Warsaw University of Technology

FACULTY OF
ELECTRONICS AND INFORMATION TECHNOLOGY

Bachelor's diploma thesis

in the field of study Computer Science
and specialisation Information and Decision Systems

Implementation of syllogistic logic using RDF (Resource Description Framework) based upon a non-sql database

Daniel Jaździkowski
student record book number 199302

thesis supervisor
Prof. dr hab. inż. Jan Mulawka

Warsaw 2017

Temat pracy:
Implementacja logiki sylogistycznej przy użyciu systemu RDF (Resource Description Framework) opartego o bazę danych typu non-sql

Streszczenie
Przedmiotem pracy jest implementacja logiki syllogistycznej przy użyciu systemu RDF (Resource Description Framework) opartego o bazę danych typu non-sql. W toku realizacji pracy powstała aplikacja webowa implementująca funkcjonalność prostej galerii zdjęć. Realizuje ona wszystkie podstawowe funkcjonalności typowej galerii internetowej, oraz oferuje szereg dodatkowych funkcjonalności wyszukiwania, nie spotykanych w innych aplikacjach tego typu. Silnik syllogistyczny implementujący właściwą logikę jest modułem w pełni niezależnym, mogącym być bez przeszkód wprowadzonym do innego oprogramowania.

Słowa kluczowe: Logika sylogistyczna, Sylogizm, Logika Arystotelesa, Aplikacja sieciowa, Zarządzanie tagami, Lekka, Wyszukiwanie rozmyte

Synopsis
The subject of this thesis is an implementation of the syllogistic logic using RDF (Resource Definition Framework) based upon a non-sql database. In the course of the thesis a web application has been developed, implementing a simple online image gallery. The application contains all of the basic functionality of a typical online gallery, while providing additional search functionality, not encountered in other applications of this kind. The syllogistic engine, which implements the logic itself, is a completely independent sub-system, and can be easily deployed as part of other applications.

Keywords: Syllogistic logic, Syllogism, Aristotelean logic, Web application, Tag management system, Lightweight, Fuzzy search

Table of contents

1 Introduction

1.1 The history of logic

Logic has been the staple of knowledge and science for millennia. From the very first musings of philosophers o ancient times, to the modern sophisticated predictive algorithms. Since it had been conceived, it has been expanded and changed, developing into a multitude of systems for many different, specialized applications.

Logic, as a study of reasoning, had arisen semi-independently in many parts of the world, the very earliest known being the writings of ancient Chinese and Hindu philosophers. However, the western understanding of logic had been mostly based on, and influenced by, the writings of Ancient Greeks.

1.1.1 The beginnings of logic and Ancient Greek schools

The earliest years in the development of logic, like many other things, are shrouded in myth. Much of the information we possess is unclear, only available through much later writings. The scholars of the middle ages believed that the creator of logic was Parmenides. He was one of the first to use argumentation in his writings, however, without recognizing or studying the underlying principles.[1] His pupil, Zeno of Elea, is the author of many paradoxes, using exclusively reducto ad absurdum to refute supposed earlier notions of the world.

It is believed that among the early influences of logic, the two most prominent were rhetoric and geometry, both of which use logical argumentation to respectively refute and prove a thesis. Indeed, many rhetoricians and mathematicians made important contributions to the earliest study of language and its structure. These included: the differentiation between types of sentences. the distinction between verbs and names (the collective term for nouns and adjectives), and many others. These studies were crucial to the later development of the theory of logic.

The first to propose a complete system of argument and reasoning was Aristotle. His most famous works, known collectively as the Organon, contain the study of terms and their relations with one another, among many other topics. These studies lead to the formulation of the syllogism, which is the subject of interest for this work. Aristotelean logic became the basis of logical theory for many centuries.

Aristotle focused his studies on the categorical syllogism, that is, syllogism where the terms were exclusively describing collections of entities. Later writers explored different variations on the classic syllogism, as well as expanding on the theory of Aristotle.

A rival school to the Aristoteleans were the Megarans, and later the Stoics. Megarans mostly concerned themselves with examining paradoxes and logical puzzles. Unfortunately, not much remains of their texts. The most important contributions of the Megarans to the field of logic theory were the study of modality, and conditionals in the form of *if p then q*. Some of the most famous Megarans were Diodorus Cronus and his pupil Philo, who studied and debated the meaning of truth and its relation to time. Diodorus, who authored the so called Master argument, posited that everything, including truth, is static and unchanging. Therefore, no statement can ever change its logical value from true to false or vice versa. In contrast, Philo believed that a statement is only true at a moment in time, and that its value can change if the truthfulness of its components changes.

The Stoics continued and expanded Megarans' study of the conditionals by adding conjunctions, and disjunctions. In contrast to Aristotle, whose work regarded terms, they developed propositional logic.

From that point on, up until the middle ages, very little new logical theory is developed. Contemporary writings instead concentrate on gathering, translating, and commenting on the work of earlier logicians.

1.1.2 Arabic logic

After the Stoics, interest in logic in Europe declined rapidly. However, at the same time it rose among the Arabic scholars, who incorporated logic into their studies of theology, medicine and astronomy. Many early works of Greek logicians survived the fall of the Roman Empire thanks to their translations into Arabic. Later Arabic logicians contributed heavily to the revival of the active study of logic, rather than simple reproduction and commentary, as had been popular for many years.

The most influential of Arabic logicians was Ibn Sina, also known as Avicenna. Basing his studies on the Stoic interpretation of logic, he produced highly popular commentaries to Aristotle, expanding on the theoretical syllogism developed by the Stoics. His work also implements sophisticated forms of temporal logic and modal logic, presented together with the syllogisms as a unified whole.[2]

Avicenna's successors moved away from the syllogism in its various forms, and instead concentrated on developing an early version of inductive reasoning. Their writings focused on probability versus certainty and impossibility.

During the XIIIth century, the schools of logic in Arabia declined due to the rise of religious fundamentalism. However, by this time the interest in logic had already been rekindled in Europe.

1.1.3 European middle ages

The revival of logic studies in medieval Europe is intimately linked to the translation of Ancient Greek and Arabic works into Latin. This trend began in the XIIth century with Boethius' partial translation of the Organon, and experienced a boom in the next centuries, with multiple commentaries as well as original writings. In particular, Sophistic Refutations spawned a whole new genre of logical treatises concerning the study of fallacies.

The three biggest contributions to logic made during the medieval period were the study of supposition, the study of syncategoremata, and the theory of consequence.[3]

Supposition concerns the meaning and breadth of terms in logical propositions. In general, medieval scholars divided them into three categories:

1. Personal suppositions were terms denoting a singular entity
2. Simple suppositions were collections of entities or other universals
3. Material suppositions were the words themselves, as seen form the standpoint of the theory of language

A side development of the theory of supposition was the theory of connotation, which postulated that a single term can have different meanings depending on context. For example a word could act as both an adjective and a noun in different circumstances. This stands in stark contrast to the works of early Greek language theorists like Prodicus, who

postulated that a word can only ever have a single meaning[4]

Syncategoremata is a general term referring to all of the various words in a logical statement that do not have a meaning on their own, but instead modify the scope or meaning of other words. These are for example: while, if, then, sometimes, and many others.

Finally, the theory of consequence concerns hypothetical statements in the form of if … then, making them similar to the hypothetical syllogism. William of Ockham developed a full theory of consequence in his work Summa Logicae, where he distinguishes between material and logical consequence, that is, a statement “if a is true then b is also true” and “from the fact that a is true logically follows that b is also true”.

William of Ockham is also attributed with describing a principle called Ockham's razor, which states that from among all hypothesis, the ones with least assumptions should be selected as the most probable.

1.1.4 Renaissance and after

The transition from middle ages to renaissance was a turbulent one, filled with all kinds of social upheaval. Contemporary writings on logic reflected this, varying wildly in both general theory and particulars.

Renaissance scholars viewed middle ages and anything to do with them with disdain, preferring instead to revive the knowledge and art of the ancients as inherently superior. Paradoxically, this approach resulted in numerous critiques and refutations of Aristotle and the syllogism in particular. Many renaissance writers saw it as tainted by the multitude of medieval commentary, preferring instead to pursue different avenues.

Petrus Ramus wrote very influential works *Dialectique* and *Dialecticae libri*, in which he refutes Aristotle's notion of the syllogism, instead proposing a simplified version. He also, as one of the first, extensively studied the singular syllogism.

Still other writers postulated a return to the classical syllogism of Aristotle and a chosen few ancient commentators as the purest form of logic.

Of particular importance for later studies is *Ars magna, generalis et ultima* penned by Ramon Lull, in which he presents a study of concepts as deriving from the symbolic meaning of words and formulates symbolic logic. This and later works greatly influenced the notion of using specialized notation in logic.

Later studies postulated that a term carries with itself an 'understanding' of certain properties that are not explicitly stated. For example, the term 'triangle' implicitly infers being a poligon and having three corners.

This period saw the rise of induction as a rival method of solving logical problems, with important work done by Francis Bacon. Bacon argued that to infer knowledge about a subject, one must use empirical observation to list all cases when a certain proposition is true, all those when it is false, and all those when it might be either true or false. Only based on that one can infer truth about the universe.

1.1.5 Modern logic

What is commonly understood as modern logic was being developed from XIXth century onwards. This period saw an explosive popularity of notation logic and logical calculus. From the humble beginnings of symbolism, and extensive, but unpublished at the time writings of

1 Introduction

Leibnitz, modern mathematical tools for use with logic truly began with two people: George Boole and Augustus de Morgan.

Boole theories on the use of algebra to study and describe logic were one of the first extensive systems of notation for this field of science. His work was immediately picked up by numerous others, who improved upon it considerably.

While Boole mostly focused on assigning a notation to syllogisms, later writers, such as Gottlob Frege, strove to prove that any logic can be expressed in arithmetic, and indeed, the two were equivalent. His most important contribution was the abandonment of the traditional form of statement in favor of expressing logic as a series of functions that, given certain arguments (subjects), will produce output (predicates). Frege also distinguished between singular and general propositions. Where before it had been a standard practice to regard them as fundamentally equal, which resulted in weird implicit constructs in the form of 'Every Aristotle is a Greek' that were needed to solve some syllogisms.

An important contribution to the concept of terms and statement validity was made by Alfred Tarski. Tarski redefined the concept of truth, by posing that statements are only true when they express something that is empirically true. For example, “snow is white” is only true if we can attest by observation that snow is indeed white. This concept combines the methods of induction from empirical observation with deduction based on known facts. In doing so, it separates the object language used to describe entities from the meta language, used to make a statement about truthfulness of the object language statement.[5]

Among the myriad of great minds working on logic in the early XXth century, of particular note are Alan Turing and Alonzo Church. Turing worked on algorithmic proofs, and whose writings heavily influenced computing in general and the development of computers in particular. Church's work became the basis of λ-calculus.

In the few short decades of the XXth century we have seen a veritable boom in the study of logic, with whole new branches being developed seemingly overnight. With unprecedented access to information that current technology provides, it is impossible to fathom what breakthroughs in our understanding of logic will come next.

1.2 The subject and scope of the thesis

The subject of this thesis is an implementation of syllogistic logic with the use of RDF (Resource Description Framework) based on a non-sql database. Let us explain each part of this fairly enigmatic title in detail.

Syllogistic logic, as defined by Aristotle in his Organon, concerns relationships between groups of entities. It can be used to categorize said entities and find relationships that were perhaps not apparent.

There exists a myriad of systems used nowadays to describe and categorize resources in an information system. Of these the newest and gaining immense popularity, are the tags. A tag is a keyword describing some aspect of a resource. For example, an image could have the tags 'in-color' or 'black-and-white', while an employee could have the tags 'in training', or 'full-time'.

Tags are supposed to be used by the users of a system to easily describe a resource they themselves had added to it. As such, there is no real standardization of tags between various systems. Some administrators try to combat this fact by defining a database of tags for the users to choose from. Others give the users an unrestricted right to create new tags, and then rely on the dictionary functionality present in the system itself to group synonyms and related tags together.

The most sophisticated of these systems combine a dictionary with a database of previously used search terms. However, during my study of the subject, I have not found a system of grouping tags based on the classical syllogism.

As described later in this work, there is a number of issues arising from the use of natural language terms. To help address these, I have decided on the Resource Description Framework as the format of information. RDF is a format arising from the need to combine and relate information that might exist in various different systems, and so is not standardized. A computer finds it almost impossible to relate facts in different formats, especially if they use a different subset of the language. Thus, the W3 consortium proposed a universal format of information storage that could be understood by an automated system.

The basic unit of metadata in RDF is defined as a triple in the form of *Subject Predicate Object*. The subject is being described in terms of the object by the predicate. As can be observed, it fits almost perfectly as a premise of a syllogism.

RDF is used to describe relationships between entities of data that are difficult to present in a traditional way. As such, it is designed to work with databases that are used to store such information. While the most popular and wide-spread type of database is the relational database, certain aspects of life are difficult to map onto its structure, necessitating unusual approaches. For example, a person can be either male or female. Polish law defines marriage as a relationship of exactly one male and one female. However, this relationship is difficult to map onto the standard structure of a relational database, as it requires splitting the entity of a person into two distinct groups that can have a relation with each other, but not between members of either group alone.

Similarly, if one wanted to design a relational database for storing tags, one would inevitably end up with many-to-many relationships, as a resource can have many tags assigned to it, and each tag can be assigned to multiple resources. Furthermore, such a schema does not easily allow for fuzzy or heuristic searches, and relies on tags being physically assigned to the appropriate resources.

1.3 The content of the thesis

As part of the thesis, I have created a syllogistic engine using Aristotelean logic to perform fuzzy and heuristic searches for tags that might potentially relate to a resource stored in a database, but are not explicitly assigned to it.

The engine is designed as a separate entity that can be incorporated into different systems of information and data storage. It has no metadata knowledge about the type of resources it is describing.

As an illustration of its capabilities, and an example system for its use, I have decided to create a very basic picture gallery. This system was chosen for several reasons:

1. One of the most important senses of a human, besides touch is sight. Our brains are designed to very effectively search for and recognize objects in an image. Therefore, determining a tag truthfully describes and image is fast and easy for a human tester.
2. Using images instead of text documents makes the interface immediately visually appealing to the user.
3. All of the on-line image galleries that I have researched for the purposes of this thesis only ever allow the user to find images described by tags that are assigned to the image. Most provide tag suggestions based on most common related searches, or other tags assigned to the images that were found. Only a select few use a very basic form of relations, that is equivalent to the one-to-many relationship.
4. I have not encountered an image tagging system for single users that would have been satisfactory for me.

The system is designed with a single user in mind, and meant to help them organize their photos, be they vacation or work-related.

The contents of the thesis are as follows:

Chapter 1 contains preliminary considerations. It presents the background, motivations and goals of this work.

Chapter 2 delves into the nature of the syllogism, and the rules and potential pitfalls of the Aristotelean logic system.

Chapter 3 discusses the proposed implementation from the point of view of the architecture of the system.

Chapter 4 delves into the particularities of the syllogistic engine, exploring each implemented algorithm in detail.

Chapter 5 lists the tests that were performed in an effort to provide a functional application.

Chapter 6 concludes the work, describing goals met, issues encountered, and potential expansion opportunities of the application.

2 Syllogistic logic, its uses and machine implementations

2.1 The classical syllogism

2.1.1 The definition of syllogism

As has been said in the previous chapter, syllogistic logic, also known as Aristotelean logic, was one of the first formalized systems of deductive reasoning. The basic form of argumentation used in Aristotelean logic is the syllogism. In his work, Prior Analytics, Aristotle defined syllogism as: “discourse in which, certain things being stated something other than what is stated follows of necessity from their being so.”[6]

The above statement, while broad enough to serve as a general definition of reasoning in any kind of system of deductive logic, was subsequently narrowed by Aristotle to the form consisting of two premises that imply a conclusion.

Each statement in a syllogism is in the form of:

Quantifier Subject Copula Predicate.

Where Subject and Predicate are terms. Of such a statement, we say that the predicate is predicated of the subject. Aristotle divides all terms into singular, for example Plato, and general, for example men. Aristotle further divides all terms into those that can serve as a subject and those that can serve as a predicate. He argues that singular terms can only be subjects, and that general terms can be both a subject and a predicate. Furthermore, it is easy to show that terms describing the state of entities, such as adjectives (for example fast), can only be predicates. Aristotle and later writers predominantly concerned themselves with categorical statements, that is, statements about general terms representing categories of entities, or concepts, rather than single entities, which went largely ignored. Such syllogisms are called categorical syllogisms.

As mentioned above, the subject and the predicate can be terms denoting single entities, collections of entities, concepts, or properties. When considered together, the quantifier and copula from a relationship between the subject and the predicate that can take one of the following forms:

1. All A are B, traditionally denoted as A **a** B,
2. Some A are B, traditionally denoted as A **i** B,
3. All A are not B, traditionally denoted as A **e** B,
4. Some A are not B, traditionally denoted as A **o** B.

A syllogism consists of three such statements, and is written as:

Minor premise
Major premise
Conclusion

The subject and predicate of the conclusion are called the minor and major term respectively. The premise containing the major term is called the major premise, and the one containing

the minor term is called the minor premise. An additional third term is used to relate the major and minor premise. This term is called the Middle term, and will be henceforth denoted as **M**. For the syllogism to be valid, each of the premises must contain the middle term, and it must not exist in the conclusion. To illustrate the above, let us examine the following syllogism:

All Greeks are men.
All men are mortal.
Therefore all Greeks are mortal.

The minor term is *Greeks*, the major term is *mortal*, and the middle term is *men*. Therefore, the minor premise is *All Greeks are men*, the major premise is *All men are mortal*, and the conclusion is *All Greeks are mortal*. The above can thus be written more generally as:

S **a** M
M **a** P
S **a** P

Such a generalized pattern is called a mood. Each mood can be denoted in shorthand in one of two ways. The first is a mnemonic name, which scholars of the middle ages assigned to each of the valid moods. The vowels of each name encode the forms of the major premise, minor premise and conclusion respectively. For example, the name of the pattern above is B**a**rb**a**r**a**.

The second type of shorthand is created by putting together the letter designations of the three statements in a mood in capital letters, followed by the hyphen and a figure number, as detailed below. In this system, the above pattern has the designation AAA-1.

2.1.2 Classifications of syllogisms

Aristotle defined three different types of syllogism, called figures, based on how the middle term relates to the subject and predicate in the premises. These also describe ways one would use to prove the syllogisms in question. They are as follows:

1. By finding M such that M is predicated of S and P of M[1]
2. By finding M such that M is predicated of both S and P
3. By finding M such that both S and P are predicated of M

Later writers recognized a fourth figure, in which M is predicated of P and S is predicated of M. However, as will be shown later, such syllogisms can be reduced to the first figure, and indeed, some logicians, following the Aristotle's own interpretation, included them in the first figure.

The order of the major and minor premise does not affect the validity of the syllogism, and can be changed freely. This gives us three statements in any of the four forms, in one of four figures, for a total of 256 possible syllogistic patterns. However, of these, only 24 are actually valid, that is to say, the conclusion follows logically from the premisses, spread equally

1 Encyclopaedia Britannica states that figure 1 is one such that S is predicated of M and M of P, but that stands in clear opposition to the mood Barbara, and indeed every mood that is classified under figure 1. As such, and seeing that many early logicians did not distinguish between figures 1 and 4, I have interpreted this to be a mistake.

between all figures. The valid patterns are listed in the table below, using their medieval mnemonic names.

Table 2.1: Valid syllogistic moods

Figure 1	Figure 2	Figure 3	Figure 4
Barb**ara**	C**e**s**a**r**e**	D**a**t**i**s**i**	C**a**l**e**m**e**s
C**e**l**a**r**e**nt	C**a**m**e**str**e**s	D**i**s**a**m**i**s	D**i**m**a**t**i**s
D**a**r**ii**	F**e**st**i**n**o**	F**e**r**i**s**o**n	Fr**e**s**i**s**o**n
F**e**r**io**	B**a**r**o**c**o**	B**o**c**a**rd**o**	C**a**l**e**m**o**s
B**a**rb**a**r**i**	C**e**s**a**r**o**	F**e**l**a**pt**o**n	F**e**s**a**p**o**
C**e**l**a**r**o**nt	C**a**m**e**str**o**s	D**a**r**a**pt**i**	B**a**m**a**l**i**p

Source: https://en.wikipedia.org/wiki/Syllogism

These syllogisms can be graphically expressed with Venn diagrams. In a Venn diagram, elements of a set are represented as points on a plane, with the sets themselves denoted as areas. Overlapping areas contain elements common to two or more sets. An example diagram for modus Barbara is shown in Fig. 2.1:

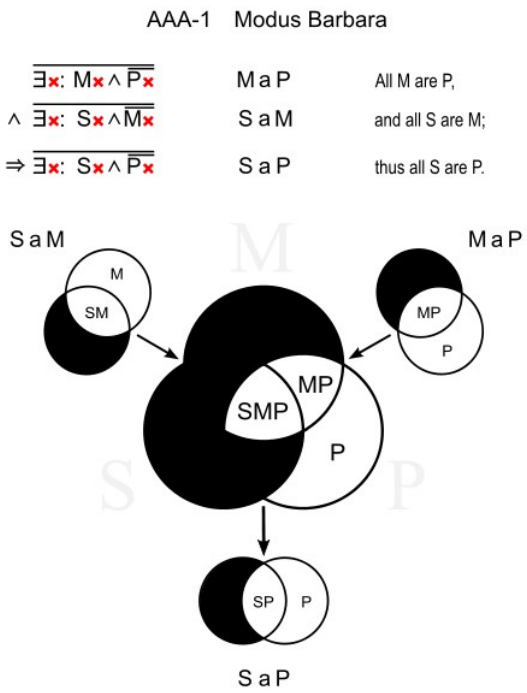

Fig. 2.1: Venn diagram of mood Barbara

Source: https://en.wikipedia.org/wiki/Syllogism

A similar diagram for Darii is shown in Fig. 2.2:

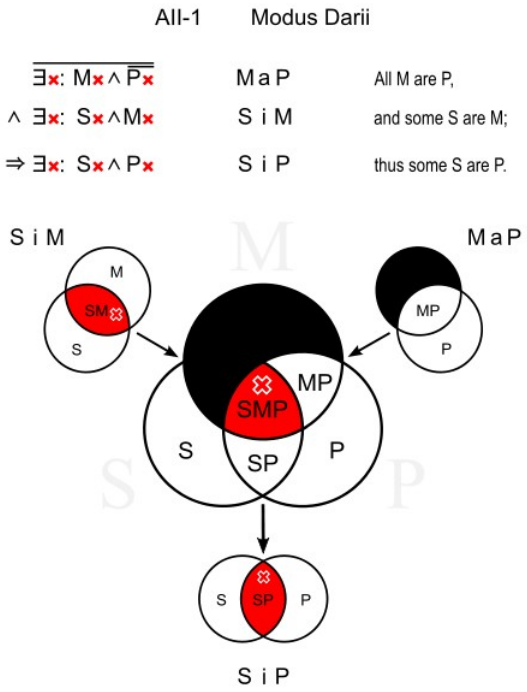

Fig. 2.2: Venn diagram of mood Darii

Source: https://en.wikipedia.org/wiki/Syllogism

Black areas in the graphs above denote empty sets, while red areas denote that only some elements are considered. Equivalent statements using predicate logic are also noted.

2.1.3 Basic operations on syllogisms

Aristotle showed that different forms that share the same subject and predicate are related by a simple set of rules, and in some cases can be changed into one another.

1. If *All A are B* is true, then *All A are not B* must be false, and if *All A are not B* is true, then *All A are B* is false, as they cannot both be true. However, both statements can be false at the same time. These pairs are called contraries.
2. If *Some A are B* is false, then *Some A are not B* is true, and if *Some A are not B* is false, then *Some A are B* is true. Both statements can be true at the same time. Such pairs are called subcontraries.
3. From *All A are B* naturally follows that *Some A are B*, and similarly from *All A are not B* follows that *Some A are not B*. Such pairs are called subalterns.
4. If *All A are B* is true, then follows that *Some A are not B* is false, and if *Some A are not B* is true, then *All A are B* is false. Similarly, if *All A is not B* is true, then *Some A is B* is false and vice versa. These pairs are called contradictories.

These relationships, when put together, create the square of opposition, as shown in Fig. 2.3:

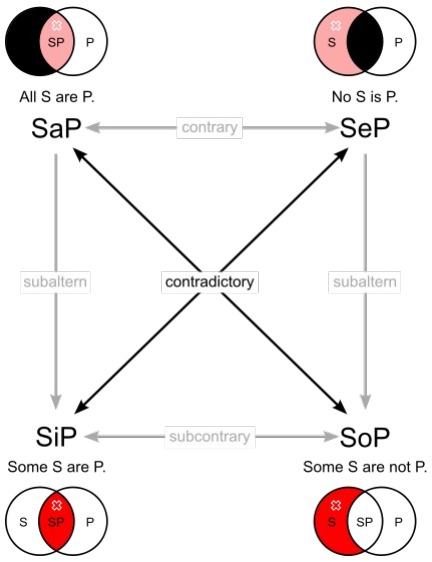

Fig. 2.3: The square of opposition

Source: https://en.wikipedia.org/wiki/Syllogism

Sets containing elements that fulfill the conditions present in the statements are denoted in red. Black denotes empty sets.

In addition to the rules of opposition, in *Prior Analytics* Aristotle defined a number of rules that allowed for the exchange of the subject and predicate of a form. These are called the rules of conversion.

1. If *Some A are B*, then it follows that *Some B are A*.
2. If *All A are not B*, then it follows that All *B are not A*.
3. If *All A are B*, then it follows that *Some B are A*.
4. If *All A are not B*, then it follows that *Some B are not A*.

The conversions 1 and 2 are called simple, because they directly change the order of the terms, without changing the form. The conversions 3 and 4 need to weaken their form from the more general one to a particular one to be valid. These are called conversions per accidens. Using these conversions it is possible to reduce syllogisms of the first form to those of the other forms. As an example, let us examine the mood Bamalip.

P **a** M
M **a** S
S **i** P

Starting from the first figure mood Barbara, as it has the same first letter

M **a** P
S **a** M
S **a** P

using conversion per accidens of the conclusion, we have

M **a** P
S **a** M
P **i** S

subsequently changing the order of the premises, we end with

S **a** M
M **a** P
P **i** S

which is the mood Bamalip with the subject and predicate exchanged.

2.2 Other types of syllogisms

Beside the categorical syllogism identified and described by Aristotle, there exist several variations of syllogism, that were explored by later writers, and the study of which led to other forms of logical argumentation. However, these expanded syllogisms are not the subject of this work.

A modal form is a statement that contains a modifier *necessarily* or *possibly*. Syllogisms that include at least one modal form are called modal syllogisms. Aristotle discussed modal syllogisms in his works, however, his approach was criticized by later logicians as unclear and fallacious.

The hypothetical syllogism, proposed and studied by Theophrasus of Eresus, contains one or more proposition that includes two or more components in a conditional statement. For example, a single proposition might be of the form *if p then q*.

The disjunctive syllogism (known as modus ponendo tollens) is a syllogism that includes at least one disjunctive statement as a premise. That is, it includes a statement in the form of *either p or q*.

The fuzzy syllogism was proposed in 2015 by Mikhail Zachnev and Bora Kumova[7] as an attempt to expand the Aristotelean definition of the categorical syllogism by splitting the particular quantifier *some* into several different ranges operating over fuzzy sets. The object of their work is creating an ontology and a set of formulae for a system of fuzzy reasoning.

2.3 Logical fallacies

The myriad of logical fallacies that can be committed during a discourse can be divided into several broad categories. The two categories important for this work are the material fallacy, and the formal fallacy. Edward Nieznański in his book *Logic* mentions, without explanation, a third category, the pragmatic fallacy. This type of fallacy concerns improper and incomplete definitions of terms, and as such is very hard or impossible to detect automatically.

2.3.1 Material fallacy

A material fallacy is committed when the reasoning includes one or more premises that are false. Because in deductive reasoning the conclusion logically follows from premises, if one

of them is false, then the conclusion might not be true. As Zygmunt Ziembiński points out in *Practical logic*, if the reasoning does not depend on a tautology, then we cannot infer that the conclusion is false from the falsehood of the premises, only that its truthfulness is unknown.[] For example, suppose that the reasoning is as follows:

It rained today,
and Peter's boots are dry
Therefore he had an umbrella.

If Peter's boots are in fact not dry but wet, then it does not necessarily follow that he did not have an umbrella. It might have been that a strong wind made the umbrella useless instead. However, if the reasoning is based on a tautology, then the falsehood of even one of the premises is enough to ascertain that the conclusion is also false.

2.3.2 Formal fallacy

A formal fallacy, also known as non sequitur, occurs when the apparent conclusion does not, in fact, follow from the premises. An example of such a fallacious reasoning is:

All dogs are animals
and all cats are animals
Therefore my dog is a cat.

The above statement commits the fallacy of the undistributed middle, in which the middle statement *animals* is not properly distributed among the premises. As with the material fallacy, from the fact that the reasoning is false, we can not infer that the conclusion is false, only that it does not follow from the premises.

Other formal fallacies, that pertain to syllogisms in particular, are:

1. Illicit major or illicit minor, where a term, other than the middle term, is present in a premise but not in the conclusion.
2. Affirmative conclusion from a negative premise and negative conclusion from affirmative premisses.
3. Exclusive premises, where both premises are negative.
4. Fallacy of the four terms, where a syllogism contains four distinct terms. This is the most difficult type of formal fallacy to detect automatically without extra context, due to the prevalence of homonyms in the common language.

2.4 Uses and machine implementations of syllogisms

2.4.1 Object-oriented programming

Being a logic system concerned primarily with relationships between entities, Aristotelean logic has been implemented, both implicitly and explicitly, in many facets of life. Indeed, a simple implementation of syllogistic forms is employed in object-oriented programming languages. These languages always include some sort of functionality for determining whether or not an entity belongs to a class of objects. When inheritance is present, the rules

relating different objects and classes are as follows:

If class A inherits from class B then all object instances belonging to A also inherit from B, and

If object O is of class A, then some elements of a set of objects of class A are O, but not necessarily all of such objects.

The above rules can be expressed as the syllogistic statements A **a** B and O **i** A respectively. Enforcing the programming rule that *if class A inherits from class B and class B inherits from class C, then class A also inherits from C* is then equivalent to proving the syllogistic mood AAA-1 (Barbara). Similarly, enforcing the rule that *if object O is of class A and class A inherits from class B, then object O is also of class B* is equivalent to proving the syllogistic mood AII-1 (Darii).

2.4.2 Law and judiciary systems

The second major use of syllogistic logic is in law, where it forms the basis of a significant portion of the propositional calculus.

The use of syllogisms in sentencing, and in law in general is a highly controversial topic. In his book, *The Judicial Application of Law*, Jerzy Wróblewski argues that actual deductive reasoning is not always used in deciding a sentence for the accused. Instead, it is just as common for the judge to use deductive reasoning *ex post facto* to justify a decision they have already reached by other means.[8]

The second controversy concerns whether or not a judge should only follow the letter of the law, producing stable and predictable outcomes, or follow the spirit of the law, using inference rather than deduction, as it is the only way to ensure that the law is adequate to the complicated facets of life.[9]

The third controversy arises from an argument as to whether or not the judicial system is a passive executor of the law, or an active participant in the process of its creation.[10] In effect, it is an argument as to whether or not precedent should be followed.

2.4.3 Semantic web and the Resource Description Framework

Semantic web is a concept postulating that, since a computer has great difficulty in understanding the way humans store and process information, we should instead adopt a system for information storage and processing that a computer can understand, but which at the same time reflects the real world as closely as possible. This, its proponents argue, will greatly increase interoperability and access to information of various systems, which would be able to compare and make use of disjointed data much more readily.

To facilitate this goal, the W3 consortium proposed, and is maintaining, RDF - a language for storing information in a machine-readable form. An RDF schema holds information as a relationship graph, where the nodes themselves are entities and data, while the edges contain information about the nature of data in the relationship. An RDF-based system searches for information by traversing the graph, linking entities with other entities via

entailment. In effect, this is equivalent to solving syllogisms, and indeed, Clay Shirky argues that the only data that a semantic web is capable of retrieving is one linked with, and inferred by, syllogisms, which makes possible applications severely limited.[2] []

2 While I agree with this statement in principle, the sheer variability of different kinds of syllogism can provide the flexibility needed by semantic web to compete with other types of databases databases in their traditional applications.

3 The implementation

3.1 Basic assumptions

The application is designed as a small standalone syllogistic engine implementing Aristotelean logic. The engine uses RDF representation of knowledge in an effort to integrate with other modern applications in the semantic web methodology.

The logic engine can be integrated into any number of document management and tag management systems with relative ease. For this example implementation, I have chosen a simple image gallery to be the functional application, allowing the user to browse for images, upload new images and change the logic rules governing the behavior of the engine.

3.1.1 Usability

With the introduction of Web2.0 and HTML5, the scope of possible applications created for the web increased dramatically, while the amount of baggage in the form of peripheral systems needed for their deployment decreased. A web browser has become a de-facto default interface for document databases, as the popularity of Google Docs and numerous other sites attests to. An additional benefit from my point of view, is the relative ease of designing and implementing a graphical user interface with the use of html and css.

The application needs to have real-world application, and be more than an exercise in solving equations.

3.1.2 Portability

One of the main goals I had set for the project, was to provide the ability for the application to be as flexible as possible, able to run on different system architectures and operating systems. To this end, I have chosen a client-server architecture. A web service is the most prominent example of such, being suitable for any system that supports a web browser and online connectivity.

In addition, with the use of a web service, it is possible to design a system in such a way that it can be used online and offline with little to no modification.

3.1.3 Ease of integration

The core component of the application is a logic engine, which has been designed to be easily integrated into other systems. To this end, it provides a number of helper functions for communicating with a database back-end, and a flexible front-end.

While in this example the logic engine is used to provide a fuzzy and heuristic search for images, it does not, in itself, care what is the format of data it is handling. This makes it very easy to re-purpose for text documents, video, database records, and any number of other applications.

3.1.4 Scale

Many existing resource management systems are huge amalgams of functionality, often incorporating a forum, a review and response system, and an extensive rights management

schema. In contrast, my goal was to design a system that would be as lightweight as possible, and would not require a myriad of additional dependencies to be met.

3.2 System architecture

The application is designed as a web service, being part of a client-system design schema. For the purposes of this demonstration of the middle-end and back-end components are hosted within the scope of the web service. However, it is relatively easy to configure the engine to use a remotely hosted database.

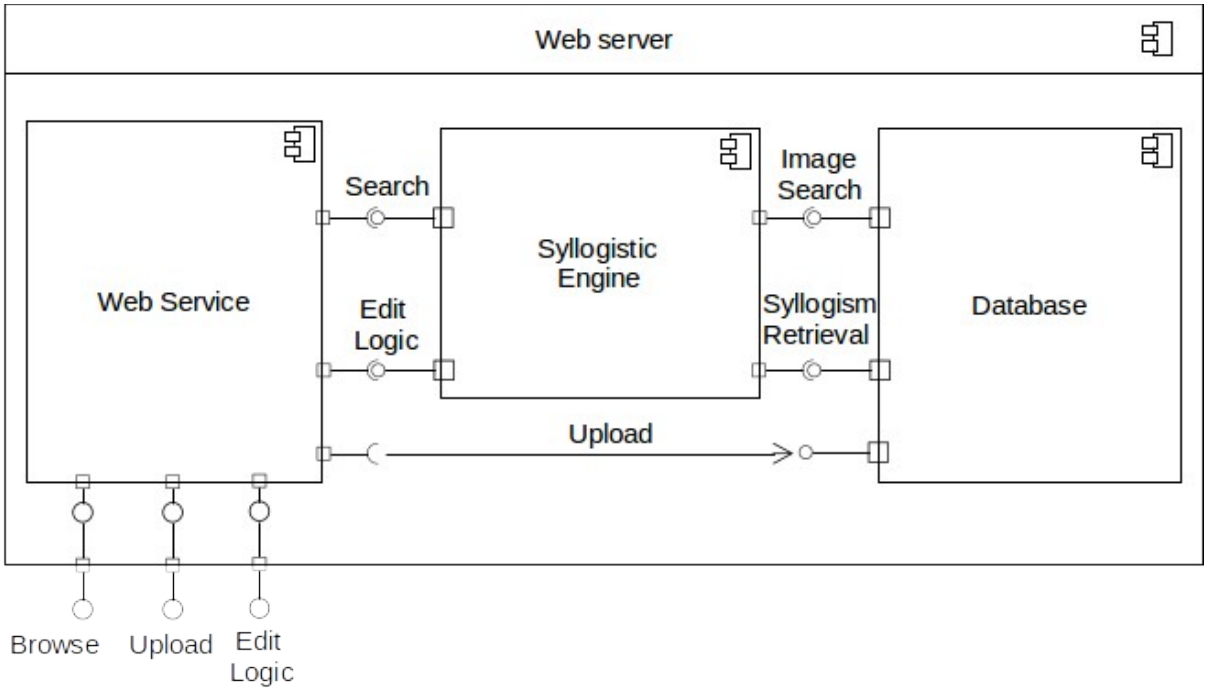

Fig. 3.1: System architecture

The main components of the application are described in detail below.

3.2.1 Web server

I have chosen Apache 2 to be the web server for the example application. This choice had been made due to its popularity and ease of modification. However, this does not limit the portability of the application, as it can be run on any web server capable of supporting Python 3's CGI module. The web server receives requests from a user's browser, and makes a decision whether to send them to the syllogistic engine, or directly to the database.

3.2.2 The syllogistic engine

The syllogistic engine has been written in Python using the RDFlib library for handling the logic model and database interactions. It is a standalone component that can be used to interface with a variety of databases and front-end applications with relatively minor modification.

3.2.3 Database

I have chosen to use BerkeleyDB as the database engine for the application. It is an open-source non-sql database utilizing the key-value model. The main advantage of this choice is that RDFlib natively supports BerkeleyDB for permanent storage of graph data, via the use of a Python library. In addition, BerkeleyDB is lightweight, but at the same time very receptive to expansion.

As a fall-back option, the syllogistic engine can use flat files in the nt or n3 format, which have the added advantage of being human-readable, and so any potential inconsistencies in the data can be relatively easily found and removed.

3.2.4 Front-end

The front-end of the application is managed by the user's web browser. All of the user interfaces are designed as web pages generated dynamically using Python CGI scripts run by the server. These scripts are independent of the syllogistic engine and database, and interface with them via the use of helper functions.

3.3 User interface and use cases

The application offers three major functionalities, shown in Fig. 3.2:

1. Adding new images

2. Browsing for images

3. Management of the syllogistic engine ruleset.

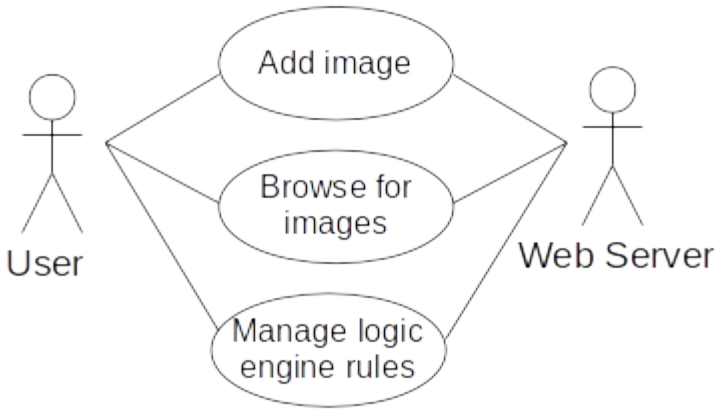

Fig. 3.2: Main functionalities of the application

Each of the functionalities has been provided with its own distinct web document, which encapsulates all of the needed logic. Each functionality is explained in detail on subsequent pages. The user can edit basic configuration by using the Settings option.

3.3.1 Adding new images

The functionality to add new images to the gallery is provided by the *Upload* web document. This functionality allows the user to save a selected image from their computer on the server in a dedicated directory, and subsequently to assign a number of keywords, or tags, to it.

This is accomplished using a series of steps shown below.

A diagram of the functionality provided by the *Upload* web document is shown in Fig. 3.3. The main agent for this process is the web service itself. A combination of JavaScript and Python is used to upload, rescale and assign tags to the image. The syllogistic engine can be referenced during this process, if the user so chooses.

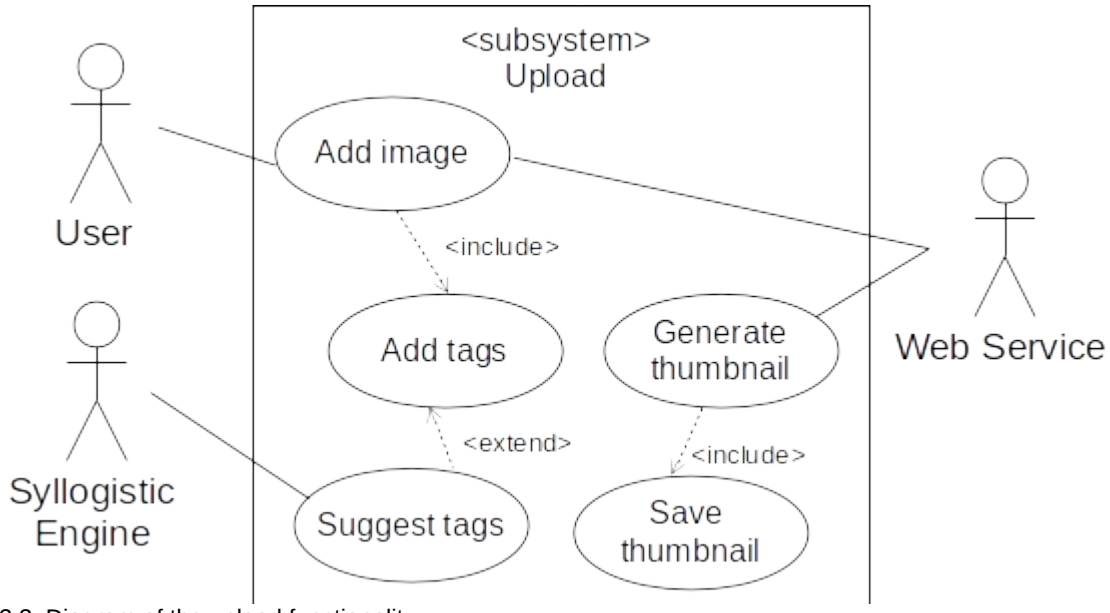

Fig. 3.3: Diagram of the upload functionality

When the user selects the *Upload* option from the main options bar at the top of the screen, he is presented with an image select prompt, shown in Fig. 3.4. This allows him to select one image from his hard drive to be saved on the server.

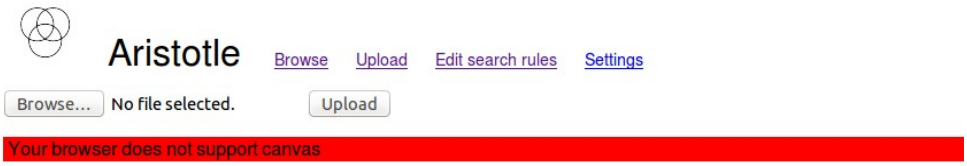

Fig. 3.4: The first screen of the upload functionality, showing an error message

The application uses a Canvas HTML element to preview the image being uploaded to the user. This serves a dual purpose. First, it allows the user to verify that they have selected the correct image. Second, the preview is saved in a separate file, along with the uploaded image. When using the Browse functionality, this preview will be shown in the search results. As we can see in Fig. 3.4, if the user's browser does not support the Canvas HTML element, the user will be notified of this fact. Here, we have simulated this behavior by disabling JavaScript for the browser. A valid image file must be selected in order to proceed.

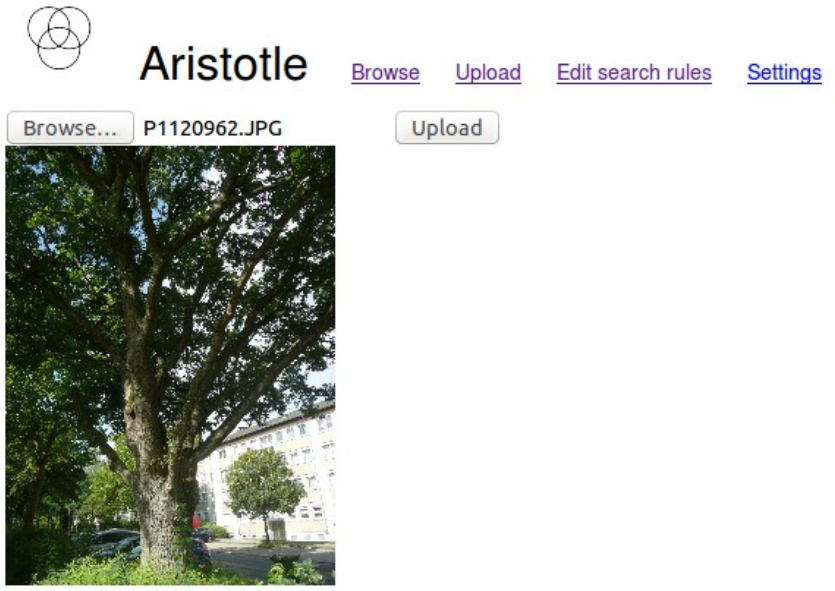

Fig. 3.5: Image preview

After the user selects an image file, the application displays a scaled down preview (popularly called a thumbnail) below the selection prompt. The name of the selected image is also displayed. The user is free to select a different image if they prefer. The preview display will update accordingly. However, only the last selected image will be uploaded.

After clicking the *Upload* button both the image and the thumbnail are saved on the server. As we are not able to stop the user from accidentally leaving the page and aborting the upload process, a special <*None*> tag is assigned to the image at this step, to signify the image has no proper tags assigned yet. This is to enable the administrator to find the image in an event that the user aborts the upload process at this stage. Once the proper tags are assigned to the image, this tag is removed. After this stage is completed, the user is taken to the tag assignment screen.

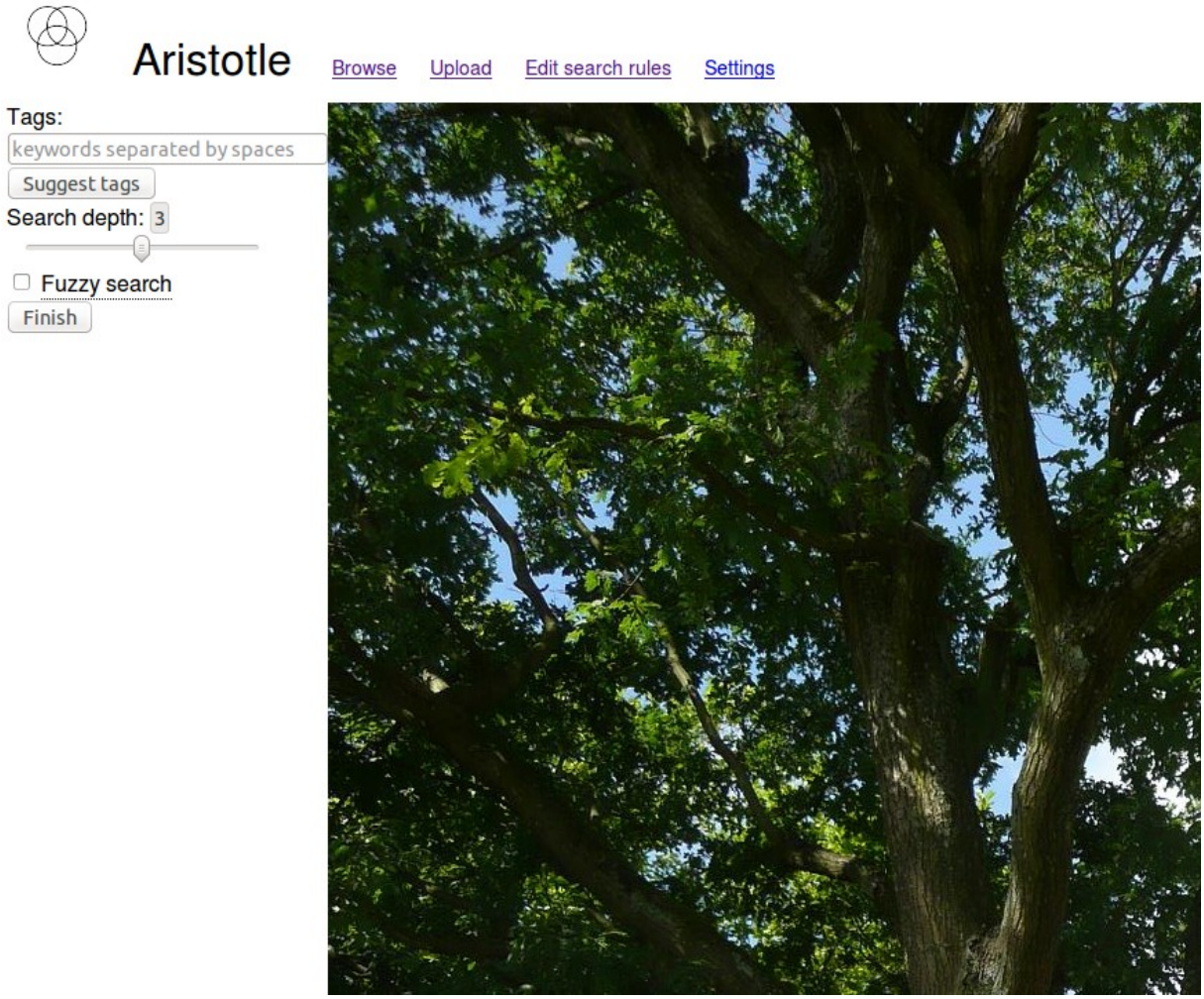

Fig. 3.6: Tags assignment screen

The screen shown in Fig. 3.6 allows the user to assign tags to the image they have uploaded. The image is displayed in full on the right side of the screen. The user is now able to use the search box to type in keywords that they think best represent the contents of the image. Filling in this field is required to proceed. There is no practical limitation on the number and nature of these tags, however, they are assumed to be plain text words, and will be treated as such.

After the user has typed the tags they prefer, they have two options:

1. They can finish the upload process by clicking the *Finish* button. The image will be saved with the tags assigned to it by the user and ready for viewing.
2. They can ask the system to suggest extra tags for the image, based on the ones they have entered, by clicking the *Suggest tags* button. The system will then use the syllogistic engine to determine a list of tags logically connected with the keywords entered, as seen in Fig. 3.7. The *Suggest tags* button may be clicked multiple times to retrieve a new list of suggestions in case the user changes their selection. This does not happen automatically.

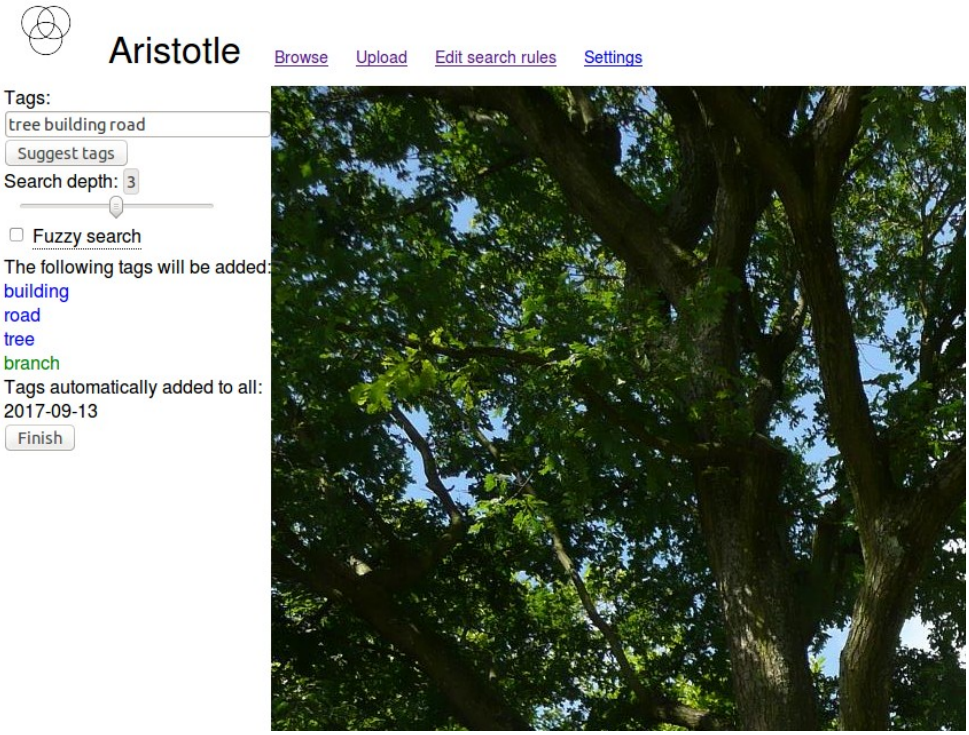

Fig. 3.7: Using the *Suggest tags* option

When asking the syllogistic engine to provide suggestions, the user has two options they can use to customize the search behavior. Search depth is a hard cap on the number of iterations that the syllogistic engine will perform, and is used to limit the amount of tags retrieved. Its value can be a natural number ranging from 1 to 5.

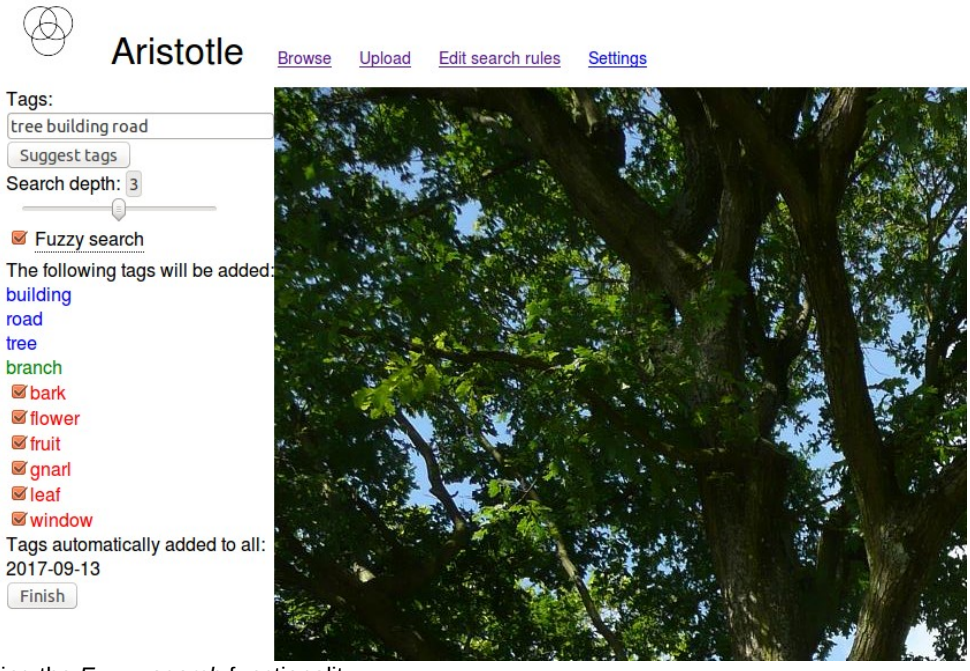

Fig. 3.8: Using the *Fuzzy search* functionality

Enabling *Fuzzy search* tells the engine to include tags related by the *Some are* syllogistic form in the search results. This functionality is designed to widen the search by possibly providing some false positives. However, it can suggest a tag that was perhaps not immediately obvious. The user is provided with the option to select or de-select fuzzy tags by the use of checkboxes. Any tags that are de-selected will not be assigned to the image.

As can be observed in Fig. 3.8, the search results are color-coded. Blue denotes the tags that the user had entered in the search box. Green denotes Direct tags. These are the tags that are linked to the user-supplied ones by syllogistic relationships of form **a**. Red denotes Fuzzy tags. These are linked to either the user-supplied tags or the Direct tags via syllogistic relationships of form **i**. Finally, the system lists Universal tags. These are automatically applied to all uploaded images. As seen in the picture, currently the only universal tag is the date of the upload.

It is important to note that the direct tags cannot be de-selected by the user. This is because they logically follow from the syllogism of mood Barbara, and as such are assumed to be always present together with their subject tags. If the user wishes to change the direct tags, they must change their selection in the search field and press the *Suggest tags* button again.

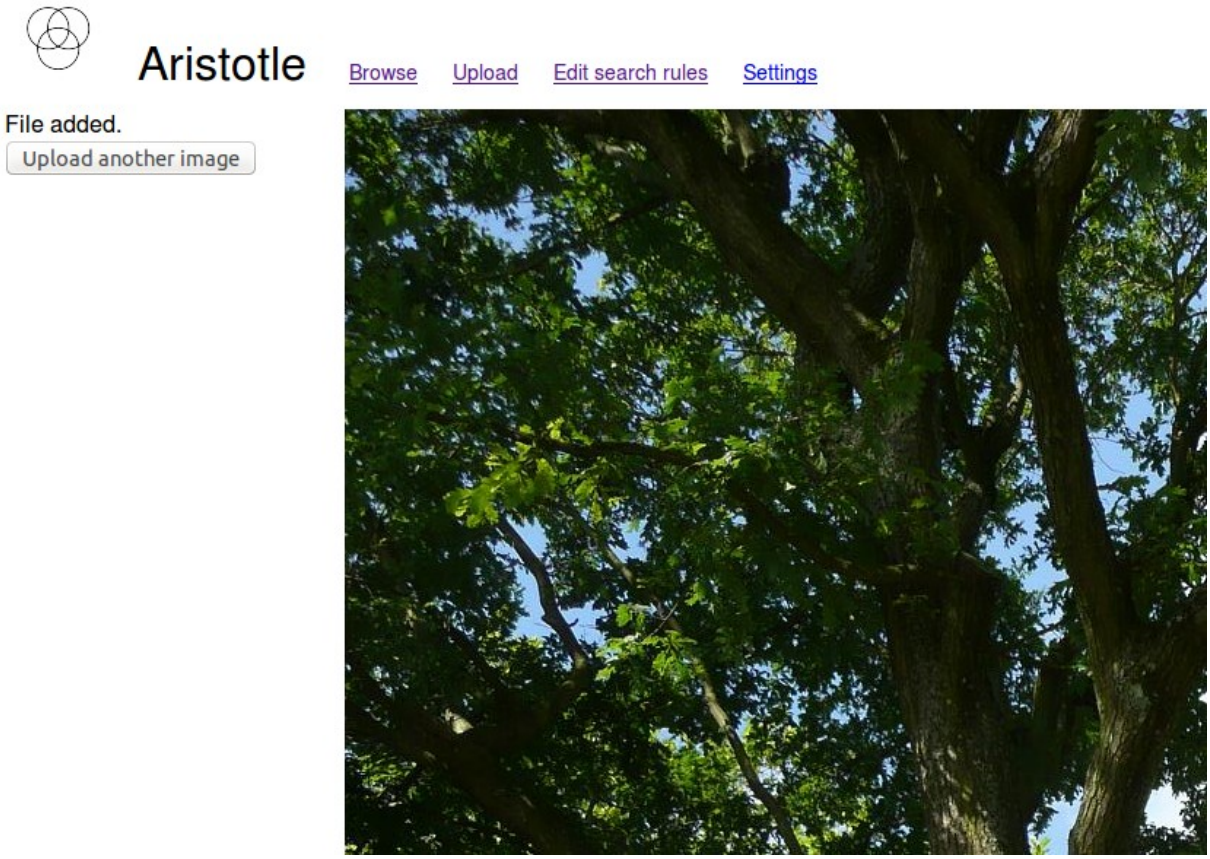

Fig. 3.9: Upload finished

After the user is satisfied with their selection, they may click the *Finish* button to finalize the upload process. Assuming they have assigned any tags to the image, the application removes the <None> tag and replaces it with a list of tags the user has chosen. The user is then able to upload another image if they so choose.

3.3.2 Browsing for images

The functionality to browse for images in the gallery is provided by the *Browse* web document. This functionality allows the user to search for images previously uploaded to the server, and subsequently choose to view or edit one of them. This is accomplished using a series of steps shown below.

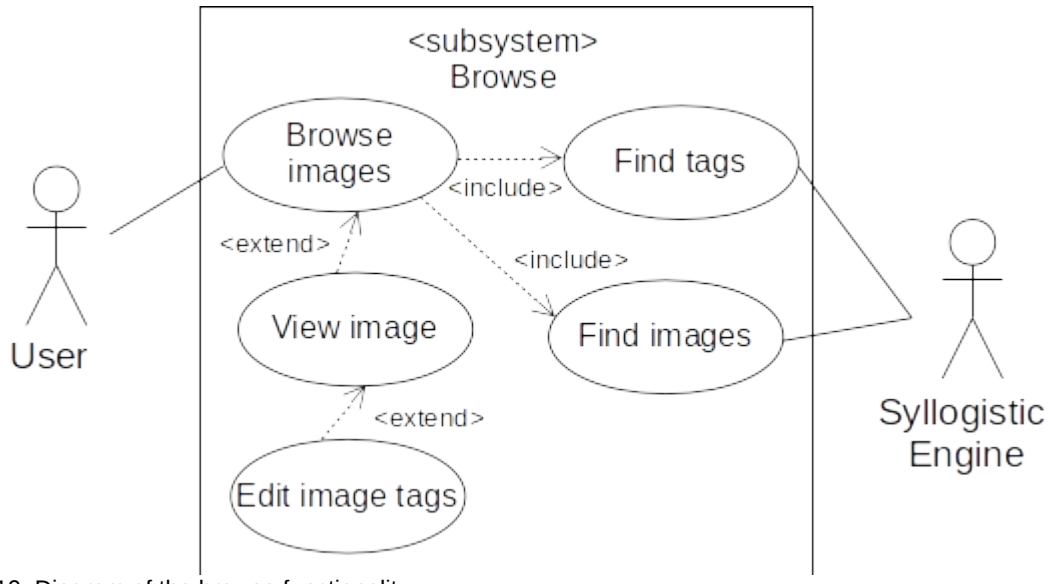

Fig. 3.10: Diagram of the browse functionality

As can be observed in Fig 3.10, the main actor active in the *Browse* subsystem is the Syllogistic engine. The web service only acts as a carrier, passing information between the user and the engine, without performing any additional functionality.

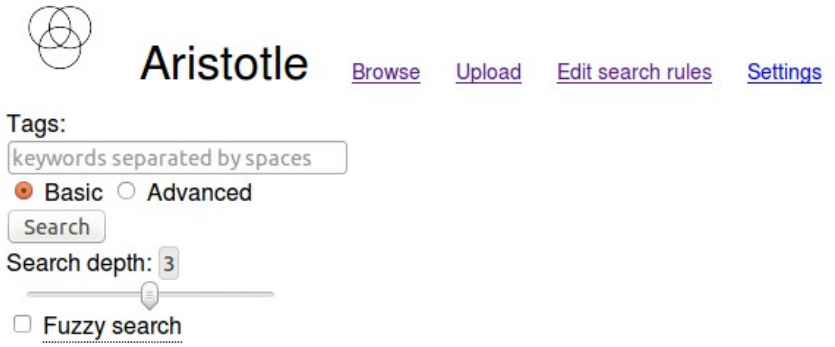

Fig. 3.11: The first screen of the browse functionality

When the user is first shown this screen, it is blank. An alternative behavior would be to show all images in the gallery, and let the user narrow down his search. However, as the purpose of the application is mainly for demonstration, I have decided to leave the results blank to facilitate easier change recognition.

The search menu shown on the left side of the screen allows the user to enter multiple tags they would like to search by. As with the upload functionality, the tags are assumed to be single word descriptors in plain text. Thus, any URLs or other special combinations entered will be treated as raw strings by the syllogistic engine. This behavior is set by the web service itself, and can be changed without modifying the engine itself.

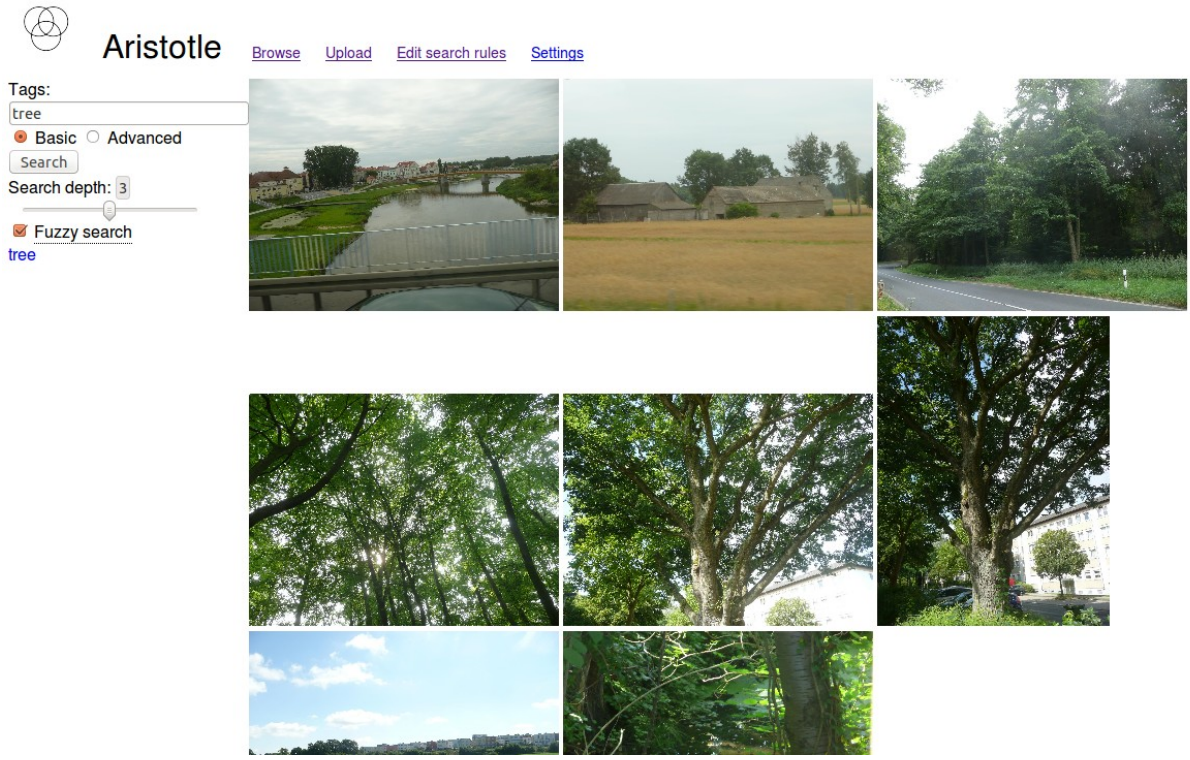

Fig 3.12: Basic search for trees

The user has the ability to choose between a basic and an advanced search. The basic search will not retrieve any more keywords, and will only show images which are tagged by any of the tags in the user's input. This mimics the functionality of regular online galleries. The query is still passed to the syllogistic engine, but it only acts as an intermediary between the web service and the database, without performing any other operations. Thus, other options have no effect.

As can be seen in Fig. 3.12, performing the simple search only yields a single tag, the one we have entered. The application found eight pictures matching this tag. It can be easily determined that they indeed contain trees to some extent. After the results have been retrieved, the user is able to select one of the images by clicking on it. The application will then show an image display screen, as seen in Fig.3.13. We shall explore the functionality of this screen in detail later. For now, it is enough to observe that the *tree* tag is indeed displayed on the left side of the screen.

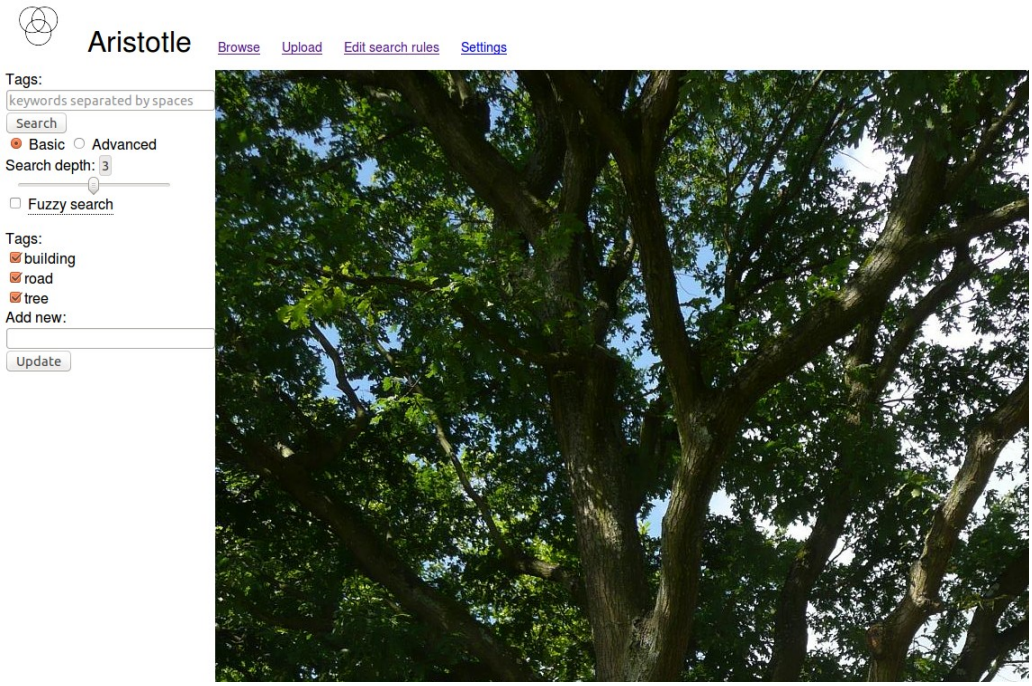

Fig. 3.13: Image display functionality for a single image tagged with the *tree* tag

The advanced search functionality engages the syllogistic engine. The user input is passed as the beginning keywords to the syllogistic_search algorithm within the engine. This algorithm is explained in detail in the following chapter. The engine finds tags that are directly linked with the search terms by syllogistic form **a**. These are called the direct results. Search depth regulates the number of iterations the algorithm will perform.

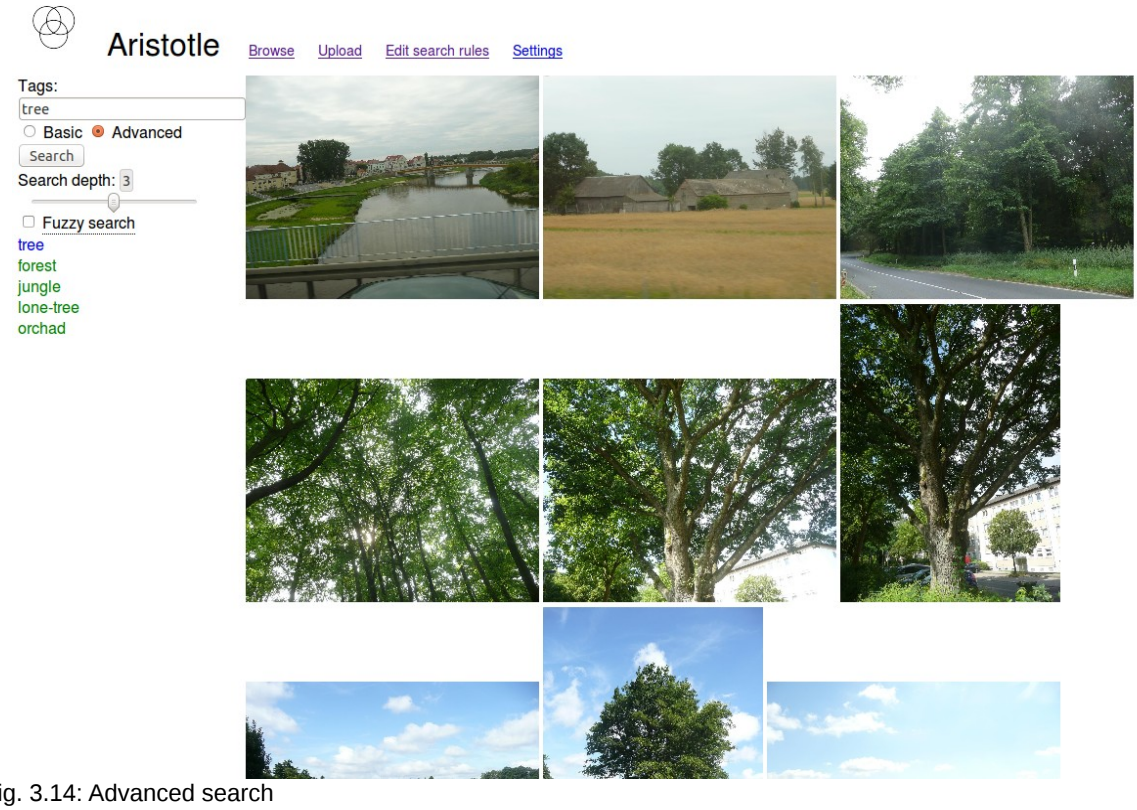

Fig. 3.14: Advanced search

As can be observed in Fig. 3.14, engaging the advanced search has resulted in additional images being found. The search terms entered by the user are once again listen in blue on the left side of the screen. Below the search terms the application listed, in green, the tags related to them by the syllogistic form **a**.

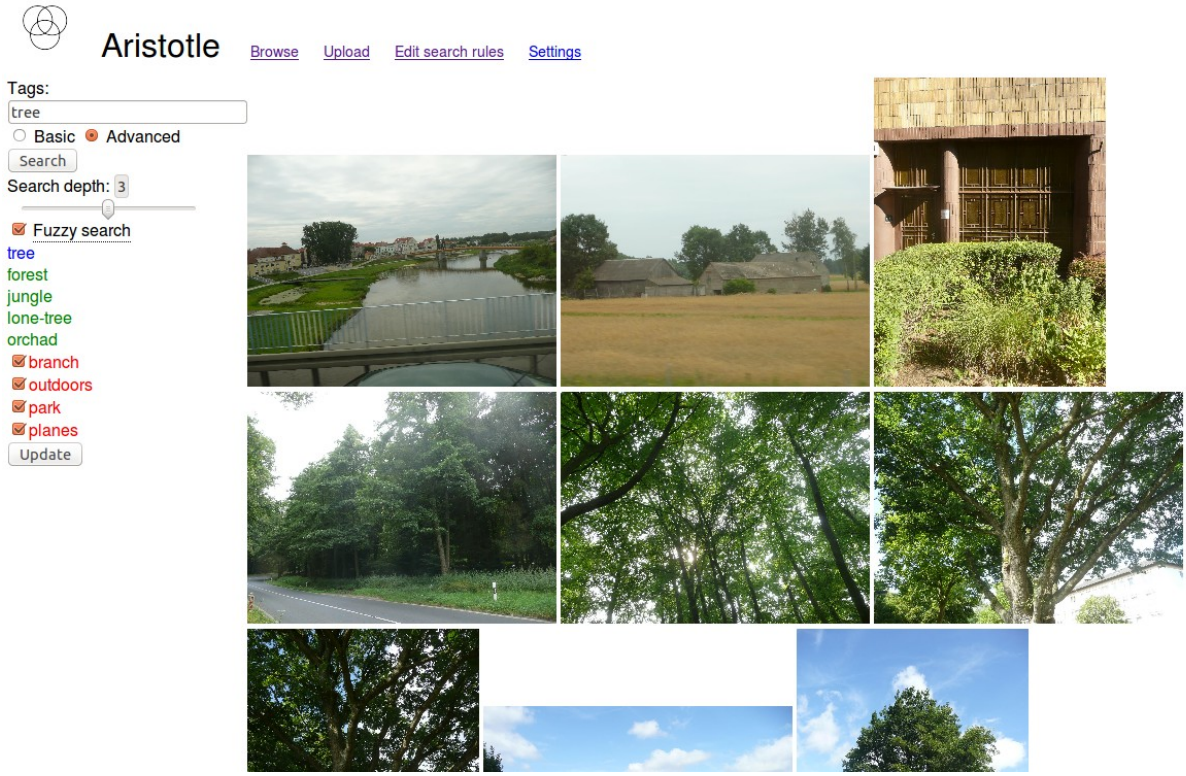

Fig 3.15: Advanced fuzzy search

Selecting the *Fuzzy search* option is only meaningful when the Advanced search is also selected. If this option is chosen, in addition to the direct results, the syllogistic_search algorithm returns tags related to either the keywords provided by the user or to the direct search results by the syllogistic form **i**. These are called fuzzy results and are depicted in red in Fig. 3.15. Additionally, the *Update* button is displayed. Because the images tagged with fuzzy tags might, but not necessarily do actually include features described by the keywords supplied by the user, they are provided with an option to toggle them on or off. If the user de-selects one or more fuzzy tags and subsequently clicks the Update button, the search will be performed again, excluding the de-selected fuzzy tags.

As we can see in Fig. 3.16, after de-selecting the fuzzy tag b*ranch* and clicking *Update*, the list of images on the right side of the screen has changed, and is now missing the third picture in the first row. As can be easily seen, this picture did not in fact include any trees, and instead was listed only with the *branch* tag due to the bush in the foreground.

The user is free to change his selection of the fuzzy tags, selecting or de-selecting them in any combination they desire. The list of images will change accordingly every time the *Update* button is clicked.

Worth noting is an interesting functionality that is perhaps not first apparent. Since clicking either of the two buttons performs a new search, if the user changes his search terms and clicks *Update* instead of *Search*, the engine will retrieve a new list of tags, but only the fuzzy tags common to the new set and the user's previous selection will be pre-selected.

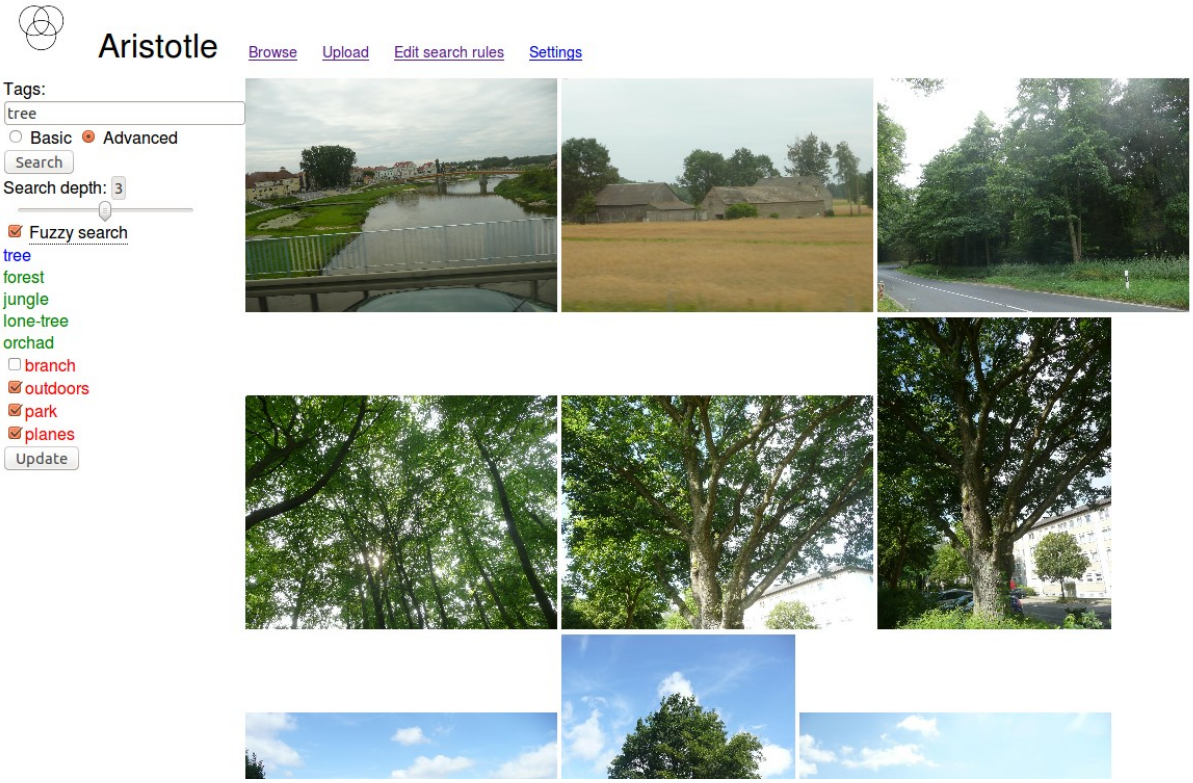

Fig. 3.16: Updating search results after de-selecting the fuzzy tag *branch*

As mentioned before, when the user has found an image they want to see, they can click on it to go to the image display screen. The selected image will be shown in full on the right side of the screen. The very left side is occupied by the search sidebar, which also lists the filename of the image, and any tags assigned to it. Let us select the bottom middle image.

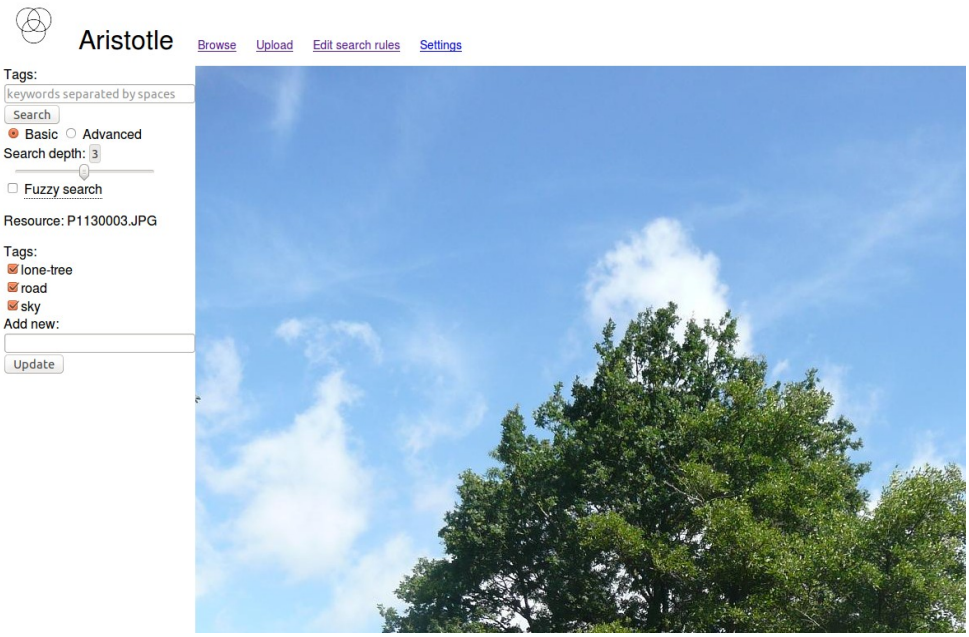

Fig. 3.17: The image display functionality

If the user so chooses, they can modify the tags assigned to the displayed image. To add a

one or more tags, the user types them into the space provided and clicks the *Update* button. Let us add the following tags to the image from Fig. 3.17: *car, cloud,* leaf. As can be seen in Fig. 3.18, after clicking the *Update* button, the new tags are displayed together with those already present.

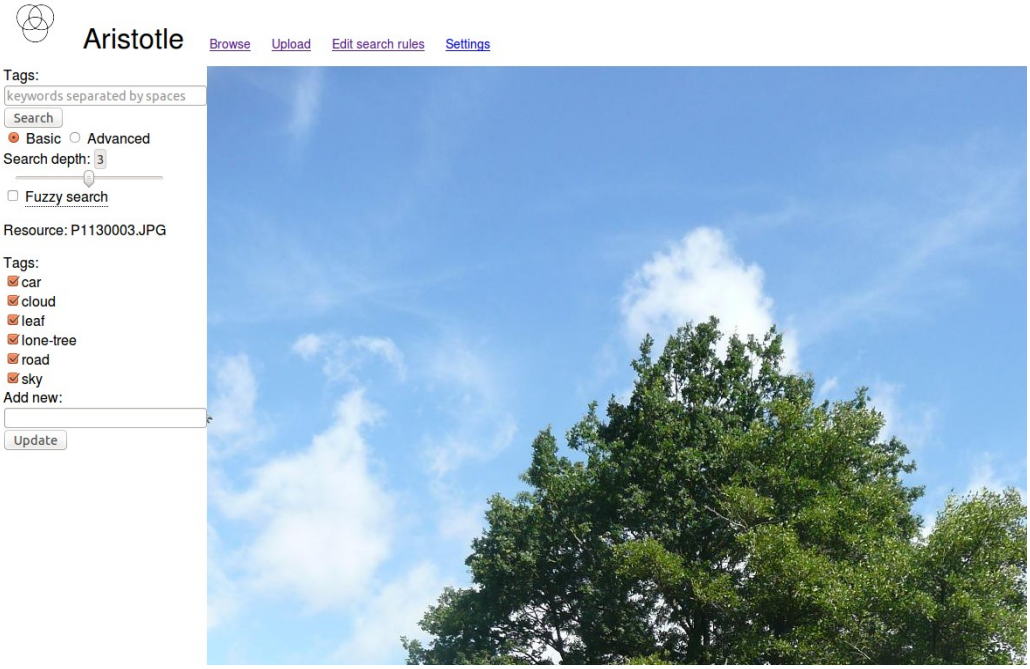

Fig. 3.18: Adding tags to an existing image

If the user wishes to remove any tags assigned to the image, they can de-select them from the list and then click *Update.* Let us remove the tag *car* from the image. The result can be seen in Fig. 3.19.

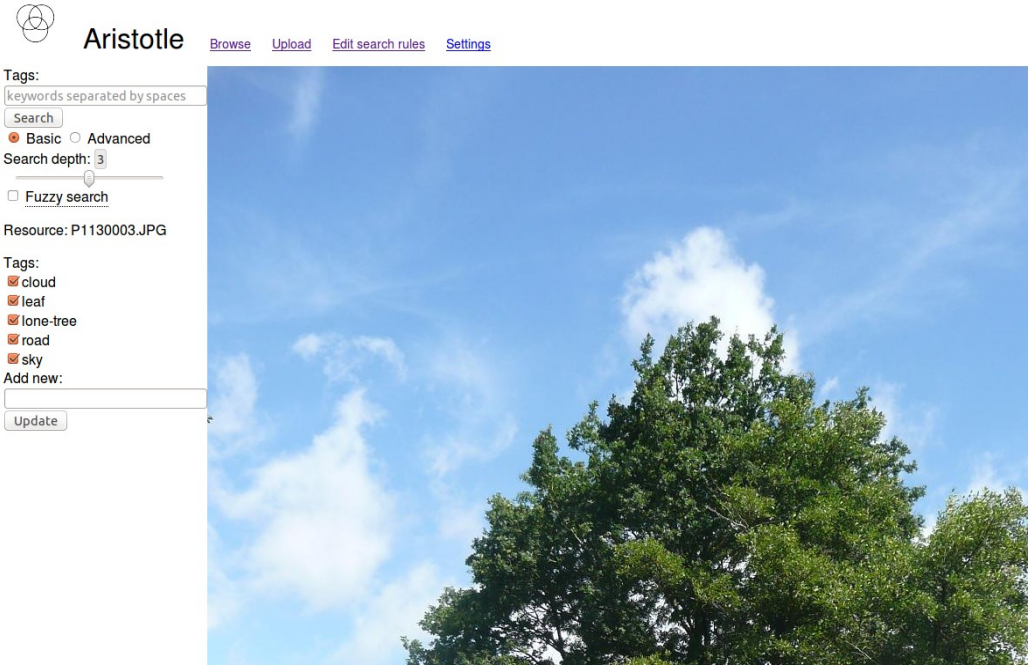

Fig. 3.19: Removing an existing tag

3.3.3 Editing syllogistic rules

The functionality to edit logic rules that the syllogistic engine uses is provided by the *Edit search rules* web document. This functionality allows the user to list and modify rules that already exist in the syllogistic database, removing them if necessary. The user can also add new rules if the desired ones do not exist.

A diagram of the functionality provided by the *edit search rules* web document is shown in Fig. 3.20. The main agent for this process is the syllogistic engine, with which the user interfaces by the means of the web service.

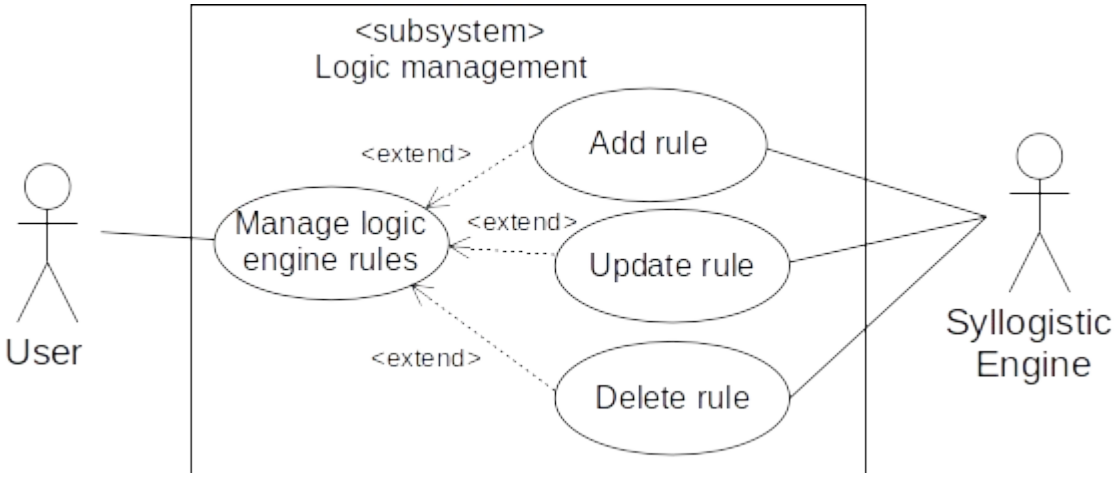

Fig. 3.20: Diagram of the *Editing logic rules* functionality

When the user clicks on the Edit search rules link in the menu bar at the top of the screen, they are shown the first screen of the logic management module (Fig. 3.21). The user is able to search for rules pertaining to specific terms by the use of the search functionality present on the page. As can be seen in Fig 3.22, every rule containing the searched term will be retrieved, irrespective of its position as subject or object of the statement.

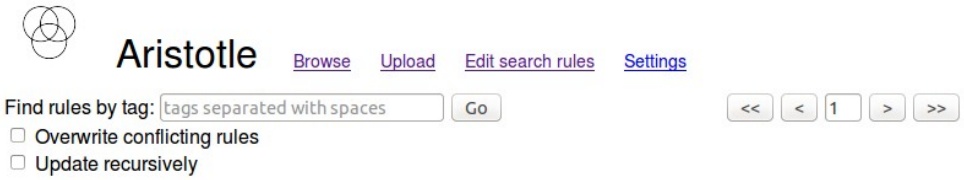

Fig. 3.21: Main screen of the logic management module

The screen will show up to 20 rules per page. If the search results fill more than a single page, the user can use the navigation buttons to move through them.

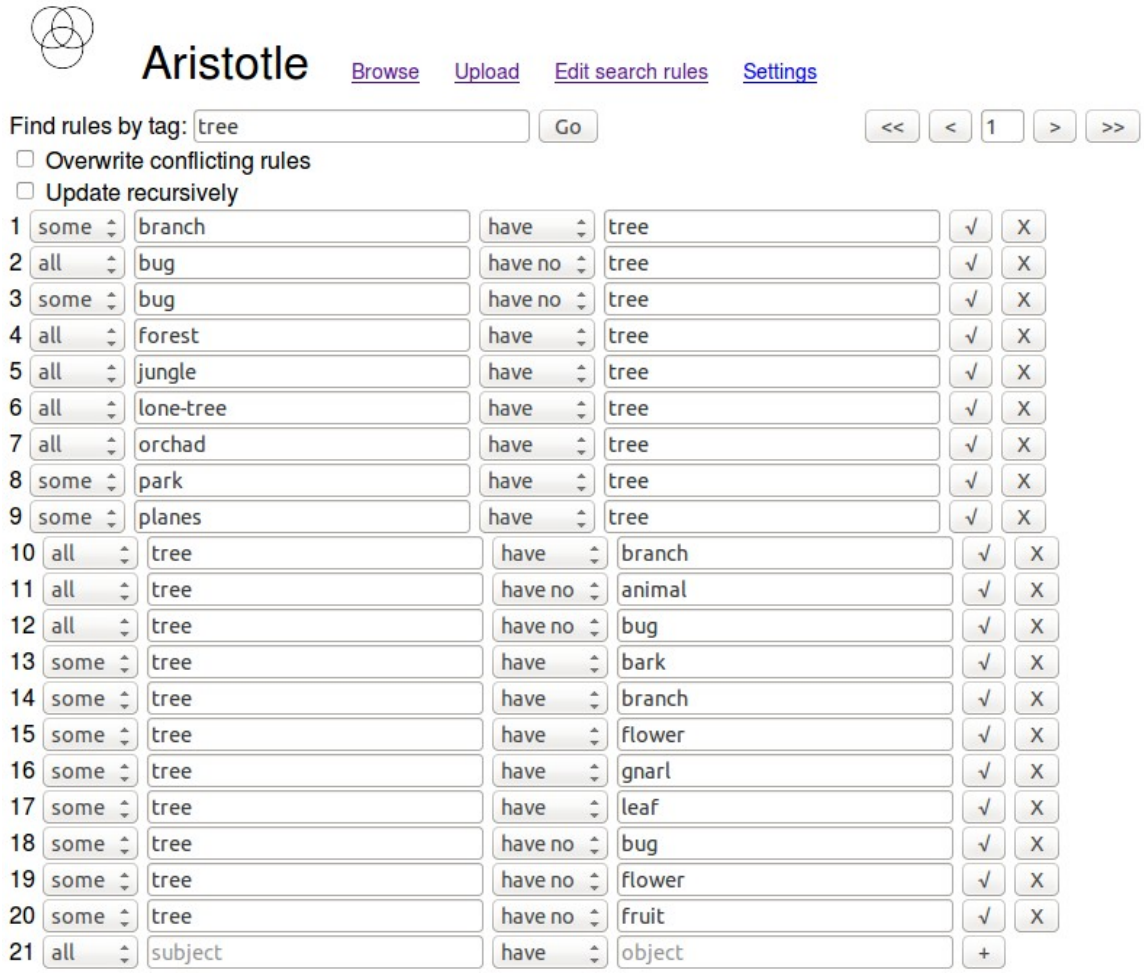

Fig. 3.22: Search results for rules containing the term *tree*

The last line will always be empty and is designed to allow the user to add new tags to the dataset. To do so, the user types the subject and object of the rule in the textboxes provided, and selects the appropriate form using the two drop downs. Clicking the **+** button will run the add_rule algorithm.

To delete a rule, the user must first locate it in the list. To help them in this task, the rules are sorted alphabetically. Clicking the **X** button next to the desired rule will engage the delete_rule algorithm.

To modify an existing rule, the user needs to locate it, as per the delete functionality. They are then able to freely change the parameters of the rule, putting in new terms as subject and object, and modifying the form. Once the user is satisfied with their choice, then can click on the **√** button to commit their choice and engage the modify_rule algorithm.

The behavior of these algorithms can be controlled by selecting the options below the search bar. They are explained in detail in the following chapter.

4 The syllogistic engine

The main part of the application consists of the syllogistic engine supported by a triplestore non-sql database. It consists of several functions which interface with the logic database. Due to the nature of the application, and keeping in mind usability, I have decided to give the engine several behaviors, selectable by the user. These behaviors will be explained in detail below.

4.1 Basic assumptions

Logic rules in the application are expressed as syllogistic statements. These statements are stored in the database as triples of the form *Subject Predicate Object*, and are handled by the engine the same way. This means that we cannot explicitly store information about the true-false state of a statement. We can only store this information implicitly by the presence or absence of rules. As a result, it is assumed that all statements present in the database are true, and all statements absent from the database are false. It follows that proving the truthfulness of a statement is equivalent to adding it to the database, and proving the falsehood of a statement is equivalent to removing it from the same.

4.2 Searching for tags

The most common operation performed by the application is searching for tags. It is performed while browsing for pictures, as well as while suggesting tags for an uploaded picture. Both of these cases are handled by the *syllogistic_search* function and its wrapper *get_tags*. However, each requires a slightly different behavior, due to the nature of pictures and how it relates to tags.

4.2.1 Suggesting tags

Let us assume that the ruleset in the database is as follows:

forest a tree
tree a branch
park i tree
forest a outdoors
forest i jungle
forest i pine-forest
tree i leaf
outdoors i forest
outdoors i park
lone-tree a tree
sky i cloud

and that the user has entered the following tags into the search box:

forest sky

We are interested in finding tags that could potentially be associated with the image being uploaded. Similarly, when searching for images, we are interested in determining which tags are assigned to the images the user might want to retrieve. Thus, we are not interested in rules of the form **e** or **o**. These are used only for completeness, and to detect potential inconsistencies. The syllogistic form **i** is the universal fuzzy quantifier, allowing us to retrieve more results at the cost of potential false positives.

To suggest tags based on syllogistic rules, we assume implicitly that the picture itself has the tag <Image>, and that all tags entered by the user, referred to henceforth as keywords, are predicated of it in form **a**. The engine creates an input set, and three output sets: Direct, Indirect, and Universal. These will hold tags predicated of the keywords in form **a**, tags predicated of the keywords in form **i**, and tags that are added to all images, respectively. The input set is seeded with the keywords.

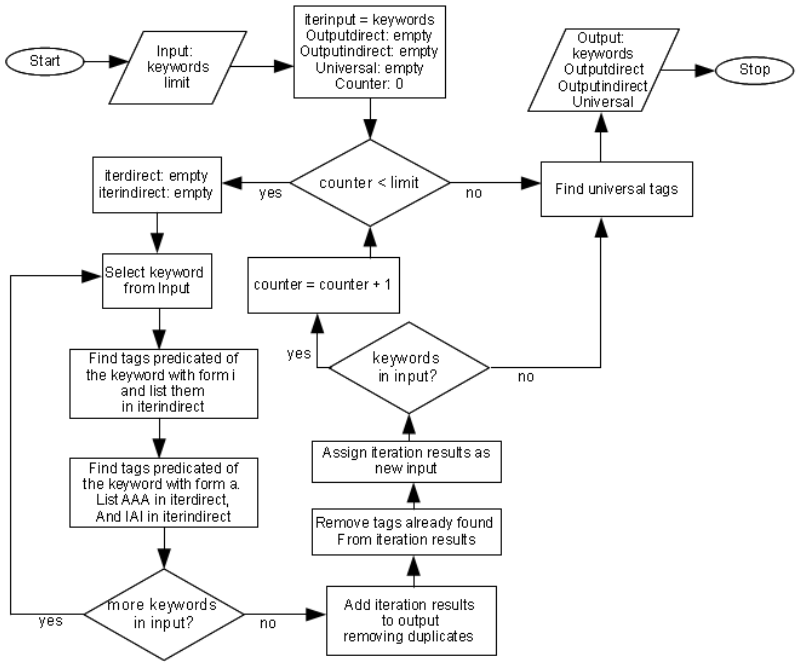

Fig. 4-1: Suggesting tags

The engine then performs the number of iterations that the user has specified. This serves to limit the scope of a search, and in doing so, time taken, and the number of tags returned. In each iteration the database is searched for all tags predicated of the elements of the input set. These are divided into direct and indirect sets for this iteration. This is equivalent of proving the following moods: AAA, IAI, and AII. The engine prioritizes AAA over AAI, as it is the stronger form. This is called forward search.

Following the same priority, if it happens that the tags are present in both the direct and indirect set, the direct set takes priority, and so they are removed from the indirect set.

Finally, the iteration sets are merged into the output sets, and the input set for the next

iteration becomes the sum of this iteration's sets, with any tags already present in the output sets at the beginning of the iteration removed. This is done specifically to prevent loops from occurring. The algorithm stops when one of the following conditions are met: either the iteration input set is empty, or the number of iterations has been reached. The algorithm in block form is presented in Fig. 4.1.

After the iterations are complete, the engine populates the Universal output set with the universal tags, that is tags that are automatically added to all images, as specified in the rules retrieved from the database.

In the example above, assuming the number of iterations is 5, the output sets will contain the following tags:

Direct:
tree
branch
outdoors

Indirect:
jungle
pine-forest
leaf
cloud
park

Universal:
None

4.2.2 Searching for images

Let us assume the ruleset from the previous example, and that the user entered *tree* in the search field. To search for images, the engine implicitly assumes that every image is equivalent to the tag <Image> and is predicated of the keywords in form **a**.

As above, the engine performs iterations over the ruleset up to the number specified by the user. However, instead of searching for tags predicated by keywords, we are searching for tags that the keywords are predicated of. This is because if the user is searching for all images containing a tree, then they will probably also be interested in images of a forest. The universal tags are superfluous, unless specifically searched for.

Assuming the number of iterations is once again 5, the output of the sets will contain the following tags:

Direct:
forest
lone-tree

Indirect:
park
outdoors

The algorithm is presented in block form in the image below.

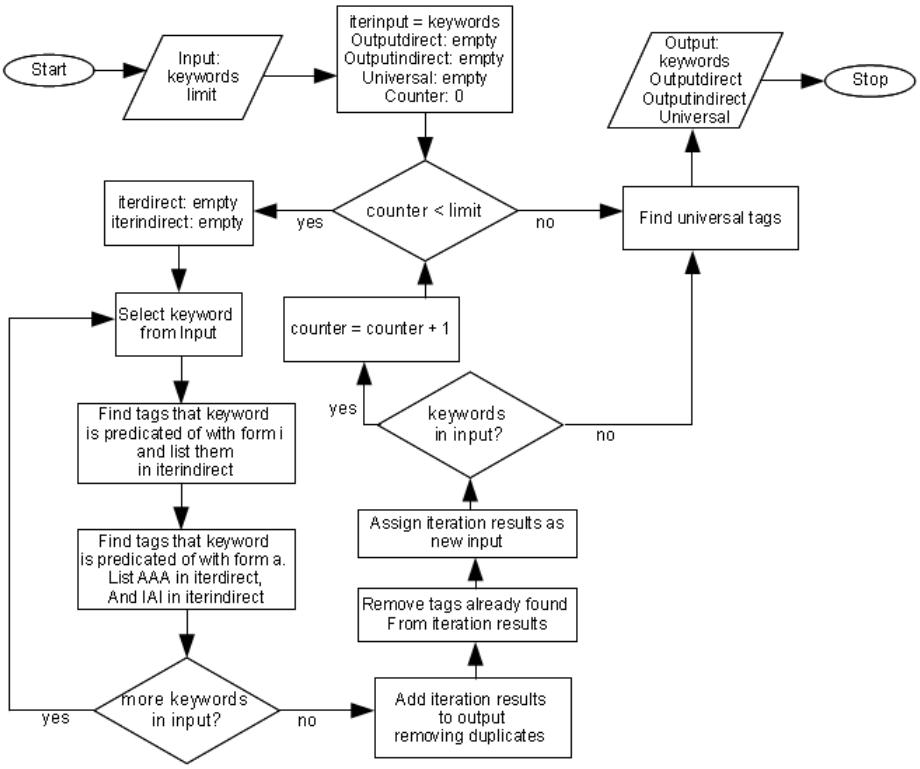

Fig. 4.2: Searching for image tags

As is apparent, from Fig. 4.2, the image search algorithm is almost identical to the tag suggesting algorithm. The main difference, as explained earlier is the 'direction' of movement through the rules graph. To suggest tags we move 'forward', proving a syllogism, while to find images we move 'backward', exchanging the subject and predicate of each premise.

4.3 Editing logic rules

Editing of the logic rules that govern the behavior of the engine is perpetuated by three functions:

add_rule
modify_rule
delete_rule

These functions can work in one of two modes:aggressive and defensive. In aggressive mode, when a user modifies the rules, and the new rule conflicts with ones already present, they will be modified as well (added or removed) to satisfy the square of opposition.

In defensive mode, if a rule that is being changed conflicts with an existing rule, the user will be notified of the conflict, and the engine will abort the action, preserving the current state of the ruleset. This is to prevent accidental deletion or modification of a large portion of the ruleset.

In addition, each function can work recursively, adding and removing additional rules based on inference.

The engine also includes a helper function find_rules, which retrieves rules from the database that are associated with provided keywords.

4.3.1 Adding rules

To add a rule, the user needs to specify all of the components of the new statement, that is the quantifier, predicate, subject and copula. The engine then searches for duplicate rules. If one exists, the user is notified of this fact and the algorithm exits.

If this basic check is passed, the algorithm then checks if any rules conflict with the new rule, and depending on the setting chosen, either overwrites the conflicting rule, or notifies the user of the conflict and exits.

If the recursive option is chosen, the final action is to recursively add rules that are inferred from the new rule by solving syllogistic moods. If any conflicts occur during this process, the algorithm will act accordingly depending on the mode, either by overwriting the offending rules, or notifying the user and exiting. Only after all the changes have been successfully completed is the change committed to the database. The schematic of the algorithm is shown in Fig. 4.3.

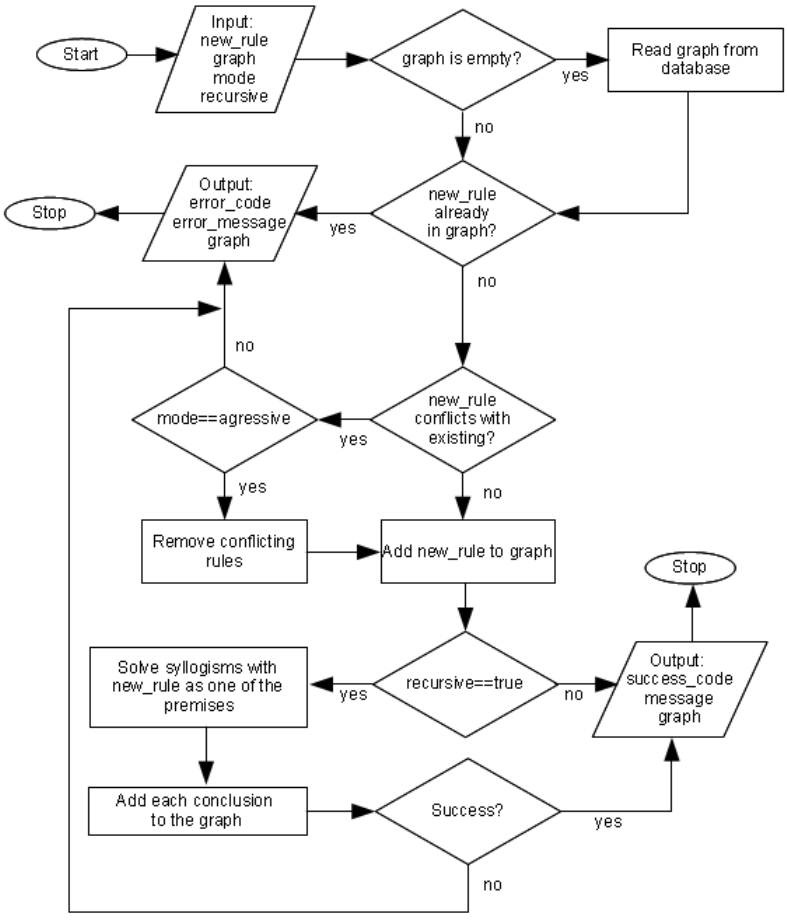

Fig. 4.3: Adding new logic rules

4.3.2 Removing rules

To remove the rule, the engine follows a methodology very similar to the one used while adding a rule, but with certain critical differences.

First, the algorithm checks if the rule actually exists in the database. If this is true, then it is removed. If the algorithm is running in aggressive mode, it will subsequently look for any rules that need to be changed to satisfy the square of opposition, and any rules inferred from solving syllogistic moods. In defensive mode, the maximum of one rule will ever be removed.

The schematic of the algorithm is shown in Fig. 4.4. While superficially very similar, the main difference between the Add and Remove algorithms is the fact that removing a rule will always succeed, provided the rule actually exists in the database in the first place. The number of additional operations that need to be performed in order to normalize the ruleset is lower after removing a rule than after adding one.

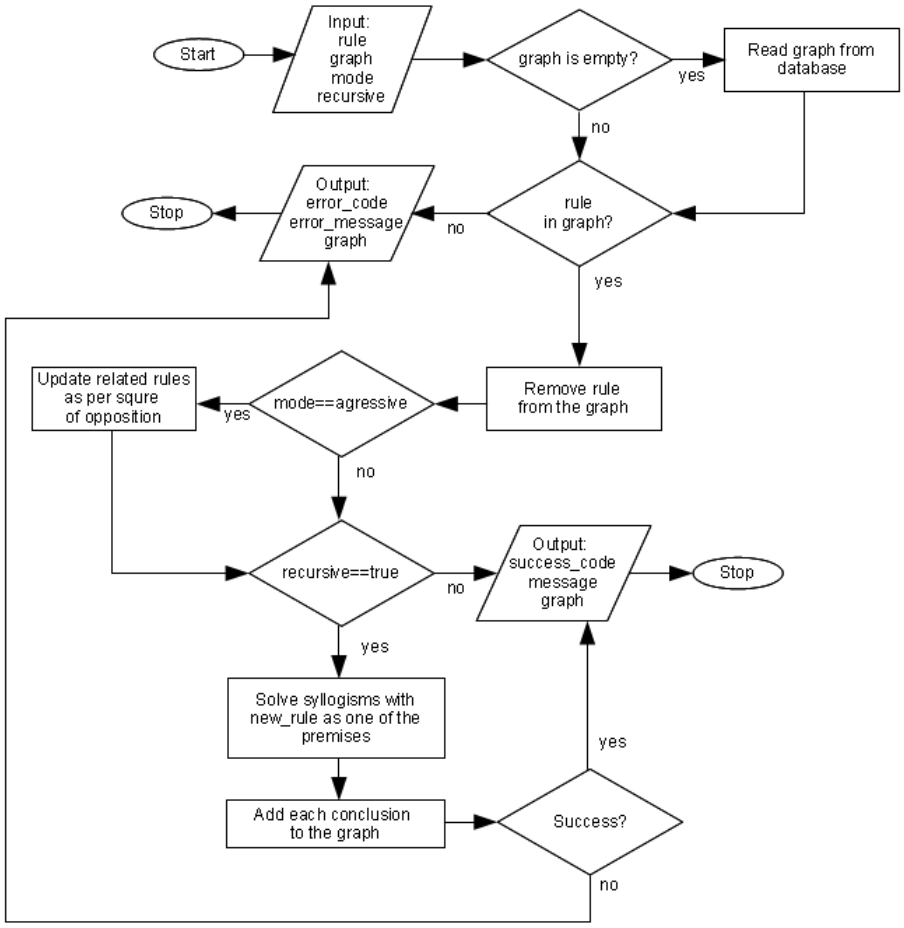

Fig. 4.4: Removing logic rules

4.3.3 Modifying existing rules

To modify an existing rule, the engine first checks if the rule in question exists. If it does, it is deleted using the delete_rule function. The last step is to add the new rule, using the add_rule function. Again, the aggressive and defensive modes apply, acting as explained above. It is important to note that only after the add operation succeeds will the change be

committed to the database.

During both the adding and removal operations, if the aggressive mode is selected, additional rules are modified to satisfy the square of opposition and the rules of conversion. These rules, when put together, create a lot of additional operations for the engine to perform. The dependencies for adding and removing each rule are listed below:

1. If A **a** B is true, then A **i** B is true, A **e** B is false, A **o** B is false, B **i** A is true, and B **e** A is false
2. If A **a** B is false, then A **o** B is true
3. If A **i** B is true, then A **e** B is false, B **i** A is true, and B **e** A is false
4. If A **i** B is false, then A **e** B is true, and A **o** B is true
5. If A **e** B is true, then A **a** B is false, A **i** B is false, A **o** B is true, B **e** A is true, B **o** A is true, B **a** A is false, and B **i** A is false
6. If A **e** B is false, then A **i** B is true, B **i** A is true, and B **e** A is false
7. If A **o** B is true, then A **a** B is false
8. If A **o** B is false, then A **i** B is true, A **e** B is false, B **i** A is true, and B **e** A is false

Written using predicate notation the above rules can be expressed as:

1. A **a** B ⇒ (A **i** B) & ¬(A **e** B) & ¬(A **o** B) & (B **i** A) & ¬(B **e** A)
2. ¬(A **a** B) ⇒ A **o** B
3. A **i** B ⇒ ¬(A **e** B) & (B **i** A) & ¬(B **e** A)
4. ¬(A **i** B) ⇒ (A **e** B) & (A **o** B)
5. A **e** B ⇒ ¬(A **a** B) & ¬(A **i** B) & (A **o** B) & (B **e** A) & (B **o** A) & ¬(B **a** A) & ¬(B **i** A)
6. ¬(A **e** B) ⇒ (A **i** B) & (B **i** A) & ¬(B **e** A)
7. A **o** B ⇒ ¬(A **a** B)
8. ¬(A **o** B) ⇒ (A **i** B) & ¬(A **e** B) & (B **i** A) & ¬(B **e** A)

To perform a recursive update, the engine first searches the logic database for any rules that contain the object of the newly changed rule. This is then followed by testing the validity of all of the 24 syllogistic moods against the results. Any new rules are added to the database using the add_rule algorithm. Once this step is complete, the engine performs a search for all the rules containing the subject of the modified rule. The syllogisms are checked for validity again, but this time with the subject and object exchanged, that is, the object becomes the new subject and vice versa.

5 Testing and verification

Due to the application being based on a series of web documents, the bulk of the testing was done with user acceptance testing. The following functionalities have been tested and confirmed:

1. Upload

1.1. Selecting an image

1.2. Previewing an image

1.3. Generating image thumbnail

1.4. Uploading the image

1.5. Adding tags to the uploaded image

1.6. Suggest tags functionality

1.7. Fuzzy search functionality for Suggest tags

2. Browse

2.1. Basic search

2.2. Advanced search

2.3. Fuzzy search

2.4. Search using multiple tags

2.5. Removing tags from the fuzzy search results

2.6. Adding tags back to the fuzzy search results

2.7. Viewing an image

2.8. Adding tags to the displayed image

2.9. Removing tags from the displayed image

3. Edit search engine rules

3.1. Searching for rules by a single tag

3.2. Searching for rules by multiple tags

3.3. Adding a new rule

3.4. Adding a new rule in aggressive mode

3.5. Adding a new rule in aggressive mode with recursion

3.6. Removing an existing rule

3.7 Removing an existing rule in aggressive mode

3.8 Removing an existing rule in aggressive mode with recursion

3.9 Updating an existing rule

3.10. Updating an existing rule in aggressive mode

3.11. Updating an existing rule in aggressive mode with recursion

5 Testing and verification

In addition, the validity of the operations of the syllogistic engine has been throughly tested in cases 3.3 to 3.11.

6 Conclusion

6.1 Results achieved

As stated in the introduction, the main goal of the thesis was the implementation of syllogistic logic. This was to be achieved by creating a tag and resource management system. As an example use, a simple image gallery was created to be the interface for the syllogistic engine. Both goals were fulfilled by the resultant application.

The secondary goals, as stated in chapter 3, were as follows:

1. Usability – The resultant application needs to be usable by the end user
2. Portability – The application should be able to run on multiple systems
3. Ease of integration – The syllogistic engine needs to be easy to integrate into other applications
4. Scale – The application should be minimalist, with as few dependencies as possible

In the course of the thesis, I have created a functional, if very basic, image gallery application, which allows the user to upload an image from their disk to the server and assign tags to it. In addition, the user can browse for images already present in the server's database, and edit the tags assigned to each one.

Python is an interpreted language and so the engine can potentially run on any operating system that includes a python interpreter. The only additional consideration for the application itself is a web server able to handle and interface with Python's CGI module, as well as JavaScript.

The syllogistic engine is independent of the image gallery itself, and can be used as part of any web application that is able to interface with Python functions. In addition, because the engine itself does not care what type of data it is handling, one can easily build any number of document management applications using it.

The syllogistic engine itself is a very small module that has only a single dependency not provided out of the box by the Python interpreter installation, that being RDFlib. The application additionally requires a JavaScript interpreter and support for the HTML Cancas element for the client machine. Most current web browsers provide both functionalities.

In light of the points outlined above, I conclude that all of the goals of this project have been met, but there is still room for improvement in usability, as the application interface is relatively basic.

My original work includes the creation of the syllogistic engine and the image gallery application, composed of around 1500 lines of code, mostly written in Python, with small parts in JavaScript.

The main difference between my applications and the ones already available is that image galleries do not use a logic engine, however basic, for suggesting tags to the user. Some have very rudimentary parent-child relationships implemented, but they are used very sparingly. None allow for heuristic or fuzzy searching.

6.2 Issues encountered

The myriad of issues encountered during writing of this thesis can be divided into two broad categories: Logical issues and Software issues. Software issues can be further divided into Compatibility issues and Documentation issues. Each of these categories is explained in detail below.

6.2.1 Logical issues

The logical issues I've encountered stem mostly from the nature of Aristotelean syllogism, and the difficulty of expressing the existing rules as a series of if...then statements. Because of the way they are described, the rules of conversion, when taken together with the square of opposition, generate many additional conditional statements. This stems from the fact that the two are closely interconnected, and contain almost exclusively mutually dependent rules. Careful consideration was required to make sure that there were no errors during this process.

The second major source of logical issues were the various logical fallacies that might appear during operations on the logical ruleset. In particular, the fallacy of undistributed middle is difficult to distinguish without context that a machine is hard-pressed to provide. Thankfully, this fallacy is avoidable by strictly regulating the moods that the syllogistic engine processes, making sure we do not end up with a conclusion of too wide a scope, nor do we generate a positive conclusion in the second syllogistic figure.

6.2.2 Software issues

The encountered software issues I have decided to divide into two subcategories. These are the compatibility issues and the documentation issues. Compatibility issues I have encountered included:

1. The latest version of BerkeleyDB has changed the nature of its license. This has wreaked havoc with the logic of Python's database management library, which refused to install. Even using a workaround described in the manual did not fix the issue, with the installer stubbornly refusing to cooperate, ostensibly in an effort to “protect me.” An earlier version of the database was required. However, this poses an issue of having to maintain possibly unsupported software, and so, I have used RDFlib's functionality to provide a fallback option in the form of flat files.
2. The standard Python3 image manipulation library is Pillow. However, Pillow depends on a number of C libraries that are not bundled together with the operating system, and need to be installed separately. This stands in opposition to one of the goals stated earlier, that of the application being lightweight and requiring as few dependencies as possible. In addition, the installer for the library did not automatically install all the required dependencies, which resulted in an never-ending cycle of running the installer, failing, installing a dependency that was listed in the error, and trying to run the installer again. In light of this, I have decided to abandon Pillow in favor of using Canvas and JavaScript.

The amount of work required to find workarounds for compatibility issues I have encountered pales in comparison to those I have decided to collectively call Documentation issues. As a general rule, these stem from incomplete or conflicting documentation, requiring workarounds and experimenting with different approaches. The most severe of these were:

1. JavaScript's parallel execution feature. Meant to prevent a web page from hanging while waiting for an operation to complete, it unfortunately also makes determining a certain order of operations very difficult to achieve. This was critical for creating and saving the thumbnail of an uploaded image using Canvas, as certain operations needed to be completed in a very specific order. In effect all of them had finish to start dependencies.
   I have tried several approaches to this issue, including promises, and using the wait function. Finally the single approach that succeeded was using events that were guaranteed to only happen in very specific circumstances.
2. Saving image data from Canvas to a file. There are a few ways to retrieve image data from a canvas object. Those are ToDataURL, and ToBlob. ToBlob is only supported by latest Firefox versions, and still only experimentally. That meant having to use ToDataURL, which returns a MIME-type object. This object is passed to Python as a base64 encoded string, which needs to be written to a file. However, the Base64 decoder in Python can only work with binary data, not text strings. Converting the string received from JavaScript to binary format required an encoding to be specified. Unfortunately, I was unable to find any concrete information as to which encoding is used for Base64. Finally, based upon a detailed description of the algorithm in question (of which there are many variants), I was able to determine that UTF-8 will be sufficient for my purposes.

6.3 Suggested further developments

While the application resulting from this thesis has basic functionality, there is a large number of improvements and enhancements that can be made, both to the user interface, general functionality, and the syllogistic engine.

In its current version, even though the application can potentially be deployed on the web, it only supports a single user with administrator privileges. While this is sufficient for the purposes of demonstration, much work still needs to be done for the application to be suitable for anything other than off-line use. As such, the next step would be to add multi-user support with different levels of access and credential management.

Additionally, the application can be expanded to handle many different types of documents without changing the underlying syllogistic engine. The application could even be expanded into a fully-fledged discussion forum in the vein of other resource sharing sites.

One of the major fields of improvement is the addition of an automatic image recognition software. Using this functionality, the application would be able to automatically assign tags to the images that the user uploads, with only a verification step on the part of the user. This functionality, when coupled with the syllogism, could provide a powerful tool for image recognition in such fields as areal photography, topography, map-making, and medicine. Currently, much time and energy is spent in cancer research on identifying potential cancerous tissue, most of the work being done by humans that need to carefully pour over each and every photo. Incorporating Aristotelean logic with image recognition software could have a tremendous impact on the effort required for diagnosis.

There is a whole field of potential applications in diagnosis in fact, as the syllogistic engine does not care for the nature of the terms it is manipulating, and as such, could be used to help a doctor diagnose a patient, effectively turning it into an expert system, with just minor modifications, like adding a functionality to ask if a specific statement is true or false, instead of just retrieving a list of related logisically valid terms.

References

[1] History of logic, Encyclopaedia Britannica. Available from: https://www.britannica.com/topic/history-of-logic [20 August 2017]

[2] History of logic, Encyclopaedia Britannica. Available from: https://www.britannica.com/topic/history-of-logic [20 August 2017]

[3] History of Logic, 2017, Wikipedia, wiki. Available at:

[3] Available at: https://en.wikipedia.org/wiki/Syllogism

[4] History of logic, Encyclopaedia Britannica. Available from: https://www.britannica.com/topic/history-of-logic [20 August 2017]

[5] History of Logic, 2017, Wikipedia, wiki. Available at: https://en.wikipedia.org/wiki/History_of_Logic

[6] *The Complete Works of Aristotle: The Revised Oxford Translation*, ed. by Jonathan Barnes, 1984, quoted after Encyclopaedia Britannica

[7] Mikhail Zachnev and Bora Kumova, *Fuzzy-Syllogistic Reasoning With Ontologies*, presented during the 5th World Congress and School on Universal Logic, https://www.researchgate.net/publication/280554834_Fuzzy-Syllogistic_Reasoning_with_Ontologies

[8] Jerzy Wróblewski*, The Judicial Application of Law*, page 231
[9] Jerzy Wróblewski, *The Judicial Application of Law*, page 230
[10] Jerzy Wróblewski, *The Judicial Application of Law*, page 230

Bibliography

1. Edward Nieznański, Logic 3rd edition, C.H. Beck, Warsaw 2011
2. Clay Shirky, The Semantic Web, Syllogism, and Worldview, published online at: http://www.shirky.com/writings/herecomeseverybody/semantic_syllogism.html
3. Jerzy Wróblewski, Judicial Application of Law, 1992, Springer-Science+Business Media, B.V.; ISBN 978-90-481-4113-5, available online at https://books.google.pl/books?id=ynh3BQAAQBAJ
4. Mikhail Zarechnev and Bora Kumova, Fuzzy-Syllogistic Reasoning With Ontologies, conference paper from 5th World Congress and School on Universal Logic, available online at: https://www.researchgate.net/publication/280554834_Fuzzy-Syllogistic_Reasoning_with_Ontologies
5. Zygmunt Ziembiński, Practical logic, Edition XXVI, Wydawnictwo Naukowe PWN, Warszawa 2011
6. Syllogism, Wikipedia, wiki. Available at: https://en.wikipedia.org/wiki/Syllogism
7. History of Logic, Wikipedia, wiki. Available at: https://en.wikipedia.org/wiki/History_of_Logic
8. History of logic, Encyclopaedia Britannica. Available online at: https://www.britannica.com/topic/history-of-logic [20 August 2017]
9. RDFlib documentation, available online at: https://rdflib.readthedocs.io/ [20 August 2017]
10. Python3 documentation, available online at: https://docs.python.org/ [20 August 2017]
11. Stackoverflow programming help forum: https://www.stackoverflow.com [20 August 2017]