Agentive Architecture, Ontology, and Judgement

Intelligent Agents Final Report

Yulia Terzieva*
GSNS Utrecht University
y.i.terzieva-7868502

Kuil Schoneveld
GSNS Utrecht University
k.g.schoneveld-7129019

Ramón Rico Cuevas
GSNS Utrecht University
r.ricocuevas-1243012

Raoul Brigola
GSNS Utrecht University
r.j.brigola-7843550

Alexandros Constantinou
GSNS Utrecht University
a.constantinou1-2126974

Parsa Beigzadeh
GSNS Utrecht University
p.beigzadeh-1203754

ABSTRACT

This paper summarizes our project implementing a software agent to perform simple claim-based reasoning. The agent's internal mechanisms enable it to reason about scenarios in a proprietary ontological environment in combination with accessing Twitter's API. We overview the construction of the agent's internal mechanisms, its relationship to the built ontology and some specific features therein, and indicate directions of future work.

KEYWORDS

agent architecture, ontology design, reasoning

INTRODUCTION

This report demonstrates a team exercise in agent design and creation. More specifically, it articulates the inner workings of our agent whose purpose is to reason about straightforward claims to approximate their truthfulness. When designing software agents for personal use, it is critical that the primary users can understand its inner workings to some degree. To this end, we have striven to create an agent which can return simple explanations for its judgements. These come in the form of a weighing of the evidence for and against a certain claim using a combination of the information contained in the agent's private knowledge base and public information from Twitter's API.

Though the current version is a straightforward functional prototype, further development would make this prototype useful to a potential user interested in creating a personalized agent capable of automated querying that combines both a local store of accrued knowledge with another public source. Moreover, it could form the basis of a system which could approximate fake news detection, insofar as it contains a method for weighing supporting evidence as either correct or not.

In what follows, we i) overview the design of our agent's architecture, ii) sketch the main features of the ontology through which the agent reasons, iii) explain the scenarios and querying methods by which the agent approximates a truthfulness score, iv) discuss our implementation strategy and its v) related performance metric, and finally vi) gesture toward further directions into which the work could have expanded given alternate constraints on the project.

1 AGENT ARCHITECTURE

To begin, the chosen agent architecture corresponds to a model-based reflex agent (i.e., a simple reflex agent with an internal state). The overall agent architecture is depicted in Figure 1. In the following, each component will be explained.

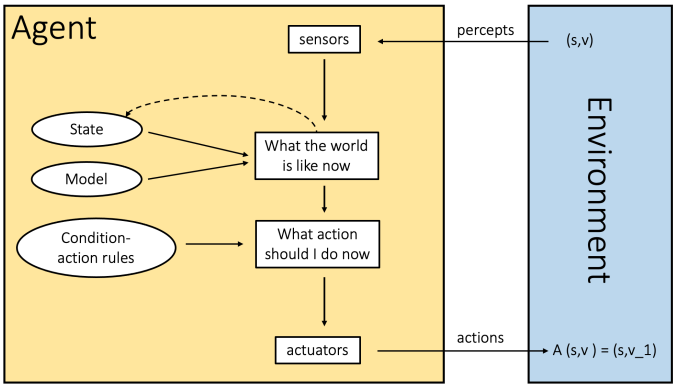

Figure 1: Agent Architecture

1.1 Environment

The set of environment states $\mathcal{S}$ is the set of tuples with the first component corresponding to a string in the English natural language and the second component corresponding to its veracity. That is,

$$\mathcal{S} = \{ (s, v) \}$$

where, s represents a statement in the English natural language and $v \in [-1, 1] \subseteq \mathbb{R}$ measures its degree of veracity. Note that a statement is completely veridical if its veracity takes value 1 and completely non-veridical if its veracity takes value -1. Moreover, for any statement s, $S_0 = (s, 0) \in \mathcal{S}$ corresponds to the initial state. Specifically, S_0 is the state corresponding to the situation in which statement s has not been processed and hence its current veracity is 0 (halfway between -1 and 1). In general, for a given statement s, S_v denotes the environment state $(s, v) \in \mathcal{S}$ where $v \in [-1, 1]$.

*All email addresses are @students.uu.nl

Proc. of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020), 2020

https://doi.org/doi

1.2 Actions

For any given environment state $S_v \in \mathcal{S}$, the agent can perform any action inside the set $\mathcal{A} = \{A_0, A_1, A_2, A_3\}$ where,

A_0 : no action,
A_1 : query ontology,
A_2 : query Twitter,
A_3 : return final answer.

Actions A_1 and A_2 have four possible outcomes: success (positive, negative, neutral) or failure. Firstly, success-positive indicates that information providing positive evidence in favour of the veracity of statement s was successfully retrieved from the source. Secondly, success-negative indicates that information providing negative evidence in favour of the non-veracity of statement s was successfully retrieved from the source. Thirdly, success-neutral indicates that balanced positive and negative evidence was successfully retrieved from the source. Last, failure indicates that no information was found in the source. Every action listed above makes the environment transition from one state to another as follows:

1. $\forall v \in [-1, 1],\ i \in \{0, 3\},\ S_v \xrightarrow{A_i} S_v$
   (i.e., actions A_0 and A_3 do not change the veracity of statement s).
2. $\forall v \in [-1, 1],\ i \in \{1, 2\},\ S_v \xrightarrow{A_i} S_{v' \neq v}$
   (i.e., actions A_1 and A_2 may change the veracity of statement s depending on their outcome: $v' = v$ when the action returns success neutral or failure, $v' > v$ when the action returns success-positive and $v' < v$ when the action returns success-negative).

`EXECUTE-ACTION` (1), implements the environment state transitions.

```
def EXECUTE-ACTION((s, v), action, w_0 = 0.5, w_1 = 0.5):
    # EXECUTE-ACTION applies a given action to a given
     environment state.
    # input: (s, v), action, weights w_0 and w_1. The
     weights represent the amount of influence on the
     veracity v of statement s each source of information
      has. Weight w_0 refers to the ontology and weight
     w_1 refers to Twitter. They are set to 0.5 by
     default.
    # output: new environment state (s, v_1)

    if action == A_1:
        if action.success == true:
            v_1 = v + w_0((number_of_positive results/
     total_number_of_results)-0.5)*2
    elif action == A_2:
        if action.success == true:
            v_1 = v + w_1((number_of_positive results/
     total_number_of_results)-0.5)*2
    else: # A_0, A_3, failed A_1 or failed A_2
        v_1 = v
    return(s,v_1)
```

Listing 1: `EXECUTE-ACTION` applies a given `action` to a given environment state `(s,v)`.

1.3 Internal state

The agent's internal state represents the agent's current conception of the environment state. The set of internal states is denoted by $\mathcal{I}$ and is defined as follows.

$$\mathcal{I} = \{\ (o,\ t)\ |\ o, t \in \{none, 0, 1\}\ \}$$

where o and t encode whether meaningful information (i.e., non-positive-negative-balanced) was returned upon querying the ontology and Twitter respectively. In fact, there only exist $|\mathcal{I}| = 9$ internal states. That is, $\mathcal{I} = \{I_0 \ldots, I_8\}$ where,

$I_0 = (none, none)$: Neither the ontology or Twitter have been queried before. Note that the agent's internal state will always initially be I_0.
$I_1 = (0, none)$: The ontology has been queried and no information or balanced positive-negative information has come back. Twitter has not been queried.
$I_2 = (none, 0)$: The ontology has not been queried. Twitter has been queried and no information or balanced positive-negative information has come back.
$I_3 = (1, none)$: The ontology has been queried and meaningful information has come back. Twitter has not been queried.
$I_4 = (none, 1)$: The ontology has not been queried. Twitter has been queried and meaningful information has come back.
$I_5 = (0, 0)$: The ontology has been queried and no information or balanced positive-negative information has come back. Twitter has been queried and no information or balanced positive-negative information has come back.
$I_6 = (0, 1)$ The ontology has been queried and no information or balanced positive-negative information has come back. Twitter has been queried and meaningful information has come back.
$I_7 = (1, 0)$ The ontology has been queried and meaningful information has come back. Twitter has been queried and no information or balanced positive-negative information has come back.
$I_8 = (1, 1)$ The ontology has been queried and meaningful information has come back. Twitter has been queried and meaningful information has come back.

The variable `state` in Listing 2 corresponds to the agent's current internal state.

1.4 Model

Updating the internal state of the agent requires a *model*. A model is simply knowledge about "how the world works". The `UPDATE-STATE` function in 2 is responsible for creating the new internal state (`state`) based on the old internal state (`state`), the most recent action (`action`), the current percept (`(s,v)`), and the agent's model of how the world works (`model`). The agent's model takes into account the previous percept denoted by (`(s,v*)`) and it comprises the following rules.

1. **if** (`action` = A_1 and `v` ! = `v*`) **then** `state` = (1, `state[1]`)
   (i.e., **if** the ontology has been queried (action A_1 was the last action) and the veracity has changed **then** meaningful information was successfully retrieved. That is, A_1's outcome was positive or negative successful. Hence, the internal state should be updated to have first component equal to 1. States like this are: I_3, I_7, I_8.)
2. **if** (`action` = A_1 and `v` = `v*`) **then** `state` = (0, `state[1]`).
   (i.e., **if** the ontology has been queried (action A_1 was the last

action) and the veracity has **not** changed **then** neutral information was successfully retrieved or no information was retrieved at all. Either way, the ontology has been queried and no information of use has come back. Hence, the internal state should be updated to have first component equal to 0. States like this are: I_1, I_5, I_6.)

3. **if** (`action` = A_2 and `v` ! = `v*`) **then** `state` = (`state[0]`, 1). (i.e., **if** Twitter has been queried (action A_2 was the last action) and the veracity has changed **then** meaningful information was successfully retrieved. That is, A_2's outcome was positive or negative successful. Hence, the internal state should be updated to have second component equal to 1. States like this are: I_4, I_6, I_8.)
4. **if** (`action` = A_2 and `v` = `v*`) **then** `state` = (`state[0]`, 0). (i.e., **if** Twitter has been queried (action A_1 was the last action) and the veracity has **not** changed **then** neutral information was successfully retrieved or no information was retrieved at all. Either way, Twitter has been queried and no information of use has come back. Hence, the internal state should be updated to have second component equal to 0. States like this are: I_2, I_5, I_7.)
5. **if** (`action` = `null` or A_0) **then** `state` = `state`. (i.e., **if** there is no last action or the last action was A_0 **then** the internal state will not change).

1.5 Condition-Action Rules

A condition-action rule has the following structure:

if (condition holds) **then** (perform an action $\in \mathcal{A}$).

Let I denote the current internal state of the agent. The condition-action rules used by the agent are:

1. **if** ($I = I_i$ for $i \in \{0, 2, 4\}$) **then** A_1 (i.e., **if** the ontology has not been queried **then** query the ontology).
2. **if** ($I = I_i$ for $i \in \{1, 3\}$) **then** A_2 (i.e., **if** Twitter has not been queried but the ontology has been queried **then** query Twitter).
3. **if** ($I = I_5$) **then** A_0 (i.e., **if** both the ontology and Twitter have been queried and no information or balanced positive-negative information has come back **then** do no action. Let the user know that no information is available to determine the veracity of the statement).
2. **if** ($I = I_i$ for $i \in \{6, 7, 8\}$) **then** A_3 (i.e., **if** the ontology and Twitter have been queried and at least one of them has successfully returned meaningful information (non-positive-negative balanced) **then** return final answer).

The variable `rules` in 2 corresponds to these condition-action rules. The function `RULE-MATCH` in Listing 2 takes as input the current internal state (`state`) and the set of condition-action rules (`rules`) and returns an action (`action`).

```
def AGENT-PROGRAM((s, v)):
    # input: (s, v) = (statement, veracity)
    # output: action (i.e., the next action to be taken)
    persistent:
        state  # the agent's current conception of the world state (i.e., the agent's internal state) initially I_0
        model  # a description of how the next state depends on current state and action
        rules  # a set of condition-action rules
        action # the most recent action, initially NONE


    state = UPDATE-STATE(state, action, (s, v), model)
    action = RULE-MATCH(state, rules)
    return action
```

Listing 2: Agent program.

Given a statement s, the function `EVALUATE-SCENARIO` in 3 shows the overall process that the agent follows to determine the level of truthfulness of such statement.

```
def EVALUATE-SCENARIO(s):
    # input: s = statement to be processed with initial veracity v = 0
    # output: (s, v*) = statement s and its found veracity v*.

    (s, v) = (s, 0)
    action = NONE
    w_0 = 0.5
    w_1 = 0.5
    while(action != A_3 or A_0):
        action = AGENT-PROGRAM((s, v))
        (s,v) = EXECUTE-ACTION((s, v), action, w_0, w_1)
    return (s, v*) = (s, v)
```

Listing 3: `EVALUATE-SCENARIO` determines the level of truthfulness of statement s.

2 ONTOLOGY DESIGN

Our ontology contains seven tightly interconnected top-level classes describing aspects of human bodies and health, sporting activities, and various foods supplying nutrients necessary for human living. To connect these classes and make this logical landscape suitable for reasoning, we have created multiple object- and data-properties restricted to particular aspects of this domain to facilitate a richer querying capacity. Figure 2 depicts the whole ontology.

2.1 Classes

To quickly describe the non-original features of our ontology shared by all such projects, we have included classes for Food, Recipes, Sports, and Health (as well as a link to Twitter as an external source). Alongside these four top-level classes, we have included classes labelled Person, Nutrition, and Body.

The Food class represents items that may be edible themselves, or are not themselves edible but which constitute the materials involved in the creation of edible Recipes. Notably, our division of the Food class includes decidedly inedible classes such as HerbsAndSpices, Oils, and Grains, but also includes categories such as Dairy, Fruit, MeatProducts, and Vegetables. In total, we have more than 50 classes of food items, each with a value of their most prominent macro-nutrient in alignment with the basic food groups (Dairy most prominently contains Fats, MeatProducts containing Protein etc).

In accordance with the above, the class of Recipes represent the items that can be eaten; sorted into American, Indian, Italian, and Mexican cuisines as well as a class of PreparationMethods. We intended to include cardinality restrictions to indicate the amounts of each material used, but the only restriction which did not create asynchrony immediately was asserting a redundant minimum of 1 ingredient to be used. As such, our property hasIngredient was only capable of expressing the value 'some' such that it could indicate whether a Recipe included a certain food, but could not express to what extent it did so. Nevertheless, though this does represent one aspect of lost information not capturable by our current ontology, this is a relatively minor loss in its expressiveness for the sake of our querying methods. Precise values notwithstanding, we have 17 Recipes in our ontology, some with further Recipes nested within, such that a recipe for a pasta dish may contain a recipe for a sauce and a recipe for garlic bread etc.

Further, the Nutrition class represents the link between the Recipes and Food and the Health class, such that we can make causal statements about the health status of a person based on their eating habits. For example, one can distinguish between Recipes rich in protein as opposed to those that contain higher levels of sugar, fats, or carbohydrates.

The Health class defines a set of possible ailments that afflict instances in the Person class, as well as possible remedies to such afflictions. These are grouped by the areas of the body they afflict, or by their symptomatic onset. This includes a selection of chronic conditions, acute injuries, or diseases transmissible via team sports. For example, eating recipes containing too much sugar causes the Health condition high blood sugar, which itself causes instances of diabetes, while a case of COVID infection can cause the Person carrying it to also exhibit asthma. Moreover, particular deficiencies can be remedied by eating Recipes dense in the missing nutrient.

To connect notions of physical exercise, our class of Persons lists individuals who can be attributed Body parts, play Sports, as well as exhibit the aforementioned health conditions. The Body class splits into upper and lower to categorize the parts constituting the Persons, and each of these parts is linked to one another to form a network of dependence. Given the natural inseparability of body parts, the connectedTo relation allows for inferences to be made about injuries and their potential effects on neighbouring body parts. The intention of organizing the class using this relation is to say that, if a forearm is strained, a connected wrist could develop a chronic pain as well.

Finally, the Sports class is divided into team and individual activities such as Waterpolo, Quidditch, Ice/Field-hockey, Basketball, and Football in the case of the former and Golfing, Swimming, Rock-climbing, and Kickboxing for the latter, each requiring use of different body parts and presenting different injury risks. For example, Rock-climbing is most associated with injuries such as torn ligaments and strain to the forearms, wrists, and hands, whereas kickboxing presents risk of fractures while depending most notably on the hips and lower legs. Furthermore, different sports could be grouped by similarity depending on their shared degree of bodily dependence, where two sports involving the same regions of the body would be more highly related to one another than two sports using different regions (waterpolo and swimming vs waterpolo and rock climbing for example).

The aim of designing a closely-knit but not redundantly overlapping set of classes for our ontology was to precisify the relationships between diet, exercise, and health, thus allowing our relations to be more specifically restricted and homogeneous in application. This challenged us to more fully understand the relationships we were asserting since the room for logical error was far less than if we had chosen topics that only loosely related to one another.

2.2 Object Properties

To further enrich our understanding of the capabilities of our classes, we can articulate some of the properties we have created for drawing inferences related to various aspects of the ontology. As required, we have included four different properties with one being transitive (hasIngredient, between recipes and foods), symmetric (connectedTo, between various body parts), functional (denseIn, between a recipe and its *primary* macro-nutrient), and inverse functional (cooksBest, between a recipe and its best chef/person).

These examples are our most commonly used properties that fit these categories, but this is certainly not exhaustive. Additionally, we have a property that is both transitive and symmetric (playsWith, demarcating participants in team sports), as well as properties that have Description Logic (DL) statements as their domain or range (for example, oftenEats has domain Person and range Recipes or Fruits or Vegetables or Dairy). In total, we have 28 properties and subproperties which specify a selection of interrelationships between our concepts. A few of our relatively interesting properties (beyond the first four mentioned) are described here in a non-exhaustive list:

- afflictedWithDisease - subproperty of afflictedWith; asymmetric, relates Persons to their ailments in Health > Diseases.
- causesDisease - subproperty of causesHealthRisk; asymmetric, relates Health or Nutrition to show how they induce Diseases.
- hasBodyPart - relates Persons to their BodyParts, a seemingly trivial step that only becomes apparent when assembling an entire instance of a Person.
- prominentlyRequires - relates a Sport to the BodyParts most likely to be used or injured while participating.
- propagatesInjury - subproperty of connectedTo; symmetric, relates Fractures, TornLigaments, and Strain to BodyParts to allow for inference of further ailments as a result of untreated injury.
- isCuredBy - relates Conditions or Diseases to the Nutrients involved in their remediation. This allows for inference of Recipes and therefore Foods that help mitigate the effects of scurvy our primary example of a diet-remediable disease.

Further, we have another 16 relations linking - for example - injuries to the sport that causes them and the body part that can be injured, or foods to their most common methods of preparation (to be made into Recipes via the PreparationMethods class mentioned earlier).

2.3 Data Properties

For additional descriptive power, we have included a few notable data properties to allow for more specific and straightforward querying, as well as to allow inferences requiring contextual information

that must be explicitly inserted into the ontology. Most non-trivially, our outdoorSport data property specifies which sports are typically played outside. This is notable because it would allow inferences to be made between the types of sports played and the participants' affinity for acquiring the recommended levels of Vitamin D, which is defined in our Health class. Our definition of isVegetarian uses the aforementioned type of DL sentence in its domain to specify those Recipes which do not have any ingredient from the MeatProducts class. As well, hasAge is useful to sort our Person instances by their relative age differences, and isHappy denotes Persons who, despite the various diseases they can acquire, maintain an optimistic outlook on life.

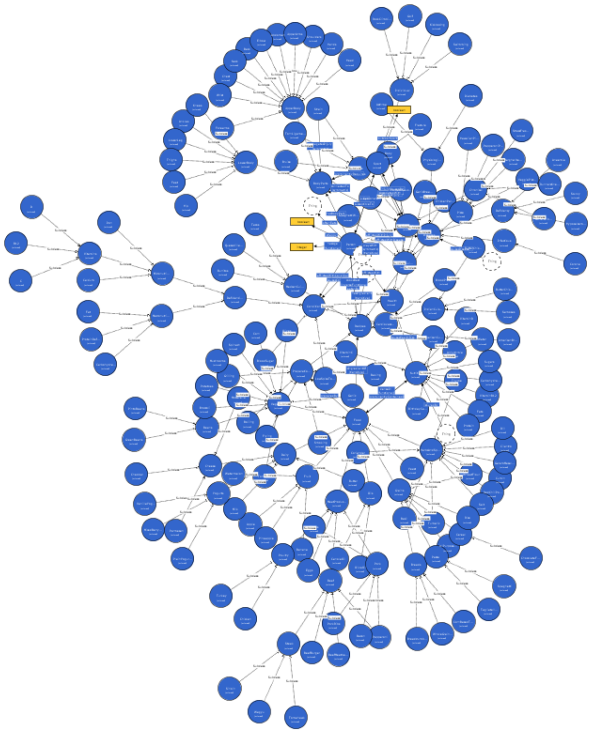

Figure 2: Ontology

3 SCENARIOS AND QUERYING

Our scenarios are intended to uniquely cover varying regions of the possible logical space of the constructed ontology and external Twitter access. As such, the first scenario is intended to be fully answerable by the agent's knowledge base, whereas the second and third gradually rely more heavily on external information. Though in reality the agent is informed by Twitter regardless of the scenario, this variation demonstrates the difference between claims which can be reasoned about using more or less of the agent's inbuilt knowledge as opposed to information it must retrieve from elsewhere. As such, it enables a difference in weight to be assigned depending on the origin of the evidence received, where Twitter's information is generally valued less than that contained in the inbuilt knowledge base. This valuation mechanism is expanded upon in Section 4.

The first scenario claims that "Healthy people are happy", and this is intended to reveal the variation of interpretations our ontology can support regarding the health of a person. As mentioned in the previous Section, our agent acts first by accessing the ontology via a series of SPARQL queries, followed by requesting relevant information from Twitter. This Section of the report discusses only the former of these two processes. Therefore, in performing a query, the agent thereby implicitly defines what constitutes a healthy person. For example, a healthy person is one that is young and does not suffer from Diseases, who plays at least one instance of a Sport and eats Foods packed with proteins in combination with vegetables. This definition of health is imposed upon the agent such that it facilitates the structuring of queries that can be used to seek evidence. As a result, the main claim to be analyzed, whether "Healthy people are happy", is thus split into a conjunction of the following seven queries.

(1) Querying "Happy people don't suffer from any disease" returns all the instances in the class "Person" that have a data property "isHappy" with value "true" (hereafter referred to as "happy people") and do not have an object property "afflictedWithDisease" with any endpoint of an instance of a disease.
(2) Querying "Sad people have scurvy" returns all the instances in the class "Person" that have that have a data property "isHappy" with value "false" and have an object property "afflictedWithDisease" pointing to an instance of the class "Scurvy", which is subclass of "Disease" under "Health". This gathers further evidence that not only are we finding the true positive cases for supporting our claim, but also populating the true negative cases.
(3) Querying "People that are training are happy people" returns the happy people that have an object property "plays" pointing to an instance in the class "Sports". This relies on the intuition that frequent exercise makes for a healthier life.
(4) Querying "People that eat recipes that have vegetables are happy" returns all the happy people that have an "oftenEats" object property that points to a instance, which itself has an object property "hasIngredients" pointing to an instance which is in class "Vegetables". Of course, a balanced diet is crucial for one's health, in contrast to our Recipe American-Breakfast for example, which contains far more sugar.
(5) Querying "People that eat food packed with protein are happy" returns the happy people that "oftenEats" object property that points to an instance, which itself has an object property "denseIn" pointing to an instance which is in class "Protein". This completes the notion of a balanced diet begun by the vegetables query (4).
(6) Querying "People that have Corona are not happy" returns all the instances of class "Person" that have that have a dataproperty "isHappy" with value "false" and have an object property "afflictedWithDisease" pointing to an instance of the class "Corona", which is subclass of "Disease" under "Health". Similar to finding individuals that have scurvy, this

identifies further negative evidence in favour of the overall claim.

(7) Querying "Young people are happy" returns all happy people that have a dataproperty "hasAge" smaller then 27. This is to use the variation in age already present in our ontology, under the assumption that people often look back on their lives and wish they had done more or lived healthier in their youth, and that typically our healthiest days are when we still have the energy to exercise etc.

Clearly, many trivial class-membership inferences are being made in these queries, and more non-trivial inferences via our defined relations help to bridge between the various parts of our ontology. Each query returns a number of instances that fulfil the described constraints. Those instances provide the evidence to be later used in calculating the veracity of a scenario, which gives an approximation of how truthful a statement may be.

In particular, our queries differ from a straightforward database lookup because there are a series of potentially branching inferences being made using our relations. Our aforementioned example of vegetables contributing to happiness (4) clearly demonstrates the use of our most common transitive property hasIngredient. As such, any recipes and sub-recipes which contain vegetables as ingredients are identified as contributing to the health of those who eat them often.

Currently, one of our more complicated queries - and thus, furthest from being a simple database lookup - is that which identifies those persons who are afflictedBy one of our defined blood sugar conditions who also oftenEat foods that are denseIn sugars. This query is conjoined with others to find the exercise habits of those who eat sugar to reason about the overall scenario. More importantly however, the act of identifying those with high blood sugar uses both inbuilt rdfs:subClassOf relations, as well as four separate relations defined within our ontology to demonstrate some of the inferences that are possible for our agent.

One aspect in the variation of our querying that is worthy of note is that the scenario "Healthy people are happy" has only positive/confirmational queries, though this is not necessarily clearly justified. For our purposes positive queries are ones that, when they return instances, contribute in favour of the statement in the scenario. This is by design, such that we could test the approximate accuracy of our performance metric on a simple case in which we knew the results of the query without having to compute it. As we have acknowledged, a tighter fit to reality could also include instances of unhealthy people who are happy, as well as healthy people who are unhappy, and those who are both unhappy and unhealthy, to make for a more clearly demarcated causal inference between health and happiness. This being said, the purpose of this scenario was more straightforward, though we mention the general nature of this issue in Section 6.1.

With more specific reference to the constraints of this project however, it should also be said that our ontology only has the capability to deal with a limited number of person instances, since the degree of ontology-population required to establish a causal relationship along numerous instantiated inferences is beyond the scope of this course. This type of causal strengthening is one aspect we would have improved upon if the course constraints allowed.

Certainly however, it is not the case that all the queries can only return positive support/confirmation of the scenario's claim. For example, the scenario "Individual sports require lower body" the agent can query two things:

"Individual sports that require lower body", whose results would be counted as contributing in favour of the statement in the scenario and "Individual sports that require upper body", the results of which would be held against the statement.

If we run this example we get n sports that are individual and require lower body and m individual that require upper body. Thus using the formula :

$$(\#positive\_instances/(\#total\_instances) - 0.5) * 2$$

we get a number between -1 and 1 representing the veracity of the statement in the scenario.

Another example where the agent queries for both confirming and disconfirming evidence is for the scenario "Sugar is good for people". This example offers a case of ambiguous definition in terms of "Good". 'Good for people' could mean inducing happiness in people, so "People that eat sugar are happy" is a reasonable query to use to investigate the scenario. Sugar also increases a person's glucose levels resulting in more energy so a query like "People that eat sugar play sports" may also be reasonable. Other queries that connect the eating habits and the health of a person may include "People that eat sugar have high blood pressure" and also "Sugar causes diabetes", which clearly undermine the claim that sugar is good for a person. Further discussion of the challenge of operationalizing our scenarios into well-defined queries is provided in Section 6.2.

4 IMPLEMENTATION

Henceforth, the term "architecture" refers to the software architecture of the program at class and module level as opposed to the agent architecture as described in Section 1.

4.1 Architecture Decisions

During development, we focused on designing a flexible and easily extendable architecture, following standard software design patterns. For example, additional data sources - other than the ontology and Twitter - can be easily added to our agent program by creating a realization of the interface *IDataSourceHelper* and a corresponding realization of the interface *IResult*. Outsourcing the data source connections to helper classes has the additional advantage that the class *Agent* does not have to concern possible simplifications in the way data sources are queried. In our program, one such simplification is the fact that ontology-queries are hard-coded for the individual scenarios. In a more advanced version, queries might be derived from the scenarios through NLP algorithms. However, if this were to be changed, it would not affect the agent class in any way.

Even more integral parts of the agent like the internal state and the available types of actions can be modified flexibly. To ensure this, we implemented actions following the strategy pattern. More specifically, an instance of the class *Action* is always of a certain action type. The actual procedure that is called when the agent executes an action is stored in the value of the action type. This

allows the agent to execute all actions in the same way, regardless of whether the action implies querying the ontology,querying Twitter, or merely updating the internal state.

As the core module of the program, the implementation of the class *Agent* closely follows the description of the agent architecture as described in Section 1. It contains only one method that is meant to be called from outside the agent. This method, called *evaluate_scenario* (see Listing 3 in section 1), is the internal starting point for the agent program. It resets all instance variables before it returns the final evaluation to the user. This makes an instance of the class *Agent* reusable, meaning a user can give as many scenarios to one agent instance as desired.

4.2 Ontology

The ontology has been constructed using Protege 5.5 and is accessed via owlready2. This requires a local copy of the ontology file, which is stored in the folder *res*. To retrieve information, we use SPARQL-queries, which return a list of instances from the ontology. The class *OntologyHelper* contains hard coded queries that correspond to specific scenarios. Queries can provide either positive or negative evidence for a scenario (confirming or disconfirming, respectively). The ontology helper also contains a method to calculate a confidence value, using all found instances. The confidence value lies in the familiar range of [-1, 1] and indicates how much positive or negative evidence the ontology returned.

4.3 Twitter Sourcing

The information in this section takes advantage of a sample of the public opinion from Twitter, and analyzes its sentiment to determine its degree of alignment with the claims made in our scenarios. To begin, a connection to Twitter is established by utilizing the Twitter API Version 2. Following a successful connection, the relevant data is gathered via our queries. To create a comparable format for the information contained in the tweets, they must be edited such that they can be conjoined with the query results and thereby form a reasonable argument. To accomplish this, we use regular expressions to separate the words and remove links, special characters and simple regex statements. The meaning of the resulting words can then be determined by utilizing text recognition. To develop a notion of the sentiment contained in the text of the tweet, we make use of the Lexicon-based sentiment analysis package provided by Textblob. This package contains a set of predetermined rules structured as a word and weight dictionary, with scores that describe the sentimental polarity of a statement.

4.4 System Requirements and External Libraries

In order to run our program it is necessary to have Python installed, preferably Python 3. Our implementation makes use of the following external libraries:

- **Owlready2** is used as an API to retrieve information from the ontology. It is required by the class *OntologyHelper*.
- **Tweepy** is used to access the twitter API and retrieve information from it. It is required by the class *TwitterHelper*.
- **TextBlob** provides a simple API for common natural language processing tasks and is used to identify a tweet's sentiment. It is required by the class *TwitterHelper*.
- **NLTK** is used to prepare data, queried from Twitter. It is required by the class *TwitterHelper*.
- **ABC** is used to realize the concept of abstract classes which is not built-in in python. It is required by the classes *IDataSourceHelper* and *IResult*.

Furthermore, the implementation makes use of standard operating system functions and standard python logging.

5 PERFORMANCE MEASURE

As specified in 1, whenever our agent is introduced in an environment, it performs a series of actions (in $\mathcal{A}$) that make it transition through a sequence of states (in $\mathcal{S}$). A performance measure p, evaluates the desirability of such a sequence (pp. 37 [1]). That is, p is simply a mapping from the set of sequences of environment states belonging to $\mathcal{S}$ (i.e., $\{(s,v) \in \mathcal{S}\}_n$) to the interval $[0,1] \subseteq \mathbb{R}$. We then propose the following performance measure:

$$p: \quad \{(s,v) \in \mathcal{S}\}_n \longmapsto [0,1]$$

$$\{(s,v_0),\ldots,(s,v_k)\} \mapsto \begin{cases} 1 \text{ if } sign(v_k) = sign(b), \\ 0 \text{ if } sign(v_k) \neq sign(b), \end{cases}$$

where $b \in \{1,-1\}$ indicates if statement s is true (1) or false (-1).

Any given set of supervised scenarios $D = \{(s_1,b_1),\ldots,(s_n,b_n)\}$ where s_i is a scenario and $b_i \in \{1,-1\}$ indicates its truthfulness (true if 1, false if -1), can be fed into the agent for evaluation. The agent would process one by one the elements of the set of initial environment states $\{(s_1,0),\ldots,(s_n,0)\}$. The output yielded by the agent corresponds to the set of final environment states $\{(s_1,v_1),\ldots,(s_n,v_n)\}$ where v_i is the final veracity of scenario s_i computed by the agent as described in 1.

The overall performance P of the agent in the dataset D is given by

$$P = \frac{1}{n}\sum_{i=1}^{n} p(\{(s_i,v_i\}).$$

More precisely, P is simply the percentage of scenarios in D correctly classified by the agent's trivial associated classifier. That is, the classifier that assigns any scenario the same sign of the agent's determined veracity (i.e., the classifier that assigns the class 1 to every scenario for which the agent outputs a positive veracity and -1 to every scenario for which the agent outputs a negative veracity). Alternatively stated, P is a measure of the accuracy of the agent's trivial associated classifier.

As with any measure of performance, it is only useful with a relatively large set of labelled solutions for the purposes of comparison. Again, if the constraints of the project were conducive of such a comparison, we would have included far more examples to compute an average of the results.

6 DISCUSSION OF FURTHER WORK

Even after many hours of population, our ontology remains somewhat sparse compared to how densely connected it could be. Of course, there is no real way to approximate an upper bound of its interconnections since it is continuously growing in its class

instances and relations, and that these features are describing a fiction. As such, the ontology merely represents a continual work-in-progress as it slowly grows into a richer representation of the causal structures of reality.

Typically, our examples are meant to show the capability intrinsic to the design of our ontology rather than a uniformly distributed set of object- and data properties that can all be queried with a high degree of precision. If the constraints of the project were such that we could spend far more time populating the ontology, it would make for a richer queriable environment that could more clearly be compared to the overwhelming amount of Twitter information.

An additional feature that we had hoped to include - but which clearly lays outside the scope of the current project - was to create a decaying transitive property. More specifically, we had hoped to create a diminishing sense of propagation using our connectedTo relation insofar as it could pass a lesser and lesser risk of injury along to neighbouring body parts, rather than being fully transitive. Similarly and perhaps even more relevant to the notion of decaying probabilistic continuance, we would have liked to denote a relation such as causesCondition with a probability of success, such that we could express that a specific percentage of COVID cases caused asthma, rather than just 'some'.

6.1 A Note on Causation and Correlation

One notable setback in representational expressiveness is in Protege and OWL's limited means of distinguishing between correlational and causal statements. Moreover, the set of inferences that can possibly be made via SPARQL queries merely count the instances fitting some set of restrictions, not that they precisely identify any causal relationships aside from those implied by the semantics of our relations. That being said, there are certainly more advanced statistical methods for establishing confidence in causal claims from correlational data, but these methods extend beyond the scope of this project's constraints.

6.2 The Future of Operationalization in Agentive Judgement

It is important to mention that there is a difference between a personal agent's individualized notion of the health of its user as compared to a different person, as well as compared to various norms and standards. Insofar as an agent has its understanding of health shaped by the input of its user, this personalizes certain aspects of the concept of healthiness. In contrast, there are still some objective measures of health independently definable from the personal. As such, though it is important for the agent to conform its idea of health to that of its primary user, this introduces a risk of possibly missing a ground truth. The heavier the user's hand is in shaping the agent, the more apt it will be to lose its autonomy. So, we must acknowledge the plurality of possible operationalizations of any given concept that we impose within our querying.

More generally, this demonstrates the fact that a higher degree of autonomy in the agent would come as a result of it performing its own operationalization of concepts, such that it is not simply receiving our constructed notion of health and querying that. Instead, it would involved a far more sophisticated analysis of the target scenario such that the agent itself decides what is relevant to query, and this instead shapes its own original notion of the concept at hand.

Additionally, these varying personalized notions of health could be compared, thus opening the door to potential ethics violations. In the case of real, private data involving a person's health characteristics and scheduled events, there could always be potential risks of allowing this sensitive information to be inferred or otherwise accessed through the querying process. In terms of discussing the relevance of trust in agents in this domain, we could allow for a data property representing a measure of sensitivity to be continually updated that is associated with different pieces of information. Of course, the full range of possible ethical violations is far more than can be covered here, so it is prescient to explicitly mention that the aforementioned conceptual issues regarding representative accuracy and causal inference would take precedence before the agent could be considered a sufficiently tight mapping to reality to present ethical issues in consumer usage.

6.3 Structure of the Working Process

Typically our group would meet once a week in person during any given week, plus online meetings for working on specific tasks. Beginning in week 5, we began to meet more frequently to account for the increased demand in our self-imposed deadlines. We approximate that we have spent roughly 25 hours meeting in person and online for general group discussions, but the task-specific meetings were far longer (a few 6+ hour meetings) and did not involve all group members simultaneously.

Generally, we decided that based on our own experiences and strengths, we would achieve our main tasks by relying most heavily on those with the most domain-specific experience. As such, the development of the agent architecture as well as the development of the evaluation metrics were completed by Ramon. The implementation of the agent and evaluation metrics were done by Raoul. The ontology and querying methods were developed and implemented by Kuil and Yulia. Our project's interface with Twitter was added by Parsa and Alex.

REFERENCES

[1] Stuart Russell and Peter Norvig. 2010. *Artificial Intelligence: A Modern Approach* (3 ed.). Prentice Hall.