- **Meal:** Pulled pork
- **Tannin Level:** Tannic
- **Country:** Italy

The expected output for this test is: **Brunello di Montalcino & Case basse Sangiovese**

Camunda

Inputs:

Decision table: All tables

Dryness
Dryness_Level: Enter String — string

Tannin
Tannin_Level: tannic — string

Taste
Taste_Level: Enter String — string

Include_Country_Preferences
Country: italy — string

Include_Region_Preferences
Region: Enter String — string

Meal_Categories
Meal: pulled pork — string

Include_Grape_Preferences
Grape: Enter String — string

Exclude_Grape_Preferences
No_Grape: Enter String — string

Exclude_Region_Preferences
No_Region: Enter String — string

Wine_Color
Color: Enter String — string

Exclude_Country_Preferences
No_Country: Enter String — string

Outputs:

Dryness_Wines:	
Tannin_Level_Wines:	Brunello di Montalcino, Juline Châteauneuf-du-Pape, Pomerol Château de
Taste_Wines:	
Flavor_Wine_Preferences:	[Brunello di Montalcino, Juline Châteauneuf-du-Pape, Pomerol Château de
Wines:	
Geo_Wine_Preferences:	[Brunello di Montalcino, Case Basse Sangiovese, Trebbiano d'Abruzzo, Vaj
Include_Country_Preferences:	Brunello di Montalcino, Case Basse Sangiovese, Trebbiano d'Abruzzo, Vajr
Include_Region_Preferences:	
Ingredients:	[pork, black_pepper, white_starches, alliums, red_pepper, grilled, smoked]
Strong_Wine_Category:	[bold red, medium red]
Weak_Wine_Category:	[]
Meal_Strong_Wines:	Brunello di Montalcino, Juline Châteauneuf-du-Pape, Pomerol Château de
Suggested_Wines:	[Brunello di Montalcino, Case Basse Sangiovese]
Grape_Wine_Preferences:	[Brunello di Montalcino, Juline Châteauneuf-du-Pape, Pomerol Château de
Include_Grape_Preferences:	
Exclude_Grape_Preferences:	
Exclude_Region_Preferences:	
Region_Wine_Preferences:	[Brunello di Montalcino, Juline Châteauneuf-du-Pape, Pomerol Château de
Meal_Weak_Wines:	[]
Wine_Color:	[Brunello di Montalcino, Juline Châteauneuf-du-Pape, Pomerol Château de
Exclude_Country_Preferences:	
Country_Wine_Preferences:	[Brunello di Montalcino, Case Basse Sangiovese, Trebbiano d'Abruzzo, Vaj

Figure 6.4: Camunda's output for test 2

Prolog

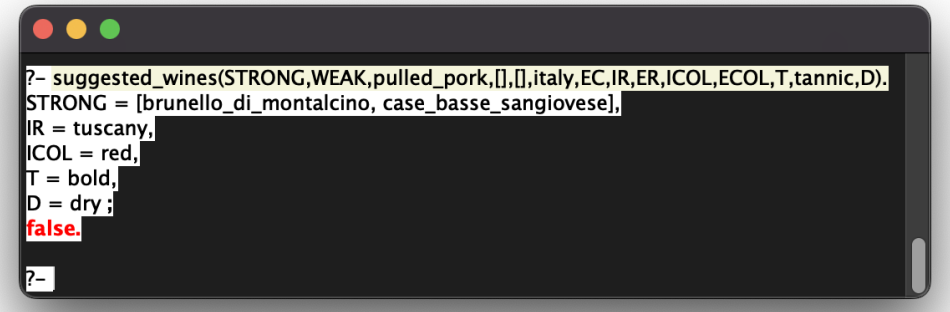


Figure 6.5: Prolog's output for test 2

Protégé

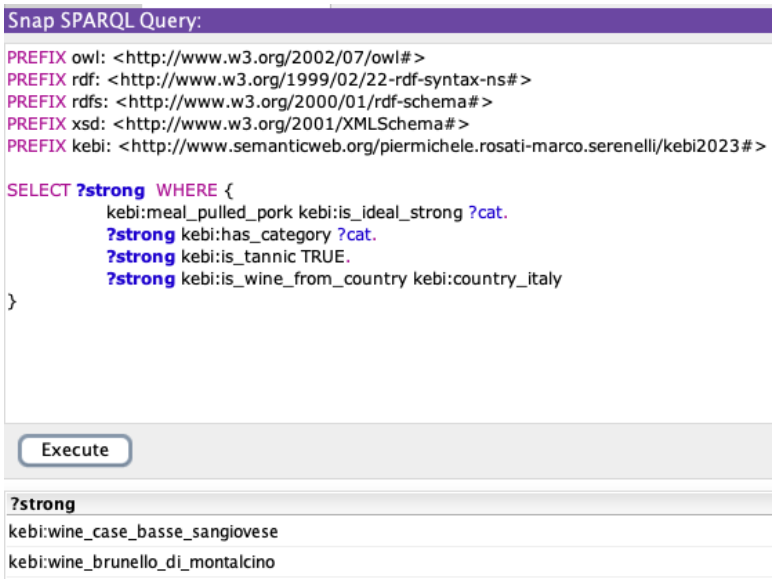


Figure 6.6: Protégé's output for test 2

Test 3

During this test, we will give the following inputs:

- **Meal:** Strawberry Cheesecake
- **Grapes to include:** Moscato, Syrah
- **Country:** Italy
- **Region:** Piemonte

The expected output for this test is: **Vajra Moscato d'Asti**

Camunda

Camunda does not support the inclusion of multiple grapes, therefore this test is not applicable.

Prolog

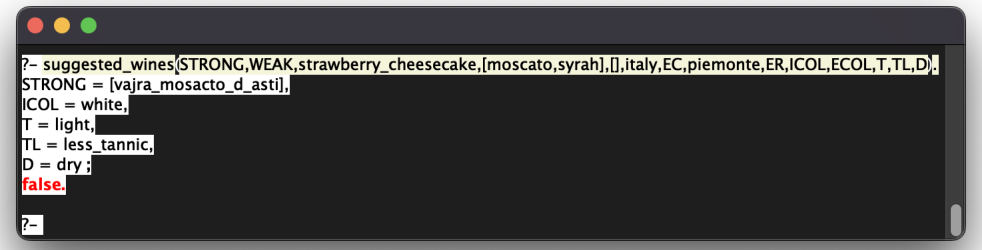


Figure 6.7: Prolog's output for test 3

Protégé

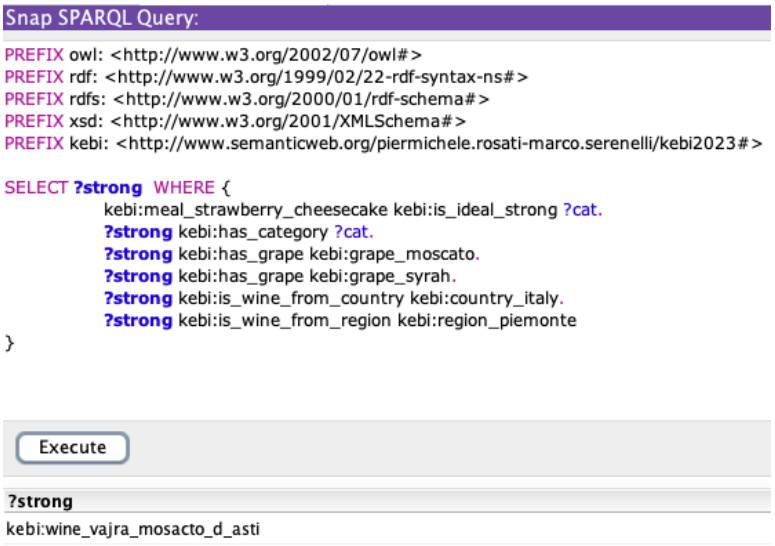


Figure 6.8: Protégé's output for test 3

Test 4

During this test, we will give the following inputs:

- **Meal:** Roast Chicken with Herbs
- **Grapes to include:** Merlot
- **Grapes to exclude:** Sangiovese, Carignan

The expected output for this test is: **Pomerol Château de Salesl**

Camunda

Camunda does not support the exclusion of multiple grapes, therefore this test is not applicable.

Prolog

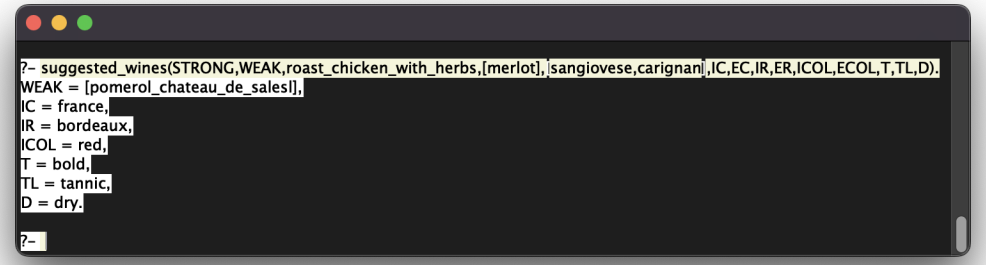


Figure 6.9: Prolog's output for test 4

Protégé

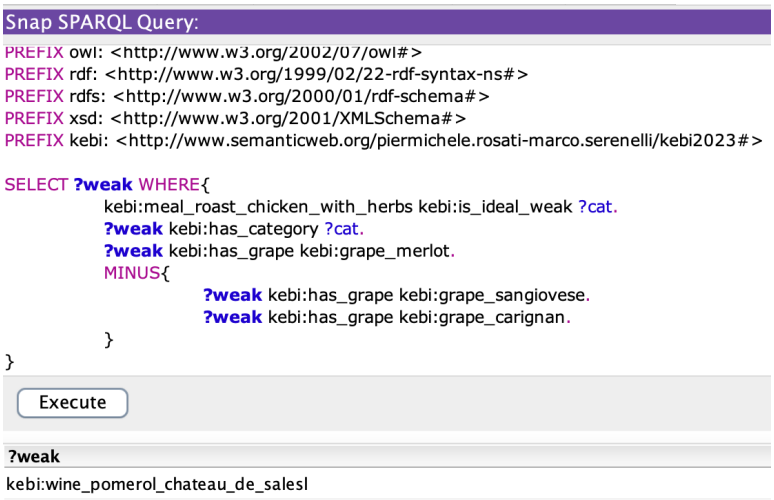


Figure 6.10: Protégé's output for test 4

Test 5

During this test, we will give the following inputs:

- **Meal:** Pumpkin Risotto
- **Country to exclude:** Italy
- **Region to exclude:** Burgundy

The expected output for this test is: **Garrus Rosé, Hermitage AOC, G-Max Reisling**

Camunda

Figure 6.11: Camunda's output for test 5

Prolog

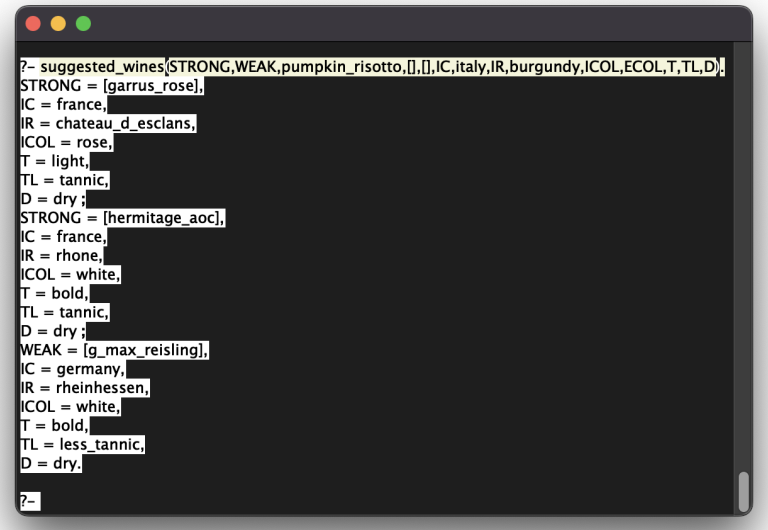


Figure 6.12: Prolog's output for test 5

Protégé

Snap SPARQL Query:

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX kebi: <http://www.semanticweb.org/piermichele.rosati-marco.serenelli/kebi2023#>

SELECT ?strong WHERE{
        kebi:meal_pumpkin_risotto kebi:is_ideal_strong ?cat.
        ?strong kebi:has_category ?cat.
        MINUS{
                ?strong kebi:is_wine_from_country kebi:country_italy.
        }
        MINUS{
                ?strong kebi:is_wine_from_region kebi:region_burgundy.
        }
}
```

Execute

?strong
kebi:wine_hermitage_aoc
kebi:wine_garrus_rose

Figure 6.13: Protégé's output for test 5, strong wines

Snap SPARQL Query:

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX kebi: <http://www.semanticweb.org/piermichele.rosati-marco.serenelli/kebi2023#>

SELECT ?weak WHERE{
        kebi:meal_pumpkin_risotto kebi:is_ideal_weak ?cat.
        ?weak kebi:has_category ?cat.
        MINUS{
                ?weak kebi:is_wine_from_country kebi:country_italy.
        }
        MINUS{
                ?weak kebi:is_wine_from_region kebi:region_burgundy.
        }
}
```

Execute

?weak
kebi:wine_g_max_reisling

Figure 6.14: Protégé's output for test 5, weak wines

Test 6

During this test, we will give the following inputs:

- **Meal:** Roast Chicken with Herbs
- **Country to include:** France
- **Region to include:** Tuscany

The expected output for this test is: **error because Tuscany is not a region of France.**

Camunda

Inputs:

Decision table: All tables

Dryness

Dryness_Level: Enter String — string

Tannin

Tannin_Level: Enter String — string

Taste

Taste_Level: Enter String — string

Include_Country_Preferences

Country: france — string

Include_Region_Preferences

Region: tuscany — string

Meal_Categories

Meal: roast chicken with herbs — string

Include_Grape_Preferences

Grape: Enter String — string

Exclude_Grape_Preferences

No_Grape: Enter String — string

Exclude_Region_Preferences

No_Region: Enter String — string

Wine_Color

Color: Enter String — string

Exclude_Country_Preferences

No_Country: Enter String — string

Outputs:

Dryness_Wines:	
Tannin_Level_Wines:	
Taste_Wines:	
Flavor_Wine_Preferences:	[Brunello di Montalcino, Juline Châteauneuf-du-Pape, Pomerol Château de
Wines:	
Geo_Wine_Preferences:	[]
Include_Country_Preferences:	Juline Châteauneuf-du-Pape, Pomerol Château de Salesl, Romanée-Saint,
Include_Region_Preferences:	Brunello di Montalcino, Case Basse Sangiovese, Occhio di pernice
Ingredients:	[poultry, herbs, roasted]
Strong_Wine_Category:	[light red]
Weak_Wine_Category:	[medium red, rose]
Meal_Strong_Wines:	Romanée-Saint, Aragone
Suggested_Wines:	ERROR: Country and Region don't match!
Grape_Wine_Preferences:	[Brunello di Montalcino, Juline Châteauneuf-du-Pape, Pomerol Château de
Include_Grape_Preferences:	
Exclude_Grape_Preferences:	
Exclude_Region_Preferences:	
Region_Wine_Preferences:	[Brunello di Montalcino, Case Basse Sangiovese, Occhio di pernice]
Meal_Weak_Wines:	Pomerol Château de Salesl, Case Basse Sangiovese, Marsannay, Garrus F
Wine_Color:	[Brunello di Montalcino, Juline Châteauneuf-du-Pape, Pomerol Château de
Exclude_Country_Preferences:	
Country_Wine_Preferences:	[Juline Châteauneuf-du-Pape, Pomerol Château de Salesl, Romanée-Saint

Figure 6.15: Camunda's output for test 6

Prolog

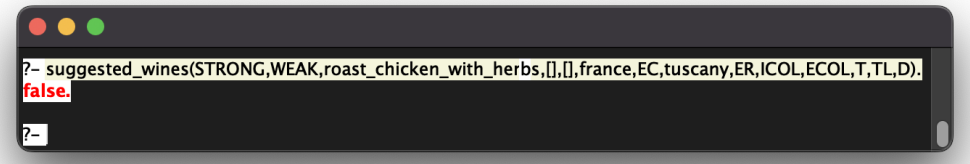


Figure 6.16: Prolog's output for test 6

Protégé

Snap SPARQL Query:

```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX kebi: <http://www.semanticweb.org/piermichele.rosati-marco.serenelli/kebi2023#>

SELECT ?strong WHERE{
        kebi:meal_roast_chicken_with_herbs kebi:is_ideal_strong ?cat.
        ?strong kebi:has_category ?cat.
                ?strong kebi:is_wine_from_country kebi:country_france.
                ?strong kebi:is_wine_from_region kebi:region_tuscany.
}
```

Execute

?strong

Figure 6.17: Protégé's output for test 6

7. Conclusions

7.1 Piermichele Rosati's Conclusion

In this section I will list the advantages and disadvantages of using each tool and pull my final conclusions on all the solutions of the different knowledge bases comparing them and tools.

Camunda

Pros	Cons
A lot of documentation is available.	To check if the model contains errors, the simulation must be carried out on the online simulator.
Camunda supports many programming languages including Java, Python and Javascript.	It is not possible to provide more than one input for the same input.
Intuitive tool for modelling one's knowledge base.	No shortcuts or features are provided to speed up modelling work. (e.g. smart copy/paste)
FEEL is a very powerful language and has a fairly simple syntax for the functions it provides.	FEEL functions provided are not many. If you have to do something advanced, you get to construct long and incomprehensible expressions.
Add food, wine or other elements is possible without employing essential changes to the decision tables.	-

Prolog

Pros	Cons
It is a very concise and intuitive language.	Defining complex predicates may require a top-down approach and this may take a long time.
It is used in AI, Machine Learning and for automation and advanced technology.	It can be a complex and confusing programming language for those who are new to programming and do not have good logic skills.
It is a declarative programming language.	There are no advanced data structures and a few native functions in the language. So as in our case, we have defined the operations to be performed with lists.
It has a built-in inference engine, backtracking: allows it to explore alternative solutions when the initially chosen path leads to failure. This can be advantageous for solving problems with multiple possible solutions or when the exact solution path is unclear.	Backtracking can be considered a big advantage but a not elegant output solution.
Adding facts and rules to our knowledge base is simple and does not require a high cost in terms of time.	-

Protégé

Pros	Cons
Highly intuitive graphical interface.	The tool sometimes has problems during use that sometimes cause errors to be made unintentionally.
Scalable and easily extensible ontology.	Defining all classes, relationships between classes and attributes takes a lot of time.
It has SPARQL for query execution.	SPARQL on some Protégé versions does not work. We have used the Snap SPARQL plugin. This tool within Protégé is not smart (it does not even have undo/redo writing queries).
It has the reasoner technology.	Since it is difficult to make distraction errors, the reasoner may not infect the SW rules well and fail to understand where the error comes from.

ADOxx

Pros	Cons
The ADOxx Graphrep Repository proves to be a valuable resource for accessing and examining different Greaphrep representations contributed by other individuals.	The official documentation on ADOxx does not cover every aspect of the tool, it lacks extensive written content, sometimes is also outdated and provides few to no examples.
The ability to upload and use icons in the model offers a wide range of customization, which can be very useful to better clarify the graphical model.	When using ADOxx to create simple models, sometimes the effort invested may not produce proportional returns, as it may take a lot of time to define the different classes.
The classes created in the libraries can be easily modified, allowing you to add new attributes or change their graphical representation without the need to delete and recreate them.	The learning curve associated with this tool is very steep, requiring a considerable amount of time and effort to become proficient with its functionalities and features.

Final Conclusion

All these tools are very powerful and easy to use, but each has its own advantages and disadvantages. Below, some experiences using the tools and a comparison between them are described.

- **Protégé** is a tool that can open random work areas during its use: in my personal experience, while adding, for example, an object property, while I was specifying the domain, the tool showed me another area, for example a data property.
  Another problem encountered during development was with the reasoner and SW rules: for reasons unknown to me, when I entered new rules and saved the rdf file, sometimes the rules were no present anymore and this led to incomprehensible errors during inferences and queries.
  One problem I had with the reasoner was that when it is stopped and a SPARQL query is started, the query will not work: the reason is that the reasoner must be set to "None" when you do not want to use one of those available. Unfortunately I understood this by trial and error, as it is not mentioned anywhere.

- About **Camunda**, on the other hand, it is much more stable and fluid software than Protégé. Unfortunately, if you need to add data to the knowledge base or set up the knowledge base, Camunda does not provide the functions to do this quickly: you have to add, remove, edit everything manually. In my personal experience, since the dmn file is a format similar to xml, I opened the dmn with an editor and copied all the grape names separated by commas and then pasted it into camunda in order not to enter one grape at a time by hand and thus to speed up the process for other data as well.
  Another feature that would have been very useful to me is a kind of hint while using FEEL expressions. Each time I entered them by hand following the official documentation, but I had many errors when testing the simulator. Not having a syntax highlighter is not difficult to get errors where commas, brackets and so

on are missing. Define FEEL expressions to determine intersections between lists was quite inconvenient for this reason.

- **Prolog** is a programming language that allowed us to define our knowledge base in a simple, fast and elegant way at the same time.
  Obviously, we also had some problems with Prolog: for example, backtracking would give us the same wine several times, as each wine has one or more grapes. So if a wine has N grapes, the output for that wine was repeated N times. Fortunately, the language provides so-called built-in functions, which are nothing more than native Prolog functions useful for standard purposes, such as obtaining the size of a list, or the fantastic setof and bagof, which we used a lot as explained in the Chapter 3. We solved the backtracking problem using setof and bagof.
  Prolog provides an extensive documentation but the online simulator can sometimes give unnecessary warnings. In my case, sometimes I had random warnings and the query correctly ran. Moreover, when I selected 10 or 100 outputs in the simulator, it was blocking and I had to abort it losing my code changes. In order to avoid seeing these warnings and lose my code changes, I used the local simulator, which works better than the online one.

- **ADOxx** is a metamodelling tool that empowers users to define metamodels and create models. In my experience, I encountered difficulties when initially navigating its graphical user interface, which was rather underwhelming at first. However, after becoming used to it, the software allows for the creation of visually appealing models.
  What I found particularly fascinating was the Graphrep repository, which includes various user-generated Graphrep examples that proved to be highly beneficial for initiating our own projects. The ability to upload icons significantly improved the visibility of our model. I found this feature particularly useful, we have used it to add icons for the different wine categories, ingredients of the meals, country icons, and a wine's characteristics such as tannic, dry or bold. Overall, I think that the usage of icons facilitates easier comprehension of the model and highlights the most pertinent information.

Making a comparison between these softwares, the one I liked the most to use was Prolog, because although it took a long time and a lot of reasoning to arrive at the final solution, it is a declarative programming language, and therefore convenient for declaring and change rules and facts. On the other hand, in Camunda every time we made changes to the knowledge base we add tables or rows and is required nontrivial time to do this and then we had to test it online and getting errors we had to re-edit it again locally and retested it online until we got no errors. Anyway, using it throughout the project we were able to understand how to optimise the rows of the decision tables well and how to harness the power of FEEL to best suit our needs.
As for Protégé, it was interesting to study the theory of knowledge graphs but it was not pleasant to use the tool as it showed some problems in use: for example, it was not possible to use SPARQL but we had to use the Snap SPARQL tool to reasons unknown to us. Despite that, I think Protégé and Camunda are very suitable for modelling one's knowledge base in an intuitive and effective way. Since ADOxx is not very scalable, I do not recommend it for large and important projects.
Finally, we had the possibility to model the knowledge base of wines in several ways: we decided to model it in a complete way using Prolog, inferring the strong and weak wines from the all ingredients compose a meal. In Camunda and Protégé we modelled

it reflecting reality as much as possible within the limits of the functionality of these tools.

7.2 Marco Serenelli's Conclusion

Within this section, I will recap my encounter with the tools and elaborate on the strengths and weaknesses that I discovered while utilizing them.

Camunda

Pros	Cons
The software comes with a user-friendly and intuitive graphical user interface	It lacks some basic features, for example, a built-in way to provide inputs and follow the execution of the code
The inclusion of FEEL language expressions within the tables offers a wide range of potential combinations that can be performed within them.	Debugging is complicated, as the Camunda doesn't say exactly where the error occurred or gives just some vague and hard-to-understand error messages.
By leveraging the option to utilize the output of one decision table as an input for another, it becomes possible to construct intricate decision-making processes	There is no possibility of applying any kind of control or validation to the inputs provided by the users
-	Expanding the knowledge base of the system may be time-consuming, as you need to create new rows in every table where the input is used

Prolog

Pros	Cons
The language is declarative	At the same timne, due to its declarative nature, in case of errors, debugging the code may be a tricky task
Adding new elements to the knowledge base is as simple as just adding a new line of code, without the need of modifying predicates.	It lacks an intuitive frontend, where the user can easily insert meals and other elements as inputs, resulting in a bad stand-alone application.
Developing a rule-based system in Prolog is very simple, and can be done with few lines of code	-

Protégé

Pros	Cons
Protégé comes with a great and easy-to-use GUI	The SWRL language, sometimes, provides some limitations
Protégé supports SPARQL queries	The documentation found online lacks some basic features and sometimes it can be hard to understand
Protégé's reasoner can be very useful for logical inference and consistency checking within the ontology.	I had a hard time setting up the reasoner in my system, mainly due to some update issues with the software

ADOxx

My considerations about conceptual modelling with *ADOxx*.

Pros	Cons
The Graphical model given as a result is very useful and simple, allowing people with a limited technical background to easily understand	Having to constantly switch between the development toolkit and the modelling toolkit to update any modifications takes up a significant amount of time
The Graphrep language offers an extensive range of customization options when it comes to graphically representing the model	The GUI provided in the tool for coding in Graphrep lacks a lot of basic functionality and features
Modelling Components, libraries, and graphical templates can be easily shared and reused, leading to the creation of communities and repositories where users share their templates and components.	I have faced different problems with the installation of the tool in computers with an operating system different than AMD64 Windows

Final Conclusion

Overall, I think that the tools are very useful, and each one is more suitable for a specific task with respect to another.

- **Camunda** is an open-source platform for decision modelling. Even though the software is quite user-friendly, I did not really enjoy working with it because I discovered it lacked some basic features. For instance, it didn't offer the ability to test the tables. Instead, every time I made a change to the tables, I had to upload the file to their website for execution, which became quite tedious over time. However, the option to enhance its capabilities through code was a game-changer for me. The software can understand and use the FEEL language expressions, which were a bit hard to understand at first but then we found them to be extremely useful for our objectives, especially since we weren't fond of the traditional approach of manually populating lengthy tables. I believe this software

could prove highly valuable when used for backend purposes, as it can also be integrated with Java and other programming languages.

- In my opinion, **Prolog** proved to be a highly rewarding programming language. Using Prolog, we were able to construct our knowledge base with remarkable efficiency, requiring only a minimal number of code lines. As we tackled more intricate tasks, we frequently consulted its documentation, which not only proved to be extremely accurate but also very reader-friendly. Prolog comes equipped with an array of powerful built-in functions, notably including *setof* and *bagof*, which emerged as our go-to tools for resolving challenges associated with backtracking. While initially, our experience with Prolog was pretty straightforward, things took a turn when we tackled more complex operations. The frontend interface proved to be less than helpful, with its unintuitive design and overall poor performance. To further complicate matters, it occasionally presented strange warnings that left us perplexed, although our queries continued to function. Additionally, we encountered persistent issues with backtracking, leading to multiple duplicate outputs, which posed an ongoing challenge.

- The use of **Protégé** left us with different challenges. The main problem I have faced was that, sometimes, while using the software and saving some work done it just did not save, specifically while developing rules, leading to errors the next time I opened it. Another major issue for me was setting up the reasoner, I had to update the software and its components at different times but still had a hard time to properly make it work. However, the reasoner proved to be very useful for making inferences through ontology. Overall my experience with the software was not pretty good at first, even if it had a nice and easy-to-use GUI it looked too basic for me and lacked some features.

- Among the three software options, I've found **ADOxx** to be the most engaging. My preference stems from my appreciation of working with graphics and user interfaces, and ADOxx closely aligns with this interest. The ability to add icons to the models is a valuable feature that enhances the visual appeal of the model and provides greater clarity regarding what to focus on. I enjoyed working with the Graphrep language and used it to improve our model. This language allows us to customize the graphics as needed. For instance, we utilized it to create custom rectangles for our classes and position icons and text precisely where we wanted. Initially, adapting to their " coordinate system" felt challenging, leading to several trial-and-error attempts to correctly position elements inside the class rectangle. However, I encountered some difficulties while working with the tool, primarily because I had to constantly switch between the development toolkit and the modelling toolkit to update changes, which resulted in a significant waste of time. Lastly, the primary issue I encountered was during the software installation process. On my Macbook M2, which runs on ARM architecture, I was unable to install and run the software. Consequently, I had to switch to a Windows AMD64 machine.

Bibliography

[1] ADOxx. *"ADOxx"*.
Url: https://www.adoxx.org/live/home.

[2] Camunda. *"Camunda"*.
Url: https://camunda.com/.

[3] Wine Folly. *"Food and Wine Pairing Basics (Start Here!)"*
Url: https://winefolly.com/wine-pairing/getting-started-with-food-and-wine-pairing/.

[4] OMG. *"Object Management Group"*.
Url: https://www.omg.org/.

[5] Protégé. *"Protégé"*.
Url: https://protege.stanford.edu/.

[6] Wikipedia. *"Decision Model and Notation"*.
Url: https://en.wikipedia.org/wiki/Decision_Model_and_Notation.

[7] Wikipedia. *"Decision table"*.
Url: https://en.wikipedia.org/wiki/Decision_table.

[8] Wikipedia. *"List of grape varieties"*.
Url: https://en.wikipedia.org/wiki/List_of_grape_varieties.

[9] Wikipedia. *"Ontology Engineering"*.
Url: https://en.wikipedia.org/wiki/Ontology_engineering.

[10] Wikipedia. *"Prolog"*.
Url: https://it.wikipedia.org/wiki/Prolog.

[11] Wikipedia. *"Resource Description Framework"*.
Url: https://en.wikipedia.org/wiki/Resource_Description_Framework.

[12] Wikipedia. *"Rule-based system"*.
Url: https://en.wikipedia.org/wiki/Rule-based_system.

[13] Wikipedia. *"Semantic Web Rule Language"*.
Url: https://en.wikipedia.org/wiki/Semantic_Web_Rule_Language.

[14] Wikipedia. *"SPARQL"*.
Url: https://it.wikipedia.org/wiki/SPARQL.

List of Figures

1.1 Intersection between the ideal wine categories of ingredients Red Meat and Grilled, which are present in the Sliced Beef Meal. In red are shown the strong ideal wine categories and in blue the weaks 6

2.1 Our Model in Camunda 12
2.2 Wines Literal Expression 14
2.3 Dryness Dec. Table 14
2.4 Tannin Dec. Table 15
2.5 Taste Dec. Table 15
2.6 Include Country Dec. Table 15
2.7 Exclude Country Dec. Table 15
2.8 Include Region Dec. Table 16
2.9 Exclude Region Dec. Table 16
2.10 Include Grape Dec. Table 16
2.11 Exclude Grape Dec. Table 16
2.12 Wine Color Dec. Table 17
2.13 Meal Categories Dec. Table 17
2.14 Strong Wines Dec. Table 18
2.15 Weak Wines Dec. Table 18
2.16 Intersection between Dryness, Tannin and Taste input lists 19
2.17 Flavor Wine Preferences Dec. Table 20
2.18 Splitted Flavor Wine Dec. Table 20
2.19 Country Wine Dec. Table 21
2.20 Region Wine Dec. Table 21
2.21 Geographic Wine Preferences Dec. Table 22
2.22 Grape Wine Preferences Dec. Table 22
2.23 Suggested Wines Dec. Table 23
2.24 I/O Camunda Example 24

3.1 Prolog query example 38
3.2 Prolog query example without using setof 38

4.1 Our Ontology Graph in Protégé 40
4.2 Our Model in Protégé 41
4.3 Data properties 41

4.4 Object properties 42

5.1 All the different Ingredients icon 50
5.2 Meal class representation 50
5.3 Possible wine category icons 51
5.4 Possible country icons 51
5.5 Tannic in a *black T*, less tannic in *red T*, followed by, dry in *black D* and less dry in *red D*. Lastly bold is represented with a *black B* and light with a *black L* 52
5.6 Wine class representation 52
5.7 Possible country icons 52

6.1 Camunda's output for test 1 55
6.2 Prolog's output for test 1 56
6.3 Protégé's output for test 1 56
6.4 Camunda's output for test 2 57
6.5 Prolog's output for test 2 57
6.6 Protégé's output for test 2 58
6.7 Prolog's output for test 3 59
6.8 Protégé's output for test 3 59
6.9 Prolog's output for test 4 60
6.10 Protégé's output for test 4 60
6.11 Camunda's output for test 5 61
6.12 Prolog's output for test 5 61
6.13 Protégé's output for test 5, strong wines 62
6.14 Protégé's output for test 5, weak wines 62
6.15 Camunda's output for test 6 63
6.16 Prolog's output for test 6 63
6.17 Protégé's output for test 6 64

Listings

3.1 List operations 26
3.2 Countries and regions 27
3.3 Regions of country 27
3.4 Grapes 28
3.5 Wine colors and categories 28
3.6 Wine color of wine categories 29
3.7 Flavor wine preferences 29
3.8 Wines 29
3.9 Grapes of wines 30
3.10 Wine's origin 30
3.11 Wine categories 31
3.12 Wine taste 31
3.13 Wine tannin level 31
3.14 Wine dryness 31
3.15 Ingredients 32
3.16 Meals 32
3.17 Ingredients of each meal 32
3.18 Strong categories of ingredients 33
3.19 Weak categories of ingredients 33
3.20 Wine checking 33
3.21 Strong/weak wine category 34
3.22 Strong/weak wine categories using bagof 34
3.23 Strong wine categories of a meal 35
3.24 Weak wine categories of a meal 35
3.25 Ideal strong/weak wine 36
3.26 Region belongs to a country 36
3.27 Exclude wine by country 36
3.28 Exclude wine by region 36
3.29 Include/Exclude wine by grapes 37
3.30 Suggested general wines 37
3.31 Suggested strong and weak wines 37
3.32 Suggested wines 37
4.1 All wines in the knowledge base 43
4.2 All meals in the knowledge base 43

4.3 All red wines . . . 44
4.4 All rosé wines . . . 44
4.5 All white wines . . . 44
4.6 All dried wines . . . 44
4.7 All not dried wines . . . 44
4.8 All tannic wines . . . 45
4.9 All not tannic wines . . . 45
4.10 All bold wines . . . 45
4.11 All light wines . . . 45
4.12 All regions that belong the corresponding country . . . 45
4.13 All wines that belong the corresponding country . . . 45
4.14 All wines that belong the corresponding region . . . 46
4.15 All strong wines that have a pairing with the Grilled Bass with Herbs . . . 46
4.16 All weak wines that have a pairing with the Grilled Bass with Herbs . . . 46
4.17 All wines that have a pairing with the Pulled Pork . . . 46
4.18 All wines that have a pairing with the Pumpkin Risotto . . . 46
4.19 All wines that have a pairing with the Pumpkin Risotto . . . 47
4.20 All wines that have a pairing with the Roast Chicken with Herbs . . . 47
4.21 All wines that have a pairing with the Sliced Beef . . . 47
4.22 All wines that have a pairing with the Strawberry Cheesecake . . . 47
4.23 All not white weak wines that have a pairing with the Strawberry Cheesecake . . . 48
4.24 All strong wines not from Italy that have a pairing with the Fried Bass with Herbs . . . 48
5.1 Strong Ideal Wine relation Graphrep . . . 53
5.2 Weak Ideal Wine relation Graphrep . . . 53
5.3 Meal class Graphrep . . . 54
5.4 Wine class Graphrep . . . 54
7.1 Our Knowledge Base and Wine Pairing in Prolog . . . 79

Prolog Code

```
% list operations
% * list membership (utility for grape inclusion/exclusion)
is_in_list(H,[H|_]).
is_in_list(H,[_|T]) :- is_in_list(H,T).
list_intersection([], _, []).
list_intersection([H|T], L2, [H|L1]) :- is_in_list(H,L2), !, list_intersection(T,L2,L1).
list_intersection([_|T],L2,L1) :- list_intersection(T,L2,L1).
% * concatenation between lists including duplicates (utility for meal derivation from ingredients)
list_concat([],L,L).
list_concat([X1|L1],L2,[X1|L3]) :- list_concat(L1,L2,L3).
% * count list occurrencies (utility for meal derivation from ingredients)
list_count([],_H,0). %_H to suppress Singleton Warning
list_count([H|T],H,C):- list_count(T,H,O), C is 1+O.
list_count([H1|T],H,C):- H1\=H, list_count(T,H,C).
list_countall(L,H,C) :-
   is_in_list(H,L),
   list_count(L,H,C).

% countries
country(france).
country(germany).
country(italy).
country(spain).

% regions
region(abruzzi).
region(anjou-saumur).
region(aragona).
region(andalusia).
region(bordeaux).
region(burgundy).
region(chateau_d_esclans).
region(piemonte).
region(rheinhessen).
region(rhone).
region(tuscany).
region(vaucluse).
region(vosne_romanee).

% country regions
region_of(france, anjou-saumur).
region_of(france, bordeaux).
region_of(france, burgundy).
region_of(france, chateau_d_esclans).
region_of(france, rhone).
region_of(france, vaucluse).
region_of(france, vosne_romanee).
region_of(germany, rheinhessen).
```

```
region_of(italy, abruzzi).
region_of(italy, friuli).
region_of(italy, piemonte).
region_of(italy, tuscany).
region_of(spain, andalusia).
region_of(spain, aragona).

% grapes
grape(brunello).
grape(cabernet_franc).
grape(cabernet_sauvignon).
grape(carignan).
grape(chardonnay).
grape(chardonnay_rose).
grape(cremant).
grape(grenache).
grape(marsanne).
grape(merlot).
grape(moscato).
grape(palomino).
grape(pinot_noir).
grape(riesling).
grape(sangiovese).
grape(syrah).
grape(trebbiano).
grape(vermentino).

% wine colors
wine_color(red).
wine_color(rose).
wine_color(white).

% wine categories
wine_category(bold_red).
wine_category(medium_red).
wine_category(light_red).
wine_category(rose_).
wine_category(rich_white).
wine_category(light_white).
wine_category(sparkling).
wine_category(sweet_white).
wine_category(dessert).

% wine color categories
wine_color_of(bold_red, red).
wine_color_of(medium_red, red).
wine_color_of(light_red, red).
wine_color_of(rose_, rose).
wine_color_of(rich_white, white).
wine_color_of(light_white, white).
wine_color_of(sparkling, white).
wine_color_of(sweet_white, white).
wine_color_of(dessert, red).

% tastes
taste(bold).
taste(light).

% tannin levels
tannin_level(tannic).
tannin_level(less_tannic).
```

```
% dryness
dry_level(dry).
dry_level(not_dry).

% wines
wine(aragone).
wine(brunello_di_montalcino).
wine(case_basse_sangiovese).
wine(corton_charlemagne_grand_cru).
wine(coteau_de_la_beylesse).
wine(cremant_d_alsace).
wine(fino).
wine(g_max_reisling).
wine(garrus_rose).
wine(hermitage_aoc).
wine(juline_chateauneuf_du_pape).
wine(marsannay).
wine(occhio_di_pernice).
wine(pomerol_chateau_de_salesl).
wine(romanee_saint).
wine(sauternes).
wine(trebbiano_d_abruzzo).
wine(vajra_mosacto_d_asti).

% wine grapes
grapes_of(aragone, [carignan]).
grapes_of(brunello_di_montalcino, [brunello, sangiovese]).
grapes_of(case_basse_sangiovese, [sangiovese]).
grapes_of(corton_charlemagne_grand_cru, [chardonnay]).
grapes_of(coteau_de_la_beylesse, [marsanne]).
grapes_of(cremant_d_alsace, [cremant]).
grapes_of(fino, [palomino]).
grapes_of(g_max_reisling, [riesling]).
grapes_of(garrus_rose, [vermentino]).
grapes_of(hermitage_aoc, [marsanne]).
grapes_of(juline_chateauneuf_du_pape, [grenache, syrah]).
grapes_of(marsannay, [chardonnay_rose]).
grapes_of(occhio_di_pernice, [sangiovese, syrah]).
grapes_of(pomerol_chateau_de_salesl, [cabernet_franc, cabernet_sauvignon, merlot]).
grapes_of(romanee_saint, [pinot_noir]).
grapes_of(sauternes, [chardonnay]).
grapes_of(trebbiano_d_abruzzo, [trebbiano]).
grapes_of(vajra_mosacto_d_asti, [moscato]).

% wine regions
wine_from(aragona, aragone).
wine_from(tuscany, brunello_di_montalcino).
wine_from(tuscany, case_basse_sangiovese).
wine_from(burgundy, corton_charlemagne_grand_cru).
wine_from(rhone, coteau_de_la_beylesse).
wine_from(anjou-saumur, cremant_d_alsace).
wine_from(andalusia, fino).
wine_from(rheinhessen, g_max_reisling).
wine_from(chateau_d_esclans, garrus_rose).
wine_from(rhone, hermitage_aoc).
wine_from(vaucluse, juline_chateauneuf_du_pape).
wine_from(burgundy, marsannay).
wine_from(tuscany, occhio_di_pernice).
wine_from(bordeaux, pomerol_chateau_de_salesl).
wine_from(vosne_romanee, romanee_saint).
```

```
wine_from(bordeaux, sauternes).
wine_from(abruzzi, trebbiano_d_abruzzo).
wine_from(piemonte, vajra_mosacto_d_asti).

% wine categories
category_of(brunello_di_montalcino, bold_red).
category_of(juline_chateauneuf_du_pape, bold_red).
category_of(case_basse_sangiovese, medium_red).
category_of(pomerol_chateau_de_salesl, medium_red).
category_of(aragone, light_red).
category_of(romanee_saint, light_red).
category_of(garrus_rose, rose_).
category_of(marsannay, rose_).
category_of(corton_charlemagne_grand_cru, rich_white).
category_of(hermitage_aoc, rich_white).
category_of(sauternes, light_white).
category_of(trebbiano_d_abruzzo, light_white).
category_of(coteau_de_la_beylesse, sparkling).
category_of(cremant_d_alsace, sparkling).
category_of(g_max_reisling, sweet_white).
category_of(vajra_mosacto_d_asti, sweet_white).
category_of(fino, dessert).
category_of(occhio_di_pernice, dessert).

% wine tastes
taste_of(aragone, bold).
taste_of(brunello_di_montalcino, bold).
taste_of(case_basse_sangiovese, bold).
taste_of(corton_charlemagne_grand_cru, light).
taste_of(coteau_de_la_beylesse, bold).
taste_of(cremant_d_alsace, bold).
taste_of(fino, bold).
taste_of(g_max_reisling, bold).
taste_of(garrus_rose, light).
taste_of(hermitage_aoc, bold).
taste_of(juline_chateauneuf_du_pape, bold).
taste_of(marsannay, light).
taste_of(occhio_di_pernice, light).
taste_of(pomerol_chateau_de_salesl, bold).
taste_of(romanee_saint, bold).
taste_of(sauternes, bold).
taste_of(trebbiano_d_abruzzo, bold).
taste_of(vajra_mosacto_d_asti, light).

% wine tannin levels
tannin_of(aragone, tannic).
tannin_of(brunello_di_montalcino, tannic).
tannin_of(case_basse_sangiovese, tannic).
tannin_of(corton_charlemagne_grand_cru, tannic).
tannin_of(coteau_de_la_beylesse, tannic).
tannin_of(cremant_d_alsace, less_tannic).
tannin_of(fino, less_tannic).
tannin_of(g_max_reisling, less_tannic).
tannin_of(garrus_rose, tannic).
tannin_of(hermitage_aoc, tannic).
tannin_of(juline_chateauneuf_du_pape, tannic).
tannin_of(marsannay, tannic).
tannin_of(occhio_di_pernice, tannic).
tannin_of(pomerol_chateau_de_salesl, tannic).
tannin_of(romanee_saint, tannic).
tannin_of(sauternes, less_tannic).
```

```
tannin_of(trebbiano_d_abruzzo, less_tannic).
tannin_of(vajra_mosacto_d_asti, less_tannic).

% wine drynesses
dryness_of(aragone, dry).
dryness_of(brunello_di_montalcino, dry).
dryness_of(case_basse_sangiovese, dry).
dryness_of(corton_charlemagne_grand_cru, dry).
dryness_of(coteau_de_la_beylesse, dry).
dryness_of(cremant_d_alsace, dry).
dryness_of(fino, dry).
dryness_of(g_max_reisling, dry).
dryness_of(garrus_rose, dry).
dryness_of(hermitage_aoc, dry).
dryness_of(juline_chateauneuf_du_pape, dry).
dryness_of(marsannay, not_dry).
dryness_of(occhio_di_pernice, not_dry).
dryness_of(pomerol_chateau_de_salesl, dry).
dryness_of(romanee_saint, dry).
dryness_of(sauternes, not_dry).
dryness_of(trebbiano_d_abruzzo, dry).
dryness_of(vajra_mosacto_d_asti, dry).

% ingredients
ingredient(fish).
ingredient(herbs).
ingredient(pork).
ingredient(saulted_or_fried).
ingredient(black_pepper).
ingredient(white_starches).
ingredient(alliums).
ingredient(red_pepper).
ingredient(grilled).
ingredient(smoked).
ingredient(red_meat).
ingredient(root_vegetables_and_squash).
ingredient(soft_cheese_and_cream).
ingredient(poultry).
ingredient(roasted).
ingredient(fruit_and_berries).

% meals
meal(fried_bass_with_herbs).
meal(pulled_pork).
meal(pumpkin_risotto).
meal(roast_chicken_with_herbs).
meal(sliced_beef).
meal(strawberry_cheesecake).
meal(pumpkin_risotto).

% meal ingredients
ingredient_of(fried_bass_with_herbs, [fish, herbs, saulted_or_fried]).
ingredient_of(pulled_pork, [pork, black_pepper, white_starches, alliums, red_pepper, grilled, smoked]).
ingredient_of(sliced_beef, [red_meat, grilled]).
ingredient_of(pumpkin_risotto, [white_starches, root_vegetables_and_squash, soft_cheese_and_cream]).
ingredient_of(roast_chicken_with_herbs, [poultry, herbs, roasted]).
ingredient_of(strawberry_cheesecake, [fruit_and_berries]).

% ingredients strong categories
strong_category_of(fish, light_white).
strong_category_of(herbs, light_white).
```

```
strong_category_of(saulted_or_fried, light_red).
strong_category_of(black_pepper, bold_red).
strong_category_of(pork, medium_red).
strong_category_of(red_pepper, medium_red).
strong_category_of(alliums, medium_red).
strong_category_of(grilled, bold_red).
strong_category_of(smoked, medium_red).
strong_category_of(red_meat, bold_red).
strong_category_of(root_vegetables_and_squash, rose_).
strong_category_of(soft_cheese_and_cream, light_red).
strong_category_of(soft_cheese_and_cream, rich_white).
strong_category_of(poultry, light_red).
strong_category_of(poultry, rich_white).
strong_category_of(roasted, bold_red).
strong_category_of(fruit_and_berries, sweet_white).

% ingredients weak categories
weak_category_of(fish, rich_white).
weak_category_of(fish, sparkling).
weak_category_of(herbs, rich_white).
weak_category_of(herbs, rose_).
weak_category_of(herbs, light_red).
weak_category_of(herbs, medium_red).
weak_category_of(saulted_or_fried, rose_).
weak_category_of(saulted_or_fried, rich_white).
weak_category_of(saulted_or_fried, light_white).
weak_category_of(saulted_or_fried, sparkling).
weak_category_of(pork, bold_red).
weak_category_of(pork, rose_).
weak_category_of(pork, sparkling).
weak_category_of(black_pepper, medium_red).
weak_category_of(white_starches, bold_red).
weak_category_of(white_starches, medium_red).
weak_category_of(white_starches, light_red).
weak_category_of(white_starches, rose_).
weak_category_of(white_starches, rich_white).
weak_category_of(white_starches, light_white).
weak_category_of(white_starches, sparkling).
weak_category_of(white_starches, sweet_white).
weak_category_of(white_starches, dessert).
weak_category_of(alliums, bold_red).
weak_category_of(alliums, light_red).
weak_category_of(alliums, rose_).
weak_category_of(alliums, rich_white).
weak_category_of(alliums, light_white).
weak_category_of(alliums, sparkling).
weak_category_of(alliums, sweet_white).
weak_category_of(red_pepper, bold_red).
weak_category_of(red_pepper, rose_).
weak_category_of(red_pepper, light_white).
weak_category_of(red_pepper, sparkling).
weak_category_of(red_pepper, sweet_white).
weak_category_of(grilled, medium_red).
weak_category_of(grilled, light_red).
weak_category_of(grilled, sparkling).
weak_category_of(grilled, sweet_white).
weak_category_of(smoked, bold_red).
weak_category_of(smoked, light_red).
weak_category_of(smoked, rose_).
weak_category_of(smoked, sparkling).
weak_category_of(smoked, dessert).
```

```
weak_category_of(red_meat, medium_red).
weak_category_of(root_vegetables_and_squash, rich_white).
weak_category_of(root_vegetables_and_squash, sweet_white).
weak_category_of(soft_cheese_and_cream, medium_red).
weak_category_of(soft_cheese_and_cream, rose_).
weak_category_of(soft_cheese_and_cream, light_white).
weak_category_of(soft_cheese_and_cream, sparkling).
weak_category_of(soft_cheese_and_cream, dessert).
weak_category_of(soft_cheese_and_cream, sweet_white).
weak_category_of(poultry, medium_red).
weak_category_of(poultry, rose_).
weak_category_of(poultry, light_white).
weak_category_of(poultry, sparkling).
weak_category_of(roasted, medium_red).
weak_category_of(roasted, light_red).
weak_category_of(roasted, rose_).
weak_category_of(fruit_and_berries, sparkling).
weak_category_of(fruit_and_berries, dessert).

% rules
wine_checking(W,IC,EC,IR,ER,ICOL,ECOL,T,TL,D) :- wine(W),
   region_belongs(IR,IC), exclude_wine_by_country(W,EC), exclude_wine_by_region(W,ER),
   wine_from(IR,W),
   wine_category(CAT), wine_color_of(CAT,ICOL),
   ICOL \== ECOL, category_of(W,CAT),
   taste_of(W,T), tannin_of(W,TL), dryness_of(W,D).

ideal_strong_category(M,C) :-
   ingredient_of(M,I),
   is_in_list(ING,I),
   strong_category_of(ING,C).

ideal_weak_category(M,C) :-
   ingredient_of(M,I),
   is_in_list(ING,I),
   weak_category_of(ING,C).

get_strongs(M,R) :- bagof(
           X,
           ideal_strong_category(M,X),
           R).

get_weaks(M,R) :- bagof(
           X,
           ideal_weak_category(M,X),
           R).

ideal_strong_categories(M,C) :-
   setof(CAT,
       (get_strongs(M,R1),
            get_weaks(M,R2),
            list_concat(R1,R2,R),
          wine_category(CAT),
          is_in_list(CAT,R1), % check if is a strong category
          is_in_list(CAT,R), % and is in concatenation list
          list_countall(R,CAT,O),
          ingredient_of(M,I),
          length(I,L), O >= L),
       C).

ideal_weak_categories(M,C) :-
```

```
   setof(CAT,
        (get_weaks(M,R),
           wine_category(CAT), is_in_list(CAT,R),
           list_countall(R,CAT,L),
           ingredient_of(M,I),
           length(I,L)),
        C).

is_ideal_strong_wine(W,M) :- wine_category(CAT), ideal_strong_categories(M,C),
   is_in_list(CAT,C), category_of(W,CAT).

is_ideal_weak_wine(W,M) :- wine_category(CAT), ideal_weak_categories(M,C),
   is_in_list(CAT,C), category_of(W,CAT).

region_belongs(R,C) :- country(C), region(R), region_of(C,R).

exclude_wine_by_country(W,EC) :- wine_from(R,W), region_belongs(R,C),
   EC \== C.
exclude_wine_by_region(W,ER) :- wine_from(R,W),
   ER \== R.

include_wine_by_grape(W,IG) :- IG=[] -> (grapes_of(W,_));
   grapes_of(W,GS), is_in_list(G,IG), is_in_list(G,GS).

exclude_wine_by_grape(W,EG) :- grapes_of(W,GS), list_intersection(GS,EG,NW),
   not(is_in_list(G,NW)), is_in_list(G,GS).

suggested_general_wines(W,M,IG,EG,C,EC,R,ER,COL,ECOL,T,TL,D) :-
   wine_checking(W,C,EC,R,ER,COL,ECOL,T,TL,D),
   include_wine_by_grape(W,IG),
   exclude_wine_by_grape(W,EG),
   meal(M).

suggested_strong_wines(STRONG,M,IG,EG,C,EC,R,ER,COL,ECOL,T,TL,D) :-
      suggested_general_wines(STRONG,M,IG,EG,C,EC,R,ER,COL,ECOL,T,TL,D),
   is_ideal_strong_wine(STRONG,M).

suggested_weak_wines(WEAK,M,IG,EG,C,EC,R,ER,COL,ECOL,T,TL,D) :-
      suggested_general_wines(WEAK,M,IG,EG,C,EC,R,ER,COL,ECOL,T,TL,D),
   is_ideal_weak_wine(WEAK,M).

% suggested_wines(STRONG,WEAK,roast_chicken_with_herbs,[],[],IC,EC,IR,ER,ICOL,ECOL,T,TL,D)
% NOTE: M, the meal is needed.
% IG, Include Grape and EG, Exclude Grape should be empty lists [], [] if
% there are no preferences about inclusion and exclusion of grapes.
%
% IC: Include Country, EC: Exclude Country
% IR: Include Region, ER: Exclude Region
% ICOL: Include Color, ECOL: Exclude Color
% T: Taste, TL: Tannin Level, D: Dryness. To negate one of these: bold/light, tannic/less tannic, dry/not dry
%

suggested_wines(STRONG,WEAK,M,IG,EG,IC,EC,IR,ER,ICOL,ECOL,T,TL,D) :-
   setof(STRONG, suggested_strong_wines(STRONG,M,IG,EG,IC,EC,IR,ER,ICOL,ECOL,T,TL,D), STRONG);
   setof(WEAK, suggested_weak_wines(WEAK,M,IG,EG,IC,EC,IR,ER,ICOL,ECOL,T,TL,D), WEAK).
```

Listing 7.1: Our Knowledge Base and Wine Pairing in Prolog