IMPLEMENTING TOOLS FOR SUPPORTING LEGAL REASONING

By

David Bareham

A DISSERTATION

Submitted to

The University of Liverpool

in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

24th September 2021

ABSTRACT

IMPLEMENTING TOOLS FOR SUPPORTING LEGAL REASONING

By

David Bareham

This project brings together the research which has been conducted in the University of Liverpool on the ANGELIC project into a single interactive tool. The ANGELIC project has provided a methodology for implementing a system to determine the outcome of legal cases based upon legal domains constructed from precedent cases. This methodology draws from prior work into ADFs (Abstract Dialectical Frameworks), which are sets of nodes with directed links used to represent the factors of legal domains and the relationships between them. Several domains had been individually modelled and evaluated using this methodology, but this had not yet been brought together in a unified tool to be used in a practical setting.

The tool produced enables the user to query a legal case for one of the domains modelled by answering a number of questions instantiating the case as a list of factors. These cases are then evaluated according to the relevant ADF graph using a depth-first search providing a case outcome. The result of the case is predicted, and a list of reasons for why that outcome was reached is generated, alongside a visualisation highlighting the relevant factors for the decision. The tool also allows the user to create ADFs for new legal domains or edit existing one.

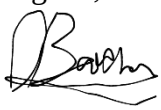
The project was successful when evaluated in regard to accuracy, explainability and usability. The tool was able to replicate the results of the previous implementation within Prolog, achieving the same accuracy, provide a transparent outcome with clearly stated reasons and visualisation for explainability as well as aiding usability by providing a comprehensive user manual and intuitive user interface.

DECLARATION

I hereby certify that this dissertation constitutes my own product, that where the language of others is set forth, quotation marks so indicate, and that appropriate credit is given where I have used the language, ideas, expressions, or writings of another.

I declare that the dissertation describes original work that has not previously been presented for the award of any other degree of any institution.

Signed,



ACKNOWLEDGEMENTS

I would like to thank my supervisor Professor Katie Atkinson whose guidance and advice has been crucial in being able to complete this project. Thanks also to my fiancée for her endless support.

TABLE OF CONTENTS

LIST OF FIGURES

Section 1 Introduction

The ANGELIC project (ADFs [Abstract Dialectical Frameworks] for kNowledGe Encapsulation of Legal Information from Cases) developed a methodology for determining the outcome of legal cases based upon legal domains constructed from precedent cases and other sources. My project aims to bring the research together into a unified tool and meet the following key objectives:

- Creating a tool providing outcomes for cases in four legal domains for both pre-defined and user-defined cases.
- Allowing the user to edit existing ADFs and create new ones from scratch.
- Evaluating the tool's success against the level of accuracy it achieves, its usability and explainability regarding its outcomes.

1.1 Legal Domains

The four domains the tool will capture are briefly explained here.

- **Wild Animals:** These are cases in which the plaintiff's chasing of wild animals was interrupted by the defendant [5].
- **US Automobile Exception to The Fourth Amendment:** Centres on cases which involve searching a moving automobile without a warrant in exception to the Fourth Amendment which protects persons against unreasonable searches and seizures [5].
- **Trade Secrets:** This domain concerns information, which is unknown by competitors, providing a company an economic advantage that is subject to reasonable efforts to maintain its secrecy [5].

- **Noise-Induced Hearing Loss (NIHL):** Within this domain are cases in which it is alleged the claimant's hearing loss is due to a previous employer's negligence [7]

1.2 Related Work

Within the literature concerned with defining methodologies for designing systems to determine the outcome of legal cases, I found there to be two distinct approaches used which, broadly, can be divided into data-driven methods and knowledge representation methods.

A representative example of the data-driven approach can be found in [1] which leverages advances in Natural Language Processing (NLP) and Machine Learning (ML) to predict whether particular articles of the European Convention on Human Rights (ECHR) have been violated given textual evidence from the case in relation to the relevant pieces of law. In this paper they derived N-grams or word clusters from the text as their features for use in a regularised linear Support Vector Machine (SVM) achieving an average accuracy of 79%. Another study used SVM [9] to achieve a slightly lower accuracy of 75% in the ECHR domain. Other data-driven approaches within the field have used a variety of machine-learning techniques, such as Convolutional Neural Networks, to predict whether a court of appeal would affirm or reject the lower court's decision, achieving an accuracy of 79% [13] or [10] which achieved an accuracy of 70.2% on US Supreme Court judgements from 1816-2015 using feature engineering and a random forest classifier.

Despite these approaches all achieving an accuracy that would be promising in other domains, [2] emphasises that in the legal domain it would be unacceptable for over 20% of cases to be decided inaccurately. If this technology was going to be adopted within the legal sector, the inaccuracies need to be greatly reduced. Perhaps the most crucial critique regarding the approach in [1] and [9] is the lack of explainability, as affirmed within [2], that

no explanation is provided in the study other than a list of the most frequent words in order of their weights in the SVM.

This need for explainability and transparency is a key pillar of legal decision making, which is affirmed as a positive right within the European Union's Charter of Fundamental Rights, which explicitly states, "[…] the obligation of the administration to give reasons for its decisions." [3]. Considering this obligation, 'black box' ML methods such as deep learning appear *prima facie* unsuitable for the task of predicting legal decisions, if such techniques are to be applied within the sector, as methods such as Deep Neural Networks, for example, typically feature a degree of complexity so great that the specific weights for individual features cannot be determined for all predictions generated [4].

In light of these issues the ANGELIC methodology seeks to incorporate advances in knowledge representation to address the explainability gap and provide transparency in the AI decision-making process. This approach follows from a larger body of work focusing on legal case-based reasoning such as HYPO [12] or CATO [11] which use factors based upon the absence or presence of relevant facts within a case. Work such as CATO is theoretically underpinned by the fact that the law does not consist merely of a set of rules but also of a body of previous cases and legal decisions [14]. A new case must be decided within the context of previous decisions, which form legal precedents, that can be used as the basis of a legal reasoning system.

Following on from this idea [5] sets out the foundation for the approach we will focus on. This methodology utilises an abstraction of Dung's [6] abstract argumentation frameworks called Abstract Dialectical Frameworks (ADF). An ADF can be defined, as per [5], as a set of nodes with directed links between them which contain acceptance conditions local to each node. These nodes, in a similar manner to CATO, take the form of a hierarchical list of

factors with the base-level factors as the children of more abstract factors [5]. The directed links take the form of attacking or supporting links. If a child node has an attacking link with its parent, then that factor can be seen as counting against the acceptance of the parent, whilst a supporting link counts towards that parent being accepted. Thus, these children form the basis of the acceptance conditions for the parent nodes to be considered absent or present within the ADF. Acceptance conditions within an ADF are structured with nodes as the operators and logical conditions such as ‘and’, ’or’ and ‘not’ as the operands.

To better understand the way in which an ADF operates, let us use an example of a node and its acceptance conditions within Wild Animals [5]. Figure 1 shows the AntiSocial node whose children are DMotive (Defendant Motive), Nuisance and Impolite. As stated earlier, the acceptance condition of a parent node is formed from the children nodes. In this case the acceptance condition is ‘(Nuisance or Impolite) and (not DMotive)’. This condition must be determined after legal analysis considering the precedent cases and statutes governing the domain. If the acceptance condition is satisfied, then the node to which it belongs is accepted as a factor within the case and this drives the decision-making process.

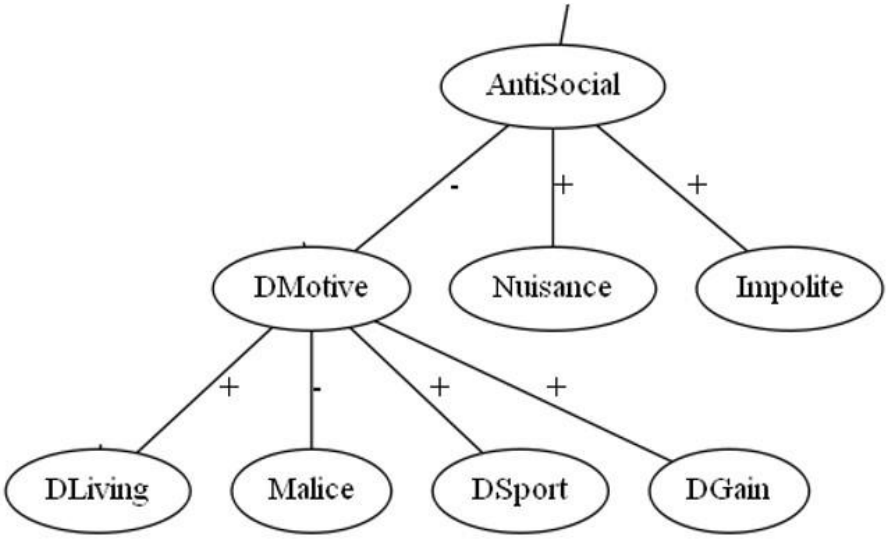


Figure 1- Part of Wild Animals as generated in the visualisation function described in §8.2.

The decision-making process begins by instantiating a list of base-level factors corresponding to facts in that case. These factors need to be chosen by a person with some familiarity with the case and this may be done through the use of 'Yes' or 'No' questions to establish if each base-level factor is absent or present within the case. Once these factors are assembled, the ADF proceeds by ascending through the acceptance conditions of the nodes until an outcome is reached. Due to its inherent structure as a set of hierarchical factors it can output the factors which directly caused the achieved outcome. This output provides the transparency and explainability in an AI-driven process which the data-driven methods do not achieve due to the complexity of their 'black box' processes.

Another comparison between the two approaches outlined can be made with respect to [2] who were able to achieve a 100% accuracy in a set of 10 test cases taken from the same domain, ECHR, as [1]. Despite the small number of test cases, it is a promising demonstration of the power of the knowledge-representation approach, compared to the data-driven ML approach, in offering the explainability required.

Section 2 Ethical Use of Data

The data used in this project is primarily from the public domain. The case data for Wild Animals, Trade Secrets and Automobile Exception was gathered from freely available public court records. Given the public nature of the data, there are no ethical issues with these domains. This data and the accompanying ADFs have been obtained from [5] which is one of the primary papers underlying the implementation of this tool. However, NIHL [7], was developed with an industry partner of the University of Liverpool. Due to non-disclosure agreements, the ADF for this domain is not able to be published within this paper. The confidentiality of the case data for the domain also meant I was unable to obtain any real cases to test with my implementation so instead relied on a few synthetic cases which I constructed.

Section 3 Design

This section will provide a statement of the project requirements and outline the full design of the system's frontend and backend (see Appendix A for the original design documentation).

3.1 Design Outline

The system design aims to realise several key objectives and implement the required features necessary to meet these. The core features which the design aims to realise are:

- **Query an existing ADF**: The user will be able to query an existing ADF by answering a number of questions corresponding to each of the base-level factors in the ADF. After the base-level factors have been generated, the tool should generate an outcome, and show the different factors which went into making the decision.
- **Editing an existing ADF**: The user will be able to edit and update existing ADFs in light of new legal precedents.
- **Creating an ADF**: This tool will allow the user to create an ADF for a new legal domain defining the factors, relationships, acceptance conditions, acceptance/rejection statements and questions instantiating the base-level factors.
- **User Interface**: This implementation will provide a clear graphical user-interface which will allow the user to easily use any of the aforementioned features.
- **Saving/Importing**: This feature will save the ADF into a '.xlsx' file which can then be read back into the tool instantiating the saved ADF.
- **Reporting:** This function will provide some basic information about each ADF, such as how many nodes there are and the ability to generate visualisations for each case outcome.

3.2 Methodology and Technology

Whilst the initial implementations from prior research were achieved in Prolog [5], due to the declarative nature of the acceptance conditions, I will use Python 3 because I believe it offers the flexibility and robustness necessary in developing the entire tool from start to finish. For the frontend I have used Tkinter since it is well documented and offers a sufficient degree of customisation.

The Agile methodology has been used throughout the software development process to develop functional systems quickly, incrementally iterating and expanding the features available [8]. This allowed me to seek feedback from my supervisor regularly and easily implement any proposed changes for the next iteration, ensuring that the project did not stray from the desired outcome.

3.3 Backend Design

The backend design is centred around the core process of creating and generating an ADF within the system. In designing this, I have followed the Object-Oriented Design paradigm. The three primary classes in the backend are:

- Node() - representing the individual factors as nodes with their associated acceptance conditions etc.
- MultiNode() - inheriting from the Node class by adding the ability to create nodes instantiated by multiple choice questions. This class was added later on in the development as per the explanation in §3.3.1.
- ADF() - modelling the ADF in terms of the individual nodes' relationships with one another and providing core functionalities such as tree traversal and saving.

The structure of these classes is displayed in a UML class diagram in Figure 2.

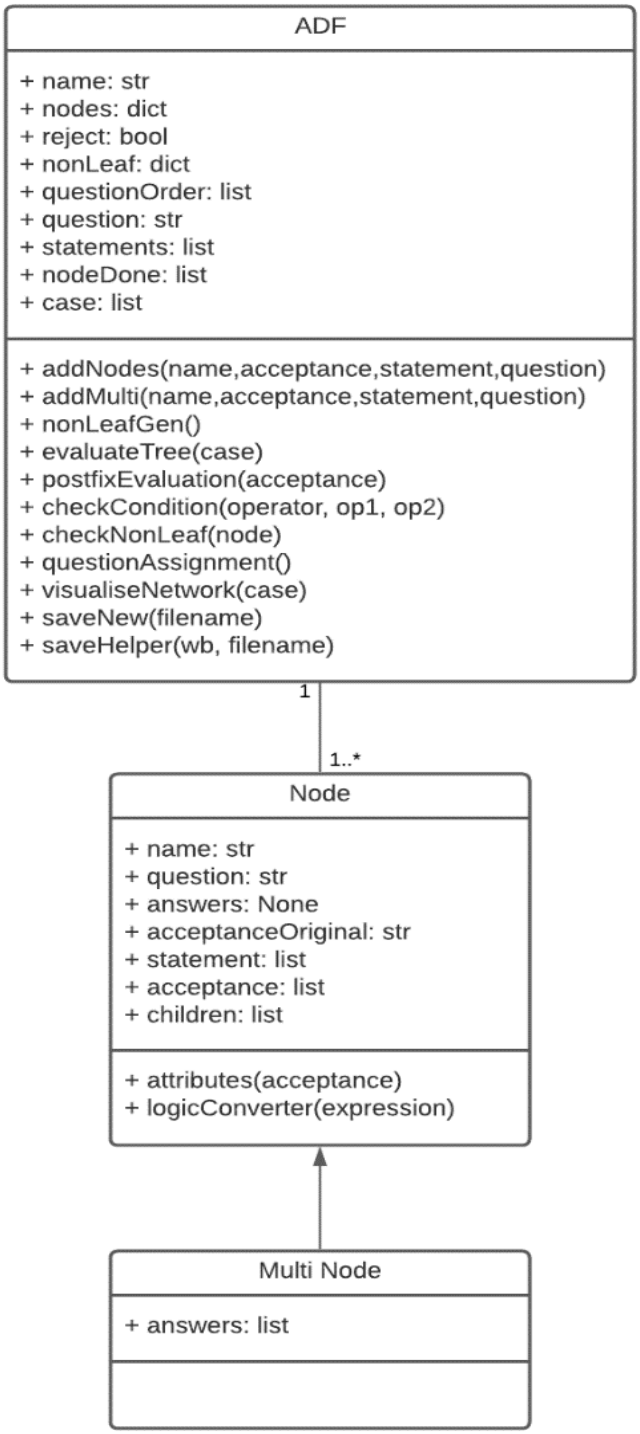


Figure 2 - UML Class Diagram of the Backend

3.3.1 Node() Class

The class's parameters will take in the factor name, acceptance conditions and their corresponding statements. This class can be used to initialise non-leaf factors or base-level factors. The main difference is that a base-level factor needs an associated question to be presented to the user when determining a factor's presence as part of the case, whereas the absence or presence of non-leaf nodes is determined solely by their acceptance conditions.

A key part of the Node() class is the conversion of the acceptance condition from infix to postfix notation. This is due to the challenge of parsing logical conditions which may contain parenthesis such as in "NOT (NotCaught) OR (Vermin and HotPursuit)" [1]. The advantage of postfix notation is that it enables logical statements to be expressed without parenthesis whilst maintaining operator precedence, e.g., 'InfoKnown **or** InfoAvailiableElsewhere' → 'InfoKnown InfoAvailiableElsewhere **or**'. Postfix notation is more computationally efficient and easier to parse in determining whether the relevant condition has been satisfied or not (see Appendix A for the pseudocode).

3.3.2 MultiNode() Class

While the Node() class is suitable for the majority of cases, there are a couple of domains, such as Automobile Exception and NIHL, for which it proved insubstantial during the implementation due to the nature of some of their base-level factors. Most base-level factors within an ADF have a Boolean nature and are either true or false, but others are too complex to be captured with such a binary, for instance the PermittedDuration node from Automobile Exception. This node concerns how long an automobile was permitted to stay in a stationary location. This can be answered with 'short stay', 'long stay' or 'overnight'. To implement this in the original Node() class would require treating PermittedDuration as a non-leaf factor with the answers such as 'short stay' as its children and subsequently base-level factors.

Nevertheless, that design would require a question for each of the base-level factors, adding unneeded repetition to the question-base.

To avoid this, I introduced the MultiNode() class: a child of the Node() class inheriting the same methods but differing in its initialisation. MultiNode() has an attribute named ‘answers’, which is composed of the names of the children nodes, that stores the multiple-choice answers to the question asked. Thus, base-level factors are instantiated by a single sub-factor, such as ‘short stay’, or even a combination of sub-factors. Consequently, this class acts as a blend between a leaf node and a non-leaf node, requiring a question as part of its initialisation whilst also needing an acceptance condition.

3.3.3 ADF() Class Design

This class will take a name as its only compulsory parameter, the other attributes will be initialised as empty lists or dictionaries which are then modified with the introduction of Node() instances into the ADF. A dictionary has been chosen as the data type to store the nodes constituting the ADF with the node name as the key and the node object instance as the value. This allows the ADF() to be flexible in adding new nodes, deleting old nodes, and editing existing nodes as specific node instances can be found quickly using their key.

The core algorithm which drives the functionality of this class is evaluateTree(). This method only takes the case as an input and provides the prediction of the outcome as an output. It begins by generating a dictionary of non-leaf nodes by iterating through each node in the ADF and determining whether it has children or not, since a non-leaf node must by definition, always have children.

1. For each non-leaf node which has not already been evaluated, the algorithm checks whether its children have been evaluated before or whether the children are all base-

level factors. In the first iteration, for instance, this would result in the algorithm initially evaluating a node where all its children are base-level factors. This creates the effect of ascending the ADF from the base-level factors, through the intermediary factors and up to the final decision node, which reaches an outcome.

2. Next, the algorithm tests the acceptance conditions of the chosen node against the base-level factors originally supplied in the case and any additional intermediary factors which may have been added previously. If the acceptance condition is satisfied, it adds that node's factor to the factors of the case and the corresponding statement accompanying the condition to a list of statements, which will later be output to the user. If the acceptance condition is not satisfied, then the corresponding rejection statement is appended to the statements. In either case the node is removed from the list of non-leaf factors to be evaluated.

3. It returns to step 2 until there are no nodes left which have not been evaluated, aside from the decision node in which case it evaluates that node. After the decision has been determined, the algorithm ceases, and the outcome is shown to the user.

3.4 Frontend Design

The frontend design is focused around providing a clear user experience. Figure 3 shows the original wireframe of the user interface from the design document. Whilst there have been changes made to its design since this wireframe was created, it still shows the core pathways

the user can take through the software. Those pathways are domain creation, editing and querying.

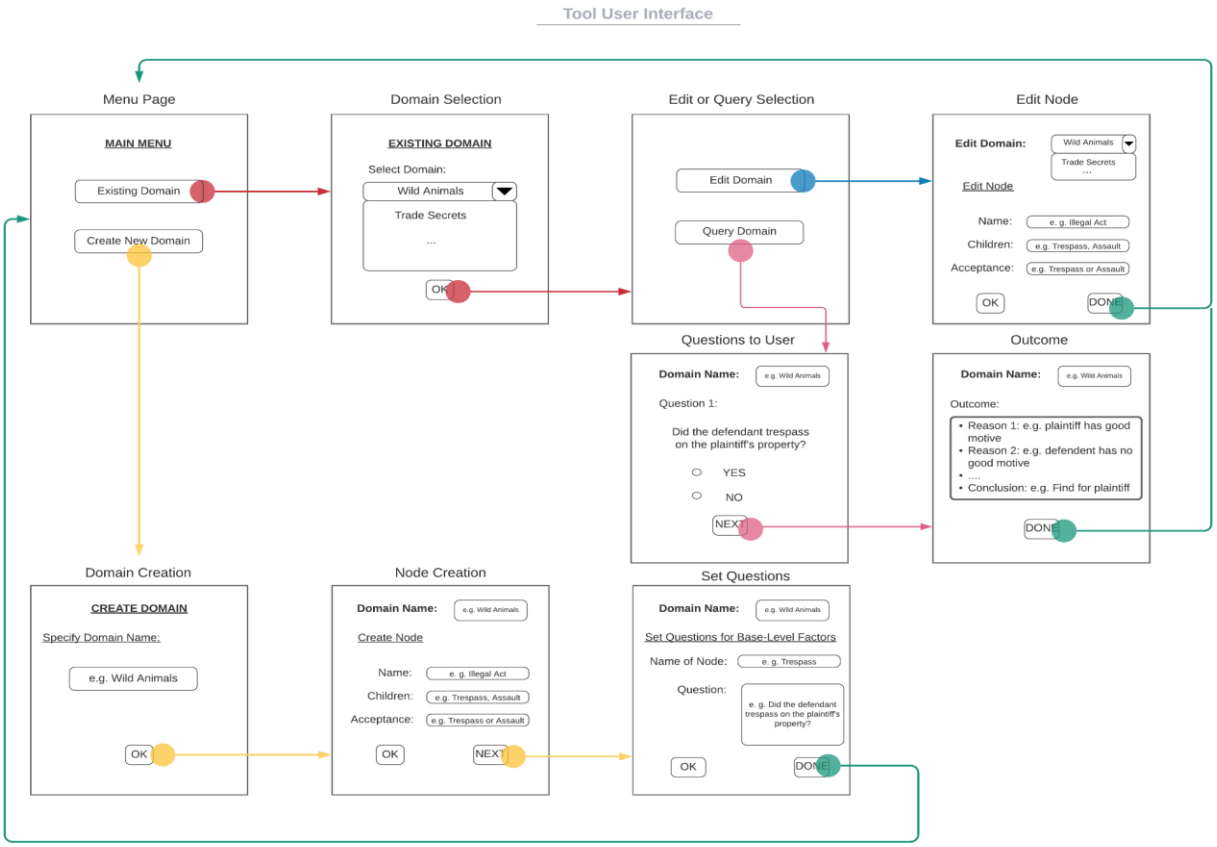


Figure 3 - A mock-up of the user interface.

Section 4 Realisation

This section aims to explain how the backend and frontend of the software were realised within Python (see Appendix B for the full code listing).

4.1 From Prolog to Python

The original realisations of the ANGELIC project for Wild Animals, Trade Secrets and Automobile Exception were all constructed within Prolog [5]. Prolog is a language which was created with the goal of using logic to represent knowledge by using a subset of predicate logic [15]. This lent itself to the structure of the ADF, which consists of acceptance conditions expressed in predicate logic. The authors of [5] explicitly write that "The Prolog program was formed by ascending the ADF, rewriting the acceptance conditions as groups of Prolog clauses to determine the acceptability of each node in terms of its children." While this approach was beneficial for ensuring the methodology worked, if I was to transfer this into a fully featured tool, with the facility to generate an ADF without needing to explicitly hard-code it, I needed a language which would allow declarative expression whilst also supporting other paradigms such as object-oriented programming. Python has been able to provide the flexibility required in creating the various features of the software, but in attempting to capture the declarative nature of Prolog there have been some challenges.

4.1.1 How Prolog works for an ADF

To understand how Prolog's methods were implemented into Python it is important to understand how the ADFs were originally constructed within Prolog. The following example aims to illustrate this.

Let us focus upon a specific node from Trade Secrets called ConfidentialityAgreement (f121 in the Prolog). The acceptance condition provided in [5] for this node is – 'Accept

ConfidentialityAgreement [f121] if AgreedNotToDisclose [f4] and (not WaiverOfConfidentiality) [f23] otherwise reject ConfidentialityAgreement'. In Prolog this is first interpreted into a rule. Rules constitute a consequent, followed by the symbol ':-' (read as 'if') and the antecedent. In our case the consequent is ConfidentialityAgreement, and the antecedent is its acceptance condition. The acceptance condition can be represented as a series of facts which are statements describing object properties or object relations [15]. As shown in Figure 4, the antecedent is represented by 'member(f4, Factors), not(member(f23,Factors))'. This can be read as checking whether AgreedNotToDisclose is a member of the list of Factors and checking that WaiverOfConfidentiality is not a member of the list. The conjunctive nature of this clause is captured in virtue of the conditions being on the same line separated with commas. To capture disjunctive conditions, one would simply have a separate rule for each clause in the order they wished the statements to be considered.

When a rule's antecedent is evaluated as 'True' the Prolog program performs any actions associated with this. In Figure 4 it will print the statement 'there was a confidentiality agreement' to the user and will add ConfidentialityAgreement to the list of factors (this is done with '[f121|Factors]'). Whereas, if the condition is rejected the program will show 'there was no confidentiality agreement' and will not add the factor to the list.

```
getf121(C,Factors):-member(f4,Factors), not(member(f23,Factors)),
                                write([there,was,a,confidentiality,agreement]),nl,
                                getf115(C,[f121|Factors]).
getf121(C,Factors):-write([there,was,no,confidentiality,agreement]),nl,
                                getf115(C,Factors).
```

Figure 4 - ConfidentialityAgreement node from the Prolog [5].

Prolog is a declarative language whose control structure needs to be explicitly stated. In our example, whether the node is accepted or not, the program will move on to evaluating

NoticeOfConfidentiality (f115). The manner in which it conducts this search is called a Depth-First Search, which is a strategy in which “The search goes down the graph until it reaches a node without successor. It then backtracks from the bottom to the last node that has successors.”[15]

4.1.2 ADF Implementation in Python

The following sections will detail the challenges faced when converting the Prolog code to Python and how I tried to solve these issues.

4.1.2.1 ADF

When implementing the ADF within Python the first challenge was how to model the logical operators. Whilst Python does natively support logical operators, using these would require me to have hard-coded each acceptance condition individually which was exactly what I was trying to avoid in building a flexible ADF creation engine. To facilitate the future construction of ADFs I needed the acceptance conditions to be able to be input by a user with the nodes as the operands accompanied by logical operators.

As stated within §3.3.1, I chose to translate each user inputted acceptance condition into postfix notation to allow me to parse them with greater ease which is evaluated as follows within postfixEvaluation(), see Figure 5:

- The acceptance condition’s tokens are iterated through, adding them to the stack.
- If the token is a logical operator, it pops the last one or two operands from the stack to evaluate whether the condition is true or false.
- checkCondition() checks whether the operands are present in the case or not, depending on the operator, or if the tokens are Boolean values of True or False. This is necessary because the output of the function is always True or False (and this

output is added to the stack) as frequently an operand corresponding to a node will be evaluated alongside a token which equals True or False.

An example will help to illustrate how this works:

If we have the base level factor A in our case and our infix acceptance condition is ‘(A or B) and not C’, this will become ‘A B or C not and’ in postfix form. The algorithm would first evaluate the condition ‘A or B’ and since A is within our case this would return True to the stack meaning our stack is now ‘True C not and’. Next ‘not C’ would be evaluated and since only A is within our case this would also return ‘True’. Our stack is now ‘True True and’ meaning the final condition to be evaluated is ‘True and True’. A truth table will tell us that the conjunction of two truth values is ‘True’ in which case the acceptance condition is accepted and the algorithm proceeds.

```
if operator == "or":
    if op1 in self.case or op2 in self.case or op1 == True or op2 == True:
        return True
    else:
        return False

elif operator == "and":
    if op1 == True or op1 in self.case:
        if op2 in self.case or op2 == True:
            return True
        else:
            return False
    elif op2 == True or op2 in self.case:
        if op1 in self.case or op1 == True:
            return True
        else:
            return False
    else:
        return False

elif operator == "not":
    if op1 == True:
        return False
    if op1 == False:
        return True
    elif op1 not in self.case:
        return True
    else:
        return False
```

Figure 5 - Logical operator evaluation code

This structure is able to mimic Prolog's use of logical operators in returning the correct outcome for even the most complex acceptance conditions. However, this code only succeeds in evaluating the acceptance conditions but not in capturing the control procedure found within the Prolog. Figure 6 describes within its comments the evaluateTree() function. This function mimics Prolog's depth-first search by only evaluating nodes who have no children which are not base-level factors, or nodes whose children are all base-level factors or factors which have already been evaluated.

```
#keep track of print statements
self.statements = []
#list of non-leaf nodes which have been evaluated
self.nodeDone = []
self.case = case
#generates the non-leaf nodes
self.nonLeafGen()
#while there are nonLeaf nodes which have not been evaluated, evaluate a node in this list in ascending order
while self.nonLeaf != {}:
    for name,node in zip(self.nonLeaf,self.nonLeaf.values()):
        #checks if the node's children are non-leaf nodes
        if name == 'Decide' and len(self.nonLeaf) != 1:
            pass
        elif self.checkNonLeaf(node):
            #adds to list of evaluated nodes
            self.nodeDone.append(name)
            #checks candidate node's acceptance conditions
            if self.evaluateNode(node):
                #enables rejection clauses - handy for automobile
                if self.reject != True:
                    #adds factor to case if present
                    self.case.append(name)
                #deletes node from nonLeaf nodes
                self.nonLeaf.pop(name)
                self.statements.append(node.statement[self.counter])
                self.reject = False
                break
            #if node's acceptance conditions are false
            else:
                #deletes node from nonLeaf nodes but doesn't add to case
                self.nonLeaf.pop(name)
                #the last statement is always the rejection statemenr
                self.statements.append(node.statement[-1])
                self.reject = False
                break

return self.statements
```

Figure 6 - The evaluateTree() method.

4.1.2.2 The Reject Condition

A group of nodes from Automobile Exception caused me to rethink how I implemented the acceptance conditions. A limit of my implementation was that it only allowed a node to be accepted if a condition was met but did not allow a condition to be rejected if it was met. In modelling some factors we may want to reject them outright if one of their children is present.

In Figure 7 we can see the Prolog implementation of the node PrivateContentsCarriage (af131) from Automobile Exception with the base-level factors ProtectionType (bf312) and GoodsCarried (bf311). This node's first acceptance condition is: if ProtectionType and

GoodsCarried then PrivateContentsCarriage is accepted, and it prints “private contents”. This is trivial to model in my implementation, the difficulty comes with the second acceptance condition: if GoodsCarried then reject and do not add PrivateContentsCarriage to the list of factors.

If GoodsCarried was present and ProtectionType was not, in my implementation only modelling the first condition, it would show the default statement of ‘contents are not considered private’. While this does not affect the case outcome, this information is a contradiction, as if GoodsCarried was accepted, it would display the message ‘private goods’ to the user, so how could there be ‘private goods’ yet ‘contents are not considered private’?

I addressed this issue by:

- Designing a rejection condition using the keyword ‘reject’ in front of an “acceptance condition”. If present, the software will not add the node to the list of factors but will print its corresponding statement as if it had been “accepted”.
- ‘reject’ causing a flag called ‘self.reject’ to be set to True if the keyword is present when evaluating a condition.
- The corresponding statement of the condition being printed if the flag is set to True and the node not being added to the list of factors.

After implementing this feature, I could now model the second acceptance condition as ‘reject GoodsCarried’ with the corresponding statement ‘private contents but not protected’. This nullifies the contradiction since the output simply expands upon GoodsCarried statement of ‘private contents’, to qualify them as unprotected rather than contradicting it.

```
%AF1-3-1    PrivateContentsCarriage
%GoodsCarried and protected
getPrivateContentsCarriage(C,Facts,Factors):-member(bf312,Factors),member(bf311,Factors),!,
                write([private, contents]),nl,
                                getConnectedServices(C,Facts,[af131|Factors]).
%GoodsCarried and not protected
getPrivateContentsCarriage(C,Facts,Factors):-member(bf311,Factors),!,
                                write([private, contents , but, not, protected]),nl,
                                getConnectedServices(C,Facts,Factors).
%default-not private
getPrivateContentsCarriage(C,Facts,Factors):-!,write([default,contents, are, not, considered, private]),nl,
                                getConnectedServices(C,Facts,Factors).
```

Figure 7 - PrivateContentsCarriage node from Automobile Exception in Prolog [5].

4.2 Software Produced

The following section will provide an overview as to what some of the key functionalities achieved look like.

4.2.1 Frontend

The Frontend was implemented in the Python library Tkinter and is built using a series of classes such as CreateDomain(), which creates a page in which the user can specify the name of the domain they wish to create. These classes inherit from Tk.Frame which acts as a container for a group of widgets. To create each page the required inputs are parent, controller, and info. Parent initialises the Tk.Frame class for the page and controller allows the page to access the methods of the controller class providing the core functionality in the UI. Info is explained in §4.2.1.1.

The controller class in my implementation, shown in Figure 8 as the UI() class, provides methods such as frameCreation() and show_frame() which allow the display and creation of new pages. This implementation was not straightforward within Tkinter since there is no direct functionality for changing pages. Instead, I was required to implement a show_frame() method utilising the Tkinter function .tkraise, which allows one page to be lifted on top of another, creating the effect of changing between pages. To keep track of these pages I used a

dictionary with the key as the class name for the page and the value as the page object itself. This data structure was particularly useful in helping to escape potential memory issues which may arise from stacking many pages on top of each other. Each time, for example, an instance of EditDomain() is created it overwrites the previous instance in the dictionary. As the old instance is no longer referenced within the program, Python's automatic garbage collection removes the original object.

```
class UI(tix.Tk):

    def __init__(self,*args,**kwargs):

        tix.Tk.__init__(self,*args,**kwargs)
        tix.Tk.title(self,'ADF Tool')
        tix.Tk.geometry(self,"750x500")

        menubar = tk.Menu(self)
        filemenu = tk.Menu(menubar,tearoff=0)
        self.config(menu=menubar)

        filemenu.add_command(label="Menu",command=lambda:self.menu())
        filemenu.add_command(label="Exit", command=lambda:self.quit())
        menubar.add_cascade(label="File", menu=filemenu)
        menubar.add_command(label="Help", command=lambda:self.help())

        self.container = tk.Frame(self)
        self.container.pack(side="top",fill="both",expand=True)
        self.container.grid_rowconfigure(0,weight=1)
        self.container.grid_columnconfigure(0,weight=1)

        self.frames = {}
        frame = Welcome
        self.info = Information()
        self.frameCreation(frame,self.info)
        self.show_frame(frame)

        #styles
        s = ttk.Style()
        s.configure('my.TButton', font=('Helvetica', 15))

    def show_frame(self, page):···

    def frameCreation(self, F, info):···

    def menu(self):···
```

Figure 8 - The UI() class

4.2.1.1 Linking the frontend to the backend

Linking the backend to the frontend involved two main steps:

1. Importing the backend in MainClasses.py and the ADF files into UI.py.
2. Creating a class named Information() which holds all the key information needed to be shared between classes such as the case factors, the ADF data etc.

4.2.1.2 Frontend Example

The design of the frontend is minimalist. Figure 9 shows a screenshot from the user-interface which helps to demonstrate the design aesthetic I chose to aim toward. Each screen shares the same light blue background with grey buttons and black text. Red is used for the question marks to demarcate their importance as tool tip help pop-ups. In Figure 9 you can see a yellow box which informs the user that a domain name should have no spaces between the words (see Appendix C for more UI screenshots).

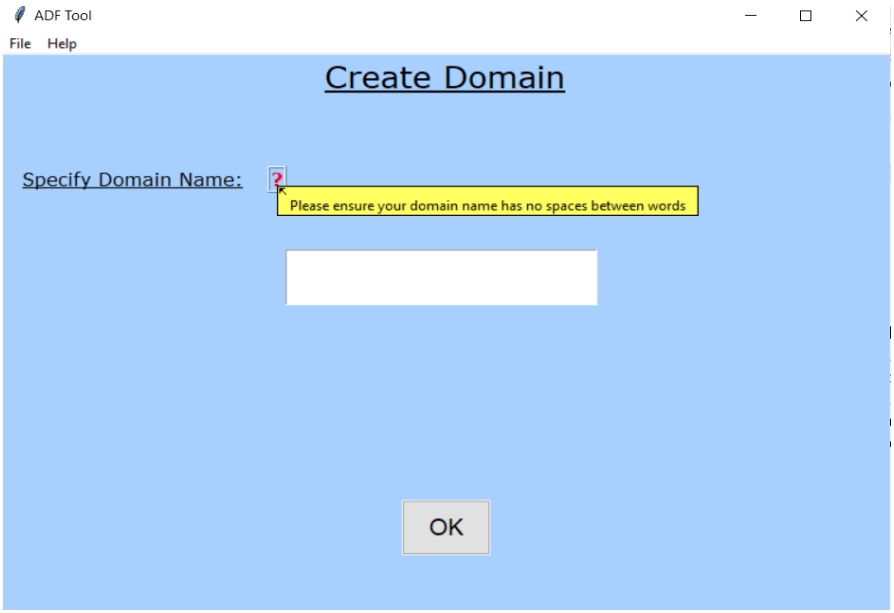


Figure 9 - Create domain screen

4.2.2 Querying an ADF

The most central feature of the software is to allow the user to reach an outcome in a case within one of the four legal domains encapsulated by the ADFs. This proceeds in the following manner:

- The user chooses one of the predefined test cases for each domain, or they can query their own case. For cases which are not pre-defined the software will present questions one at a time to the user in a non-randomised order.
- The questions establish which base-level factors are present using checkboxes to indicate ‘Yes’ or ‘No’.
- Sometimes, as discussed in §1.2, binary options are not suitable so the software also allows multiple-choice questions which are displayed to the user as square tick-boxes from which many or no answers can be selected.
- The ADF is then queried to determine the outcome of the case, providing an outcome screen similar to the one in Figure 10, of the case Arco Industries v Chemcast in Trade Secrets. This screen provides the user with a clear list of the reasons which enabled the decision to be made.
- This continues until the final decision node, which convention within this software dictates are called ‘Decide’, determines the final outcome based on its acceptance condition.

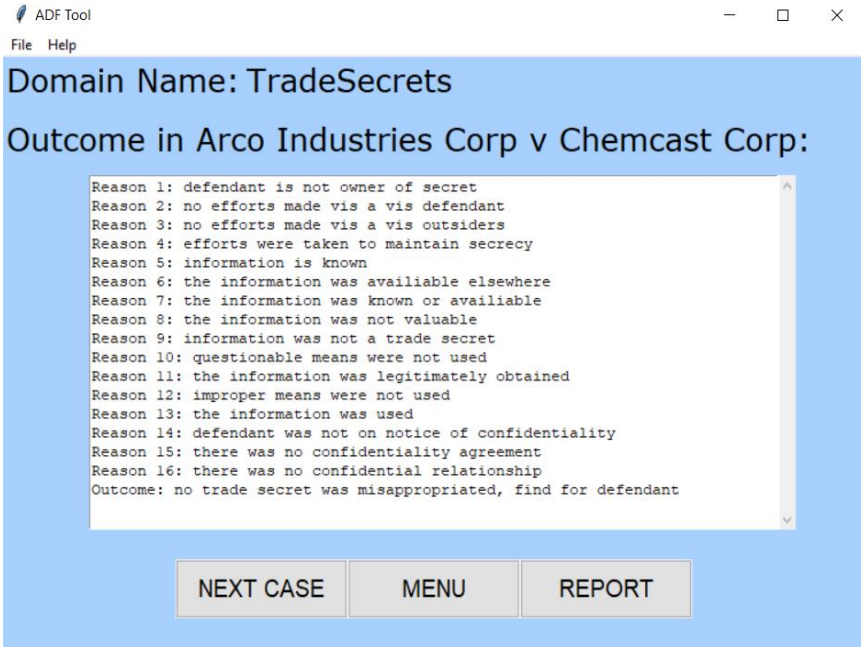


Figure 10 - Outcome Screen

4.2.3 Creating an ADF

For the user this process begins after initialising the ADF() class by setting a name for the ADF and then adding either a ‘True/False Node’, corresponding to the Node() class, or a ‘Multiple Choice Node’, corresponding to the MultiChoice() class. In most circumstances the user will use the ‘True/False Node’ but in some cases, the ‘Multiple Choice Node’ will need to be used, as explained within §3.3.2.

After all the nodes within the ADF have been added, the software prompts the user to input the questions which will be asked (when the ADF is queried) to establish whether the base-level factors are absent or present. The software automatically determines what is and is not a base-level factor, prompting the user only to set questions for those it has identified. Following the questions having been set you can choose the order you wish them to be asked in. This is useful since it allows questions on related topics to be asked in logical groups,

making it easier for the user to follow a logical process and provide answers which are correct for their case.

4.2.4 Editing an ADF

This feature allows the ADFs, whether predefined or user-created, to be modified. The ability to modify ADFs is important for a number of reasons. An ADF might require updating due to a new ruling overwriting previous precedent, an acceptance condition not adequately modelling its associated factor, or new domain knowledge requiring additional nodes. Due to the ADFs' structure, storing the nodes in a dictionary, they are very flexible to be adapted within the UI. For instance, nodes can be created or deleted, question orders changed, acceptance conditions and corresponding statements added or modified, and questions set for new or existing base-level factors.

One of the key functionalities of this feature is the save function. This allows the user to save an ADF which has been created or edited within the tool. It writes the key data in the ADF to a '.xlsx' file, a spreadsheet which can later be imported back into the tool. The tool will then read the data and instantiate an ADF from it. In order for this process to be consistent, there are various rules in the backend which govern how the save file of the ADF is structured. For example, a row representing a node will always have the first column containing the name and the subsequent columns containing the acceptance conditions and their accompanying statements (in an alternating fashion), with the last column containing the statement to be shown if the acceptance condition is false. These rules, and others, ensure that the ADF generated from the imported file is consistent with what the user has created within the tool. An example of the generated save data can be found in Figure 11.

	A	B	C	D	E	F	G
1	Ownership	(OwnsLand and Resident) or Convention or Capture	the plaintiff owned t	the plaintiff did not own the quarry			
2	None						
3	RightToPursue	OwnsLand or ((HotPursuit and PMotive) or (PMotive and (not DMotive)))	plaintiff had a right t	plaintiff had no right to pursue the quarry			
4	None						
5	IllegalAct	Trespass or Assault	an illegal act was cor	no illegal act was committed			
6	None						
7	NoBlame						
8	Was the defendant blameless in the interference of the plaintiff's pursuit?						
9	OwnsLand	LegalOwner	plaintiff owned the l	plaintiff did not own the land			
10	None						

Figure 11 - A sample of the data saved to a '.xlsx' file for Wild Animals.

Section 5: Testing/ Results

This section will detail the testing of the software produced, the data used, and the results obtained. There are three tests which have been performed: a test to determine whether the same case outcomes and factors are reached within my implementation as the Prolog; a test to determine whether the save/import functionality works correctly; and finally, a description of the manual testing which occurred for the user interface.

5.1 Test Data

In order to ensure the ADFs function correctly within my implementation, it is crucial to test whether they can replicate the results previously achieved within the original Prolog programs. For this, I used the same cases as the Prolog, represented with the same base-level factors, which would act as the input in the tests. Each domain has a different number of cases: Wild Animals has 5, Trade Secrets has 32, Automobile Exception has 9 and NIHL has 3[1].

The input data required processing to be converted into a form my software would understand. The original cases used an abbreviation of the base-level factors such as 'ft011c' instead of 'car', whereas my implementation required that the inputs be strings and that they refer to the full node's name. These cases were then stored in a function called cases() which is present within the Python file instantiating each domain e.g. TradeSecrets.py. The base-level factors for these cases were determined in [5] for the public domain data via legal analysis of the cases.

[1] The expected outcome data for NIHL was created manually, going through the various acceptance conditions by hand until a result was obtained.

```
case(cvus,[ft011c,ft051m,ft211pi,ft221is,ft231all,ft233al,ft311is]).
```

Figure 12 - Case input data for Carroll v. United States in Prolog.

```
cvus = ['car','moving','public_informant','illegal_substance','all_parts','automobile_location','illegal_substance']
```

Figure 13 - Case input data for Carroll v. United States in Python.

After processing the inputs, I was required to collect the expected outcomes from the Prolog. This data was extracted by running the Prolog for each of the aforementioned cases and retrieving their results which were then converted, in the same manner as the input data, and stored separately in the expectedOutcomeCases() function.

Whilst for the most part the data collection was straightforward there were a couple of cases which caused some issue which I will detail here.

The first issue concerned the output data of the case Coolidge v. New Hampshire (cvnh) from Automobile Exception. When this case was run within the Prolog of the domain it would not generate an outcome, instead stopping part way through and returning ‘false’ due to a bug in the code. Consequently, this case has been excluded from the unit tests detailed in the next section as I was unable to obtain a reliable expected outcome.

The second concerned a couple of amendments I was required to make to the test data to account for some inaccuracies in the Prolog. The primary example concerns the LegitimatelyObtainable (f120) node. During some early tests of my software, I noticed a clear discrepancy in outcomes between my implementation and the Prolog’s in regard to whether LegitimatelyObtainable was present in the final outcome or not, despite having checked multiple times that the acceptance conditions were identical.

I found that in eight cases my implementation triggered LegitimatelyObtainable as a factor and in none of the cases was this triggered in the Prolog (see Appendix C for full details). Using Prolog's trace feature I was able to determine that this was due to a bug in the Prolog which evaluated LegitimatelyObtainable before its sole child node InfoKnownOrAvailiable (f105), meaning that LegitimatelyObtainable could never be accepted. Consequently, in any case which triggered InfoKnownOrAvailiable in the Prolog I manually appended LegitimatelyObtainable to the expected outcome in order to keep the expected outcome data consistent to how the Prolog was created to function. I also added the final outcome statement from the Prolog at the beginning of the expected outcome data to compare this statement with the final one generated by my software. For example, in the case Keeble v. Hickeringill I added "find for the plaintiff, find against the defendant".

There were also a couple of instances of typos in the input data within the Prolog I corrected, such as in the case US v. Chadwick from Automobile Exception, in which 'ft032pl' is not present within any acceptance condition, but 'ft032ps' is, so I interpreted the former as the latter when constructing the input data and re-ran the Prolog with this modification to obtain a more accurate outcome. However, in one instance in Chambers v. Maroney, I could not identify what the factor 'ft015w' was supposed to refer to, so left it as 'ft015w' in the input data.

All the data used in testing can be found in Appendix E.

5.2 Test 1: ADF Accuracy

The first test conducted was to determine whether the ADFs I had modelled functioned correctly and obtained the same results as the original Prolog implementations. I designed

unit tests for this purpose[2] Each test is for a different legal domain, and they all pass their ADF, cases and expected outcomes into the method query_adf() of the Tests() class. This method, as shown in Figure 14, processes some small aspects of the data and conducts two core tests. Firstly, it checks whether the outcome statement produced in the Python is the same as the outcome statement of the Prolog. Secondly, it takes the set of the case factors after they have been evaluated and checks whether they are equal. The reason it converts them into sets is because otherwise the order is considered when checking equality between them, and whether the factors are generated in the same order is not a relevant factor here.

```
def query_adf(self,adf,expectedOutcome,cases):

    for key,value in cases.items():

        #queries the case
        adf.evaluateTree(value)

        #removes the decide node from the outcome as not in the prolog
        try:
            adf.case.remove('Decide')
        except:
            pass

        #finds the expected outcome in the list of cases
        outcome = expectedOutcome[key]

        #removes the first item in the list which is a string of the e
        outcomeStatement = outcome.pop(0)

        #tests the outcomes are the same
        self.assertEqual(outcomeStatement,adf.statements[-1])

        #tests the individual fators are the same - converts each list
        self.assertEqual(set(adf.case),set(expectedOutcome[key]))
```

Figure 14 - query_adf() method

All the tests passed for each of the cases in each of the four domains, meaning they all achieved the same results as the Prolog implementations.

[2] The library unittest was used for this.

5.3 Test 2: Save/Import

The second test conducted was to ensure that the save/import functions of the software performed correctly. This, like the accuracy test, was done as part of the unit tests. These tests, which all call the save_import() method, save each ADF into a '.xlsx' file, and then import the '.xlsx' file as a new ADF. These two ADFs are then compared to check they have the same nodes as each other, that the question orders set within the ADF are preserved and finally that the imported ADF achieves the same outcome as the original ADF in regard to the tests performed in §6.2.

All the tests passed for each of the four domains, demonstrating the saving/import functions ability to capture and read the ADFs as raw data within a '.xlsx' file.

5.4 Test 3: UI Testing

Extensive manual testing was carried out on the user interface to ensure that all buttons and input boxes functioned as intended. I also carried out testing as I was linking the frontend to the backend ensuring that the backend functionality was fully functional within the frontend.

Section 6 Evaluation

In the design document I set out three key objectives which the success of this project would be evaluated against:

- **Accuracy:** The aim of this project is to bring together existing research which has been conducted in the ANGELIC project into a unified tool. To achieve this, the software I have developed must be able to replicate the results which have been achieved in the relevant legal domains [5][7].
- **Explainability:** A key aim in the ANGELIC project is to further the cause of explainable AI and create an alternative to 'black-box' decision-making within the legal field providing transparent outcomes in which each judgement can be explained and clearly shown to the user.
- **Usability:** The user-interface should not require an intimate familiarity with the various algorithms and methodologies involved to be able to obtain an accurate result in a pre-existing or new model of a legal domain.

I will proceed to explain how I have met each of the evaluation criteria as well as aspects which could be improved upon.

6.1 Accuracy

In regard to accuracy, this primarily concerns whether the tool achieves the same outcomes in the same cases as the Prolog implementations of the prior research on Wild Animals, Trade Secrets and Automobile Exception. In each of the cases I have replicated the results of prior research which originally achieved predictive accuracies of 96.8% in Trade Secrets, 100% in Wild Animals and 90% in Automobile Exception [5][3]. Whilst in NIHL the cases were

[3] As stated in §6.1 this is with the exception of the case Coolidge v. New Hampshire.

synthetic so there are no real predictive accuracies to compare them against, however the results, such as finding for or against the claimant, I predicted in the expected outcomes were achieved.

There were some discrepancies not covered by the unit tests which I uncovered during manual testing that are worth noting. Whilst the factors generated are identical to the expected outcomes, I noticed that the corresponding statements produced sometimes differ in one notable regard. Due to the differences in control structures between the Prolog and Python implementations, described in §4.1, my implementation prints additional statements to the Prolog. This arises since my implementation evaluates each node every time the ADF is evaluated whereas the Prolog, which has a more selective control structure, does not always consider every node. Figure 15 shows the additional statements shown to the user in the case California v. Carney (cvc) from Automobile Exception.

```
'default there are no rooms or rooms function is not specified',
'default it is not a vessel cite carroll v us',
'default it is not towable cite carroll v us',
'default the original probable cause is not by observation',
 'default the original probable cause is not a procedure or procedure is not clarified',
'default main reason to search was not due to a crime',
'default it is not clear which part of vehicle is searched',
```

Figure 15 - Additional statements from California v. Carney.

Despite the statements differing from the Prolog, they are not incorrect statements nor contradictions in the context of the case, just additional and sometimes slightly irrelevant information. For example, in cvc it says ‘default it is not a vessel…’ this is not incorrect since the vehicle in this case is a mobile home, not a vessel. It can be argued that this extra detail adding to an already long list of statements shown to the user may have the effect of obfuscating more relevant information, but overall, I believe the extra detail provides more transparency in the decision-making process which outweighs the potential for obfuscation.

Due to following the same methodology in designing my implementation of the ADFs and legal domains to the previous Prolog implementations, a limitation of this project is that it does not further increase the accuracy in any of the domains tested from the original results achieved in the Prolog.

6.2 Explainability

To achieve the goal of providing transparency in the decision-making process, it is important that the user is able to understand the process followed in reaching an outcome as well as the reasons for the decision. Figure 10 shows the outcome page after querying a case in Trade Secrets. It clearly displays each reason, in the order they were evaluated, and an outcome. By reading through these reasons, you can come to understand the "thought process" the tool used in coming to a decision. For example, in Figure 10, one can logically follow that from reason 6 'the information was available elsewhere' and reason 9 'information was not a trade secret', the outcome that no trade secret was misappropriated makes intuitive sense. However, a limitation in the understandability of this output is that I had hoped to be able to convert it to continuous prose rather than presenting it as a set of statements, in the hope it would be able to be read in a manner closer to a judge's written decision. But the logical thread underlying the decision is clearly visible within my implementation.

Whilst the outcome screen clearly shows the reasons behind the decision, I also wanted to allow the user to understand the process in reaching the decision without needing to remember each acceptance condition or understand the algorithms traversing the tree. Figure 16 shows the result of this. Using the Python library Pydot, the tool is able to generate visual representations of ADF which highlight green the nodes which have been accepted, and red the nodes which have been rejected. This enables the user to visually understand the decision-making process at a glance and see why a decision has been made. See Appendix F for full visualisations of each domain.

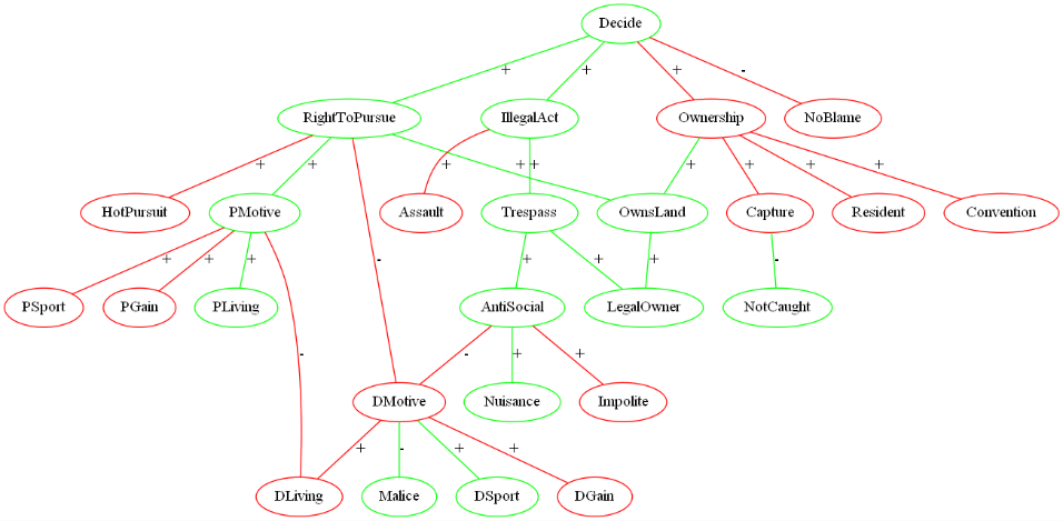


Figure 16 – Wild Animals ADF as generated within the software for the case Keeble v. Hickeringill.

6.3 Usability

In assessing usability, it is important that the functionalities of the software are easy to navigate and use. To achieve this, I have:

- Kept the user interface clean and consistent.
- Clearly labelled buttons and labels for user input.

- Implemented red question marks providing pop-up text to help with some rules regarding user input which are not immediately obvious, e.g., a domain name should have no spaces between the words. These tips help provide useful reminders to ensure the software is used correctly. Unfortunately, due to time constraints the use of hover over tips is abundant on some screens and absent from others.

- Written a user manual (see Appendix G), navigable at any time from the help button, detailing each screen present within the software and guiding the user through the best practices and rules in place to ensure a domain is queried or created successfully within the tool.

6.4 Learning Points

In this section I will surmise some of the key learning points in the project.

6.4.1 User Interface

The user interface for this software was the first user interface I had ever created. There were a few struggles in realising this objective, especially as it took me a little while to understand how to properly use Tkinter to seamlessly change between pages in the UI. There was even a moment of panic in trying to learn this library leading me to try another similar library called PyQt, but this didn't quite offer enough flexibility and customisation, so I persevered with Tkinter. In the end I was pleased with the results obtained from Tkinter and feel a lot more confident in creating user interfaces, though I would like to have had more time to really understand Tkinter in order to implement more functionality, as some aspects I wanted to include such as a 'back button', didn't work very well with the backend so were abandoned.

6.4.2 Saving Data

One aspect I would do differently, despite it being functional, is the ability to save ADFs. Originally, I wanted to implement this with an SQL database which would have facilitated a more dynamic saving process since ADFs that were created within the software would be automatically loaded and edited ADFs would update immediately. Due to my lack of experience in accessing databases through Python this did not end up happening, and I stuck with an easier to implement though more static implementation based upon excel files. Whilst this implementation functions perfectly fine, I feel that if I had taken some more time to really think about and design this feature, I could have created it in a far more efficient and dynamic manner.

6.4.3 Unit Testing

Towards the end of completing the project I began to delve deeper into unit testing as I wanted to test my code more robustly than the testing functions I had written would allow me to, especially with regard to the save/import features. These tests have made the testing procedure more rigorous and allowed me to easily spot and solve any problems which came as a result of running them. In the future I would use a more test-driven development model and create the code to fit the tests, which I believe would have saved me a lot of time and debugging headaches.

Section 7 Professional Issues

In this section I will discuss how my project is related to the British Computer Society's Code of Conduct.

7.1 IT for everyone

Accessibility and the idea of IT being as inclusive as possible to both those inside and out of the field is at the heart of many of my design decisions around usability. The software has been designed so that it is usable by someone who is not a computer scientist, e.g. a legal professional, so that the user is not required to be too familiar with the algorithms and methodologies underlying the implementation in order to receive an outcome to the case queried which is clear, concise and understandable.

7.2 Say what you know, learn what you don't

During this project there have been many tasks I lacked the knowledge to complete, so a sizeable part of this project was spent learning and understanding new techniques such as drawing a UML diagram, or writing pseudocode for the first time, to designing a user interface and learning how to use Python libraries to assist in the visualisation component. The entire process has been one of growth and reflection in my skillset.

7.3 Respect the organisation

Whilst I did not directly work with an organisation on this project, I did work with data which came from an industry collaborator who had previously partnered with the University of Liverpool. This data allowed me to model the ADF for NIHL. In order to respect this organisation and the confidentiality of their data I am not reproducing the acceptance conditions nor showing a visualization of the domain within this paper.

7.4 Keep IT real

To keep IT real and pass it on I believe this project helps to further and promote the integration of computational techniques in the legal field. The development and deployment of AI systems applied to law is increasingly happening in the real-world [16] and this paper helps to showcase how some of this research can be deployed in a responsible and effective manner.

Section 8 Conclusion

To conclude, this project was very successful in achieving its aims and objectives. In this work I have aimed to explain how the software I have developed has achieved the aims and objectives set out at the beginning to create a tool which can be used to create, edit and query ADFs. I have shown how the tool was designed with flexibility in mind from the outset to enable ADFs to be dynamically created by the user for the legal domains covered within this paper or new legal domains yet to be modelled. I have then demonstrated how this was implemented within the frontend and backend as well as with the extra features I added such as saving and loading ADFs to facilitate editing them in light of new legal precedents, or graphical visualisations of the case decision-making process. In testing the software, I was able to achieve the expected outcomes in each legal domain, replicating the previous results achieved in earlier Prolog implementations. Finally, I have evaluated the tool against the evaluation criteria of usability, accuracy and explainability, concluding that the tool has for largely met these criteria providing a transparent, user-friendly, and accurate implementation of the ANGELIC project's research.

8.1 Further Work

In regard to further work, I would love to have been able to model a new legal domain in this software, but this was beyond the scope of the project. This was due to the need for legal analysis in determining the factors and acceptance conditions for the ADFs, as well as determining the base-level factors for any real-world cases to be tested on this.

As computational efficiency was not a core objective in this project nor one of the evaluation criteria, this was not an area I was able to pay as much attention to as I would like. Whilst the algorithms run fairly quickly with the ADFs modelled in this paper, it is certainly feasible to imagine with larger ADFs for more complex legal domains that this could slow down the

running of the program. Future work focusing on more computationally efficient streamlined versions of these algorithms or using multi-threading and other related techniques would improve the outcomes achieved here.

REFERENCES

[1] Aletras, N., et al. (2016). Predicting judicial decisions of the European Court of Human Rights: a Natural Language Processing perspective. *PeerJ Computer Science., 2*, Article e93.

[2] Collenette, J., et al. (2020). An explainable approach to deducing outcomes in European court of human rights cases using ADFs.

[3] European Union. (2010). Charter of Fundamental Rights of the European Union. In *Official Journal of the European Union C83*, Vol. 53: 380, European Union.

[4] Hacker, P., et al. (2020). Explainable AI under contract and tort law: legal incentives and technical challenges. *Artificial Intelligence and Law* 28, 415–439.

[5] Al-Abdulkarim, L., et al (2016) A methodology for designing systems to reason with legal cases using ADFs. *Artificial Intelligence and Law*. 24(1):1–49.

[6] Dung, P.M. (1995) On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming, and n-person games. *Artificial Intelligence*, Vol. 77: 321–357.

[7] Al-Abdulkarim, L., et al (2017): Noise induced hearing loss: An application of the ANGELIC methodology. In Wyner, A. Legal Knowledge and Information Systems - JURIX 2017: The Thirtieth Annual Conference, Luxembourg, pp. 79-88. *Frontiers in Artificial Intelligence and Applications*, Vol. 302. IOS Press, Amsterdam,

[8] Dawson, C. (2015). *Projects in Computing and Information Systems 3rd edition: A Student's Guide (3rd. ed.).* Prentice Hall Press, USA.

[9] Medvedeva, M., et al. (2020) Using machine learning to predict decisions of the European Court of Human Rights. *Artificial Intelligence & Law* 28, 237–266.

[10] Katz, D.M., et al. (2017). A general approach for predicting the behavior of the Supreme Court of the United States. *PLOS ONE* 12(4): e0174698.

[11] Aleven, V. (1997). *Teaching case-based argumentation through a model and examples.* PhD thesis, University of Pittsburgh.

[12] Ashley, K. (1990). *Modelling Legal Argument: Reasoning with Cases and Hypotheticals.* Bradford Books/MIT Press, Cambridge, MA.

[13] Agrawal, S., et al. (2017*), A*ffirm or reverse? using machine learning to help judges write opinions, Technical report, Working Paper.

[14] Aleven, V., (2003). Using background knowledge in case-based legal reasoning: A computational model and an intelligent learning environment. *Artificial Intelligence*, Vol 150: 183-237.

[15] Nugues, P.M., (2014). *Language Processing with Perl and Prolog*, Springer-Verlag Berlin Heidelberg.

[16] Kauffman, M.E., Soares, M.N., (2020). AI in legal services: new trends in AI-enabled legal services. *SOCA* 14, 223–226.

APPENDIX A – Design Documentation

The following is the original design documentation submitted as part of the Specification and Proposed Design assessment.

Design

The system design aims to realise several key objectives and implement the required features necessary to meet these. These features are as follows:

- **Query an existing ADF**: The end-user will be able to query an existing ADF by answering a number of questions corresponding to each of the base-level factors in the ADF. For example, if a base-level factor was Trespassing, the question may be "Did the defendant trespass upon the plaintiff's land?", which may be answered 'Yes' or 'No' depending on the facts of the case the user is wishing to predict the outcome of. After the base-level factors have been generated, the tool should generate an outcome as well as showing the different factors which went into making the decision, for example, "The plaintiff had a good motive".

- **Editing an existing ADF**: The ability for the user to edit and update existing ADFs in light of new legal precedents. The motivation for this comes from the work conducted in modelling the 'US Automobile Exception to The Fourth Amendment' domain [5]. In this domain there was one case which was not predicted correctly, California vs Acevedo, because the judge overruled the previous precedent in this case. Since the ADF is a model of legal precedent it is crucial to be able to amend them when changes to the precedent occur.

- **Creating an ADF**: This tool will seek to allow the user to create an ADF, for a legal domain of their choice. Defining the factors, relationships, acceptance conditions, acceptance/rejection statements and questions instantiating the base-level factors. This will allow the tool to be flexible in not only modelling the domains which have been modelled such as Wild Animals or Trade Secrets but also facilitate the construction of ADFs for new domains.

- **User Interface**: This implementation will provide a clear graphical user-interface which will allow the end-user to easily use any of the aforementioned features.

Methodology and Technology

To achieve these design objectives, I will be implementing this tool in Python 3. Whilst the initial implementations were achieved in Prolog [5], due to the declarative nature of the acceptance conditions, I will be using Python because I believe it offers the flexibility and robustness necessary in developing the entire tool from start to finish, and it is the language I am the most familiar with. For the frontend I will be using the Tkinter since it is well documented and offers a sufficient degree of customisation to be able to achieve the project outcomes.

I will follow the Object-Oriented Design paradigm in designing the system. The two primary classes will be the Node class - representing the individual factors as nodes, i.e., Trespassing,

with their associated acceptance conditions etc and the Tree class - modelling the ADF in terms of the individual nodes' relationships with one another.

The software development process I will be using will be an Agile methodology. This method aims to develop functional systems quickly to the user, incrementally iterating and expanding the features available [8]. This is beneficial for this project as it will allow me to seek feedback from my supervisor regularly and easily implement any proposed changes for the next iteration, ensuring that the project does not stray from the end-user's desired outcome. Additionally, an Agile approach will help to ensure that the core systems are completely functional before any additional or desirable features are implemented.

Backend Design

To instantiate an ADF within the system, the first instances which will be created will be those of the 'Node' class. The pseudocode for the construction of the class is shown below in figure 1. This class will take in the factor name, children nodes, acceptance conditions and the statements to be output depending on whether the factor is or is not present within the case.

An example of the data which may be used to create a node instance [5]:

Factor Name: Trespass

Children: LegalOwner, AntiSocial

Acceptance Conditions: LegalOwner and AntiSocial

True Statement: "Defendant committed trespass"

False Statement: "Defendant committed no trespass"

A key method of the 'Node' class is the conversion of the acceptance conditions from infix to postfix notation. This is due to the challenge of parsing logical conditions which may contain parenthesis such as in "Capture if NOT (NotCaught) OR (Vermin and HotPursuit)" [1]. The advantage of postfix notation is that it enables logical statements to be expressed without parenthesis whilst maintaining operator precedence. In this form it is fast and efficient to check whether the acceptance conditions are satisfied or not for a particular case's facts. Figure 2 demonstrates a variation of the standard algorithm for infix to postfix in pseudocode.

The postfix expression generated can then be evaluated with regard to the case facts. Figure 3 demonstrates an example of what the evaluation for an 'Or' clause might look like.

Fig 1: The Node Class

```
CLASS Node DO
  FUNCTION constructor(name, children, acceptanceConditions, statements ) DO
    SET name: str
    SET children: list
    SET acceptanceConditions: str
        SET statements: list

  END

  FUNCTION acceptanceConditions(acceptanceConditions) DO
```

```
    IF acceptance condition stated:
        SET acceptanceConditions = CALL logicConverter(acceptanceConditions)
  END
```

Fig 2: Infix to Postfix Conversion

```
FUNCTION logicConverter(acceptanceConditions) DO
    SET stack: stack
    SET tokenList from acceptanceConditions: list
    SET postFixList: list
    FOR token in tokenList DO
        IF logical operator i.e and, not, or → pop higher/equal precedence
        operators to postFixList then push operator to stack
        IF '(' → push to stack
        IF ')' → pop operators until '(' popped and delete brackets
        IF operand i.e a factor - IllegalAct → add to postFixList
    RETURN postfixList as string
    END
  END
```

Fig 3: OR operator evaluator

```
  IF operator = or DO
    IF operand1 OR operand2 = TRUE
      RETURN TRUE
    ELSE-IF operand1 OR operand2 in cases
      RETURN TRUE
    ELSE
      RETURN FALSE
```

Fig 4: The ADF Class

```
CLASS ADF DO
  FUNCTION constructor(name) DO
    SET name from USERINPUT: str
    SET nodeList: dict
  END
  FUNCTION addNodes(name, children, acceptanceConditions) DO
    SET node = CALL Node(name, children, acceptanceConditions)
    Add node to nodeList
  END
  FUNCTION evaluateTree(cases) DO
    SET nodeChildren: dict
    SET nodesEvaluated: list

    FOR node in nodeList DO
      IF node.children != empty
        Add node to nodeChildren
    END

    FOR node in nodeChildren DO
      IF CALL childCheck(node) DO
```

```
            Add node to nodesEvaluated
            IF CALL evaluateNode(node) = TRUE DO
                add node name to cases
                pop node from nodeChildren
            ELSE DO
                pop node from nodeChildren
    END
END
            HELPER FUNCTION childCheck(node) DO
    FOR i in node.children DO
        IF i in nodeChildren
            IF i in nodesEvaluated
                PASS
            ELSE RETURN FALSE
        ELSE PASS
    RETURN TRUE
    END
END
HELPER FUNCTION evaluateNode(node) DO
    RETURN CALL evaluateLogic(node.acceptance, cases)
END
```

An instance of the 'Node' class can be added to the 'ADF' class which will hold all the nodes and the relationships between them. The nodes will be stored in a dictionary with the name of the node as the key and the node object itself as the value. A key function within this class is the algorithm to evaluate an ADF in respect to a given case. In the pseudocode in figure 4, this is the evaluateTree method. This algorithm proceeds in the following manner:

1. For each node it identifies those which have children (non-leaf nodes) and adds them to a dictionary. (This is because the leaf nodes do not have acceptance conditions to evaluate, with their values coming from the questions asked in the user interface concerning the facts of the case)

2. For a non-leaf node, it checks whether the children of that node have been evaluated before or whether they are all leaf nodes. In the first iteration, for instance, this would result in the algorithm finding a node where all its children are leaf nodes to evaluate first. This creates the effect of ascending the ADF.

3. Next, it tests the acceptance conditions regarding the factors of the case. If the acceptance condition is satisfied, then it adds that node's factor to the factors of the case and deletes that node from the list of non-leaf nodes to be evaluated.

4. It returns to step 2 until there are no nodes left which have not been evaluated, at which point it presents the outcome.

Those classes and methods represent the core algorithms in the tool in order to model the ADFs accurately. But, in the final version I would also expect to have: ways of saving an ADF once created and enabling it to be imported or selected again; a backend procedure for allowing editing of the ADF, which should be rather straightforward to implement given the

flexible class structure which contains it; and a reporting function which enables the user to have a greater idea of the structure of the underlying ADF.

Frontend Design

The below diagram (figure 5) shows a wireframed mock-up of the graphical user interface for the tool. The arrows show the different pathways the user can take depending on whether they want to create, edit, or query an ADF for a specific legal domain. The following will provide some greater detail on what each screen of the diagram represents.

Main Menu

- This screen acts as the homepage for the user. It allows a selection between the two core functionalities within the app: creating a new ADF and querying an existing ADF.

Domain Selection

- Uses a drop-down menu to allow the user to choose a previously created ADF to query or edit.

Edit or Query Selection

- Allows the user to choose whether to edit or query the ADF.

Edit Node

- The user can change the name, acceptance conditions or children of a node. Will likely also include the ability to easily delete a node as well.

Questions to User

- If the user is querying the ADF this screen allows the questions (which result in the base-level factors being established) to be shown to the user one at a time. Each question is compulsory, and they must be answered sequentially. The questions will ordinarily be 'Yes'/'No' questions.

Outcome

- This screen will display the outcome of the case which the process has reached. It will also display the factors which led to this decision. Desirably this output would be in continuous prose rather than bullet points.

Domain Creation

- The user can specify the name of the domain for which they wish to create an ADF. Furthermore, this instantiates the ADF.

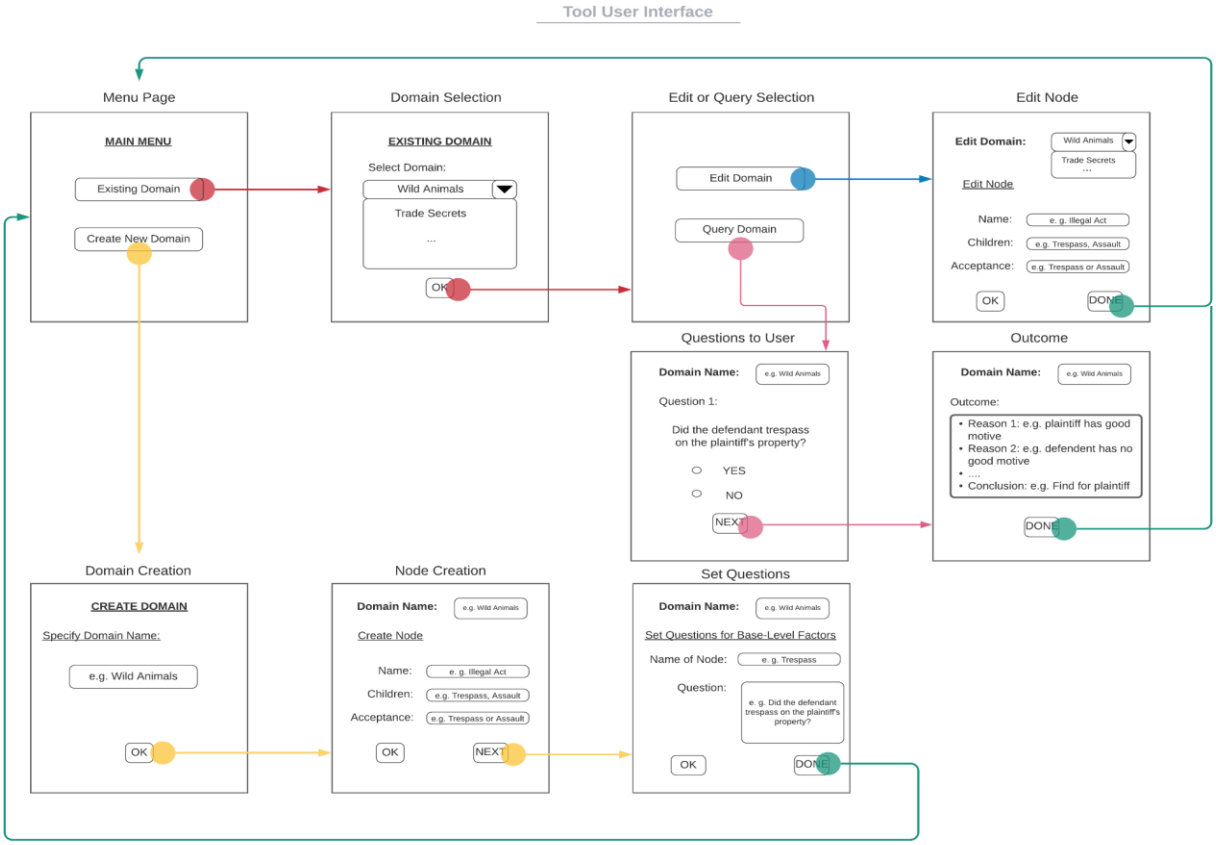
Node Creation

- This enables the user to create the factor as a node with a name, children, and acceptance conditions.

Set Questions

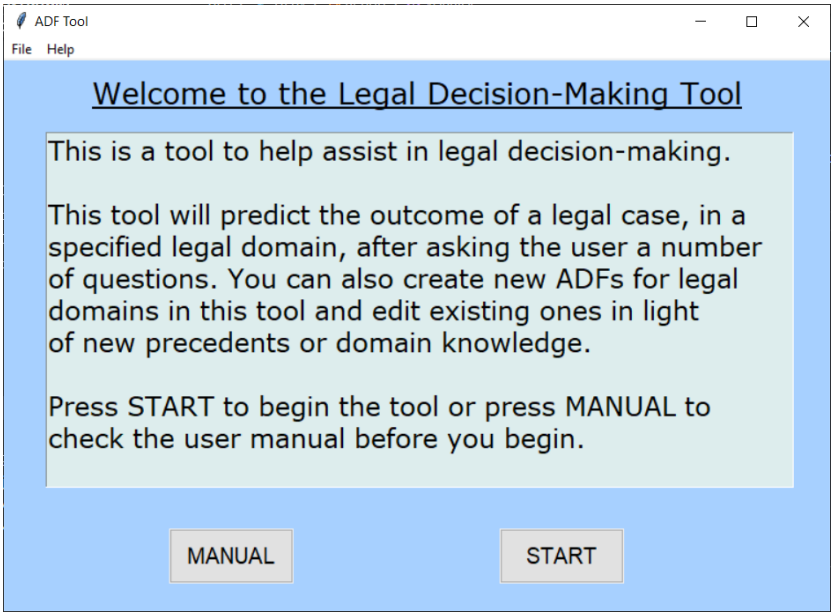
- On this screen the questions corresponding to the base-level factors can be manually entered. Ideally, this screen would present the user with the base-level factor nodes rather than relying on them remembering the names of them.

Fig 5: The User-Interface

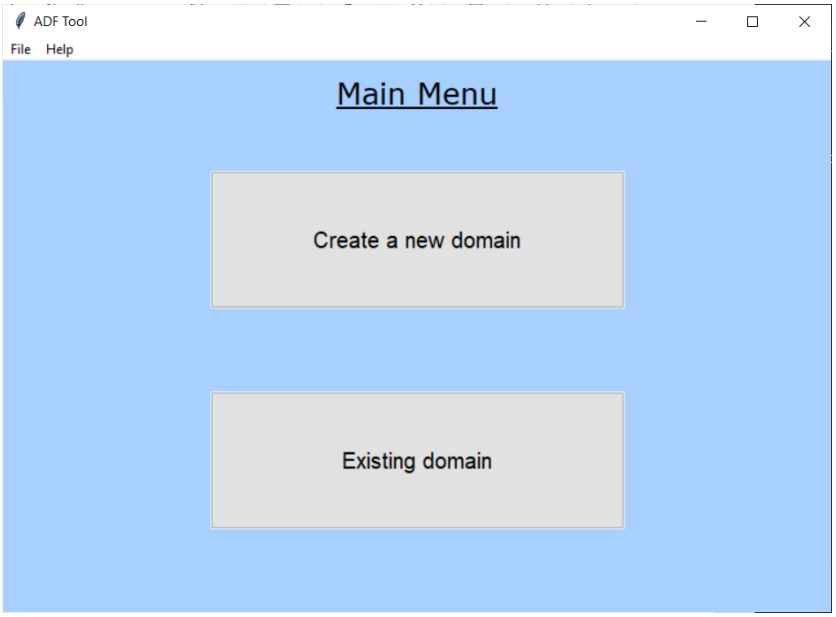


APPENDIX B – UI

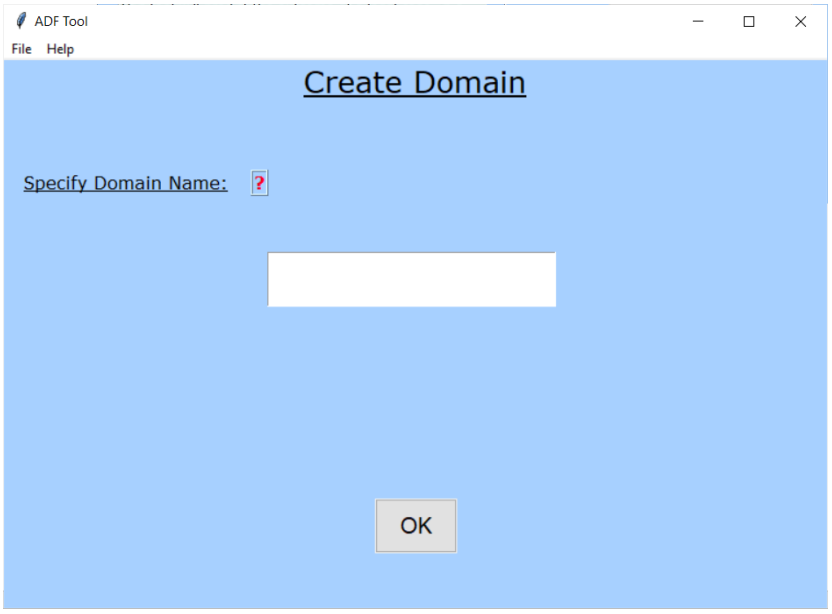
This appendix will show the different screens created within the UI and will briefly describe the purpose of each screen.



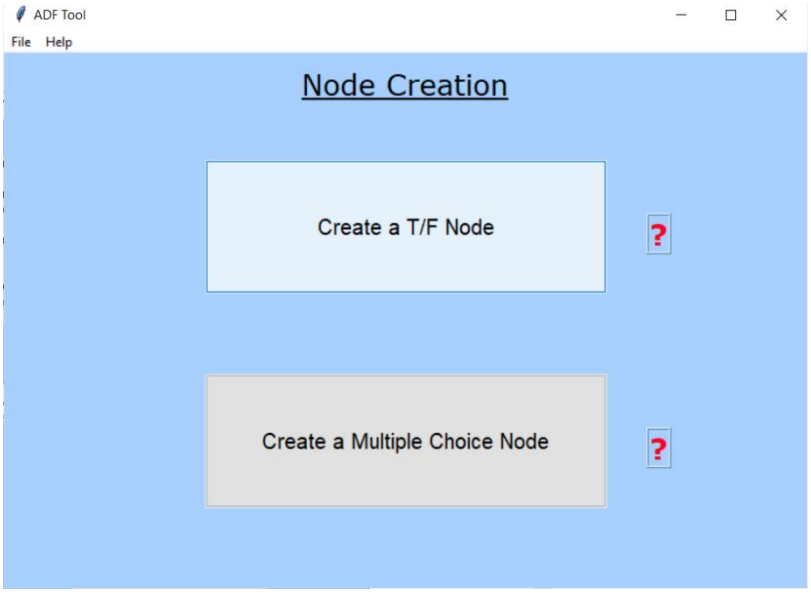
Welcome screen – provides a brief overview of the tool and allows the user to read the manual or begin using the application.



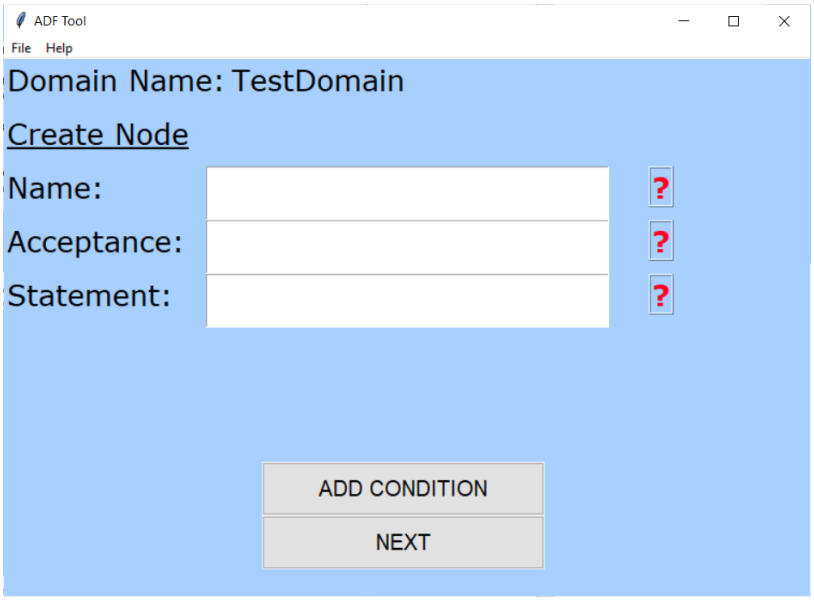
Main Menu – allows the user to create a new domain or select an existing domain.



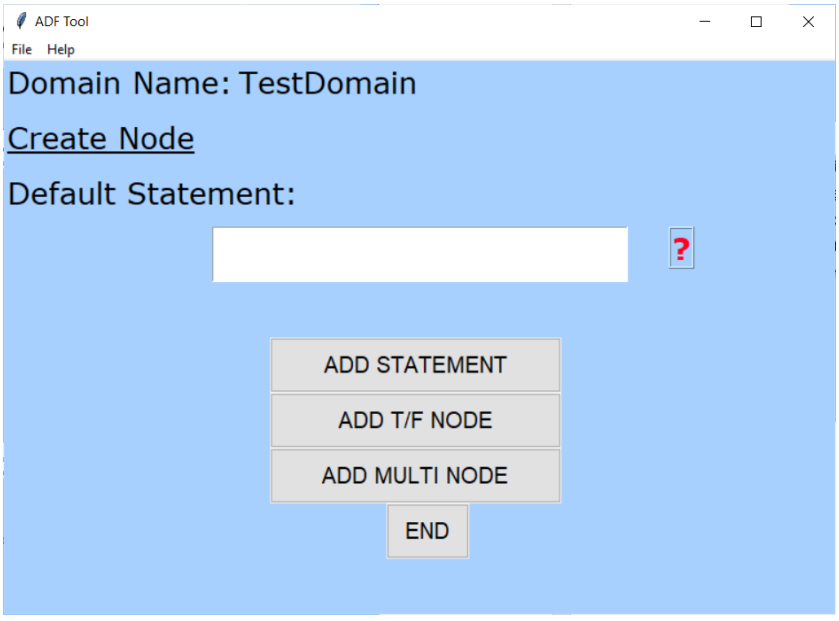
Create Domain – the user can set the name of the new domain they are creating



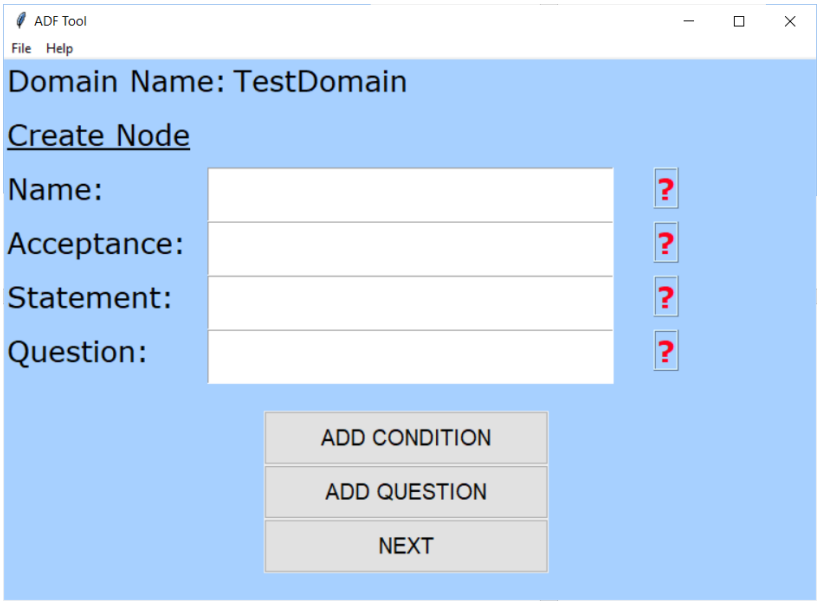
Node Creation – the user can choose to create a true/false node or a multiple choice node



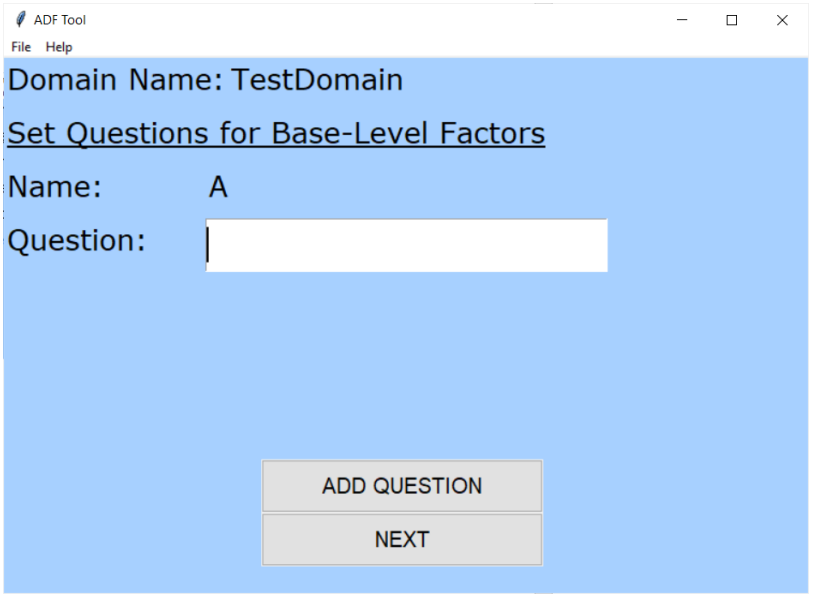
True/False Node Creation – the user can set the name, acceptance condition and statement to be printed when the node is accepted.



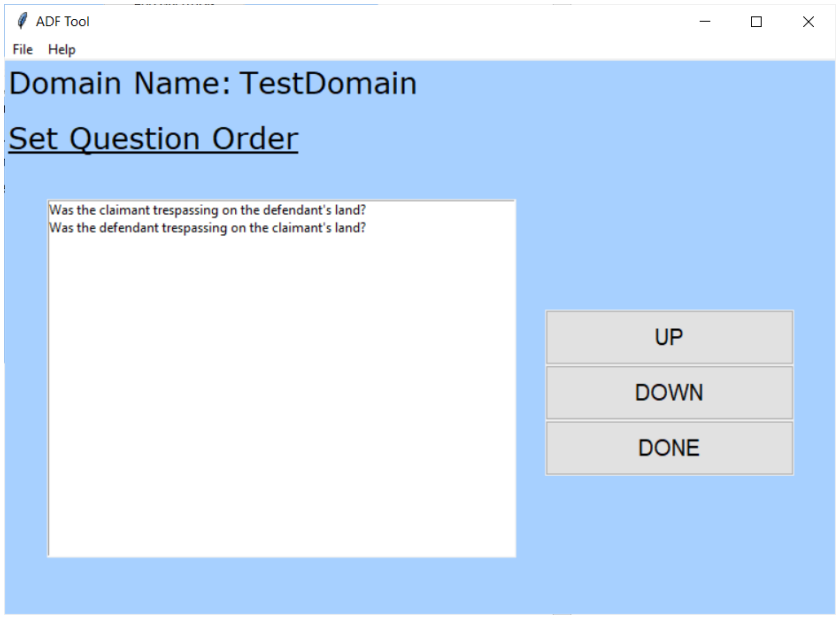
Default Statement – the user can set a statement to be printed when the node is not accepted. This screen is the same regardless of which type of node you create.



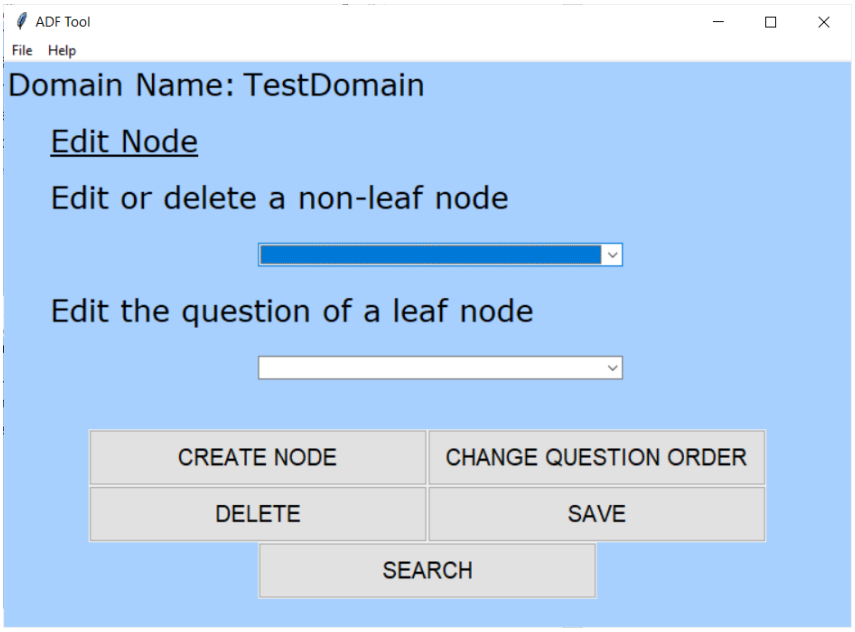
Multi-Choice Node Creation - the user can set the name, acceptance condition and statement to be printed when the node is accepted as well as the question to instantiate the node.



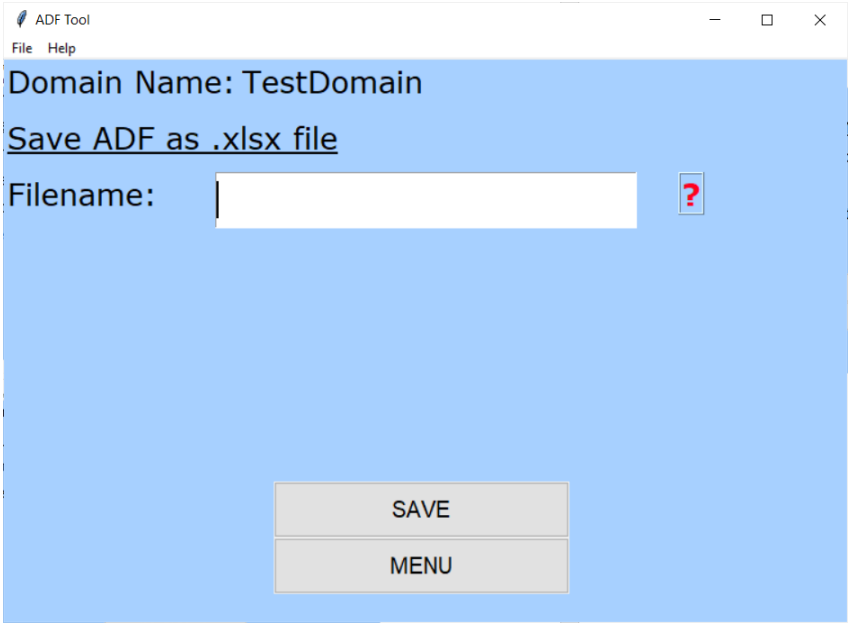
Question Creation – the software generates the base level factors which have not had questions set for them and prompts the user to set those questions.



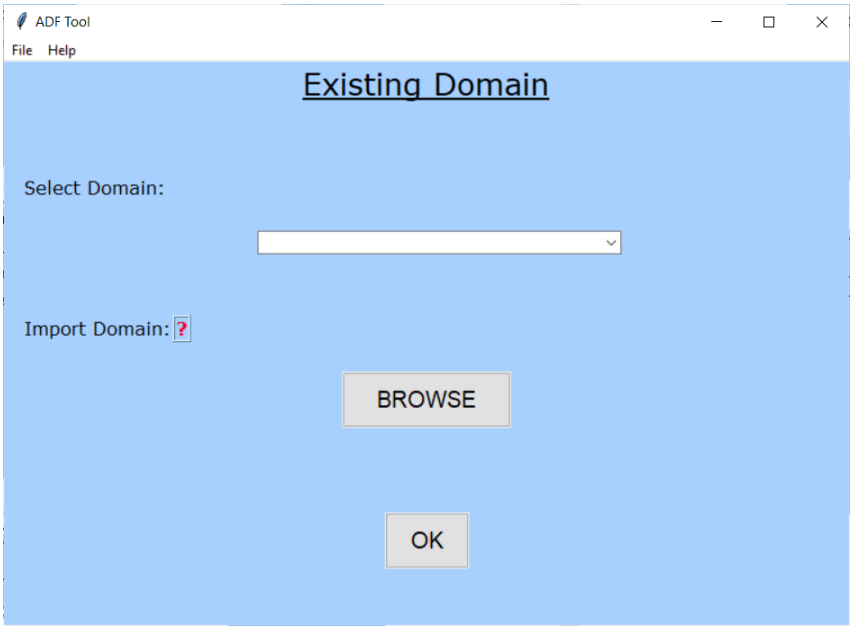
Question Order – the user can set the order the questions will be asked when the ADF is queried.



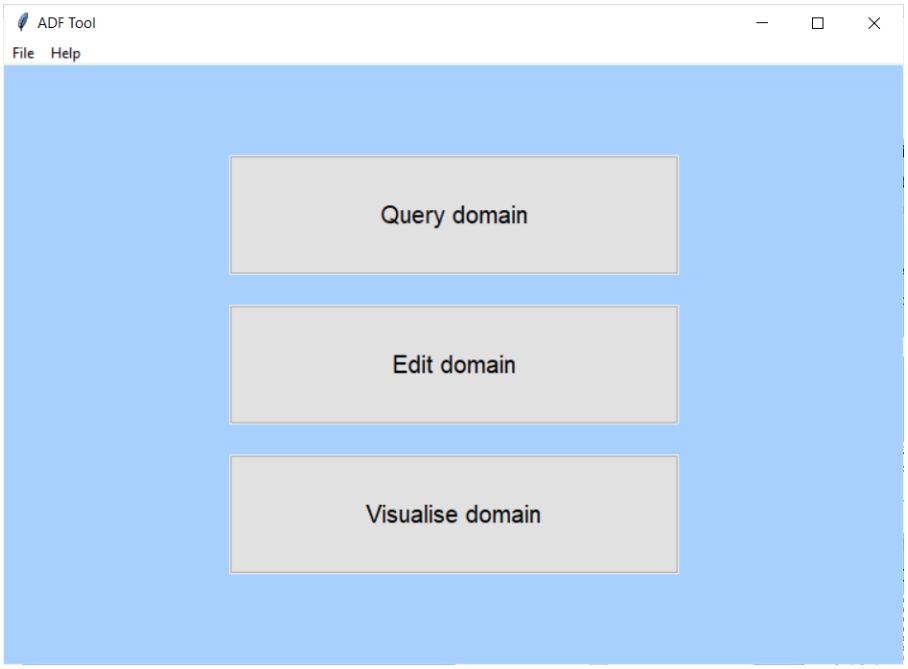
Edit Domain – the user can edit an individual node's acceptance conditions, name or statement; they can edit the question of a leaf node; the question order; delete nodes; add new nodes to the domain and finally can save the domain as a '.xlsx' file.



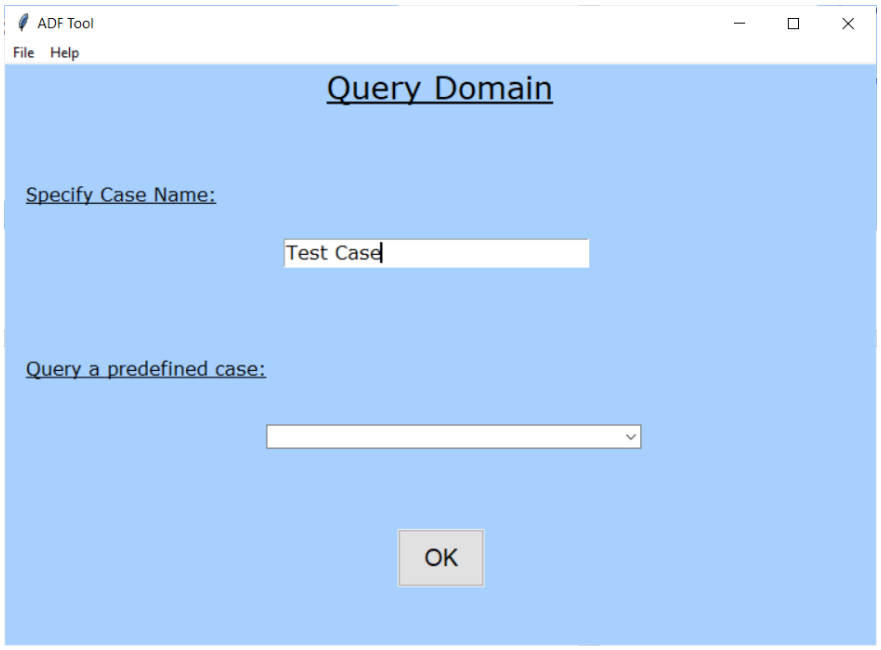
Save – the user can choose a filename to save the ADF as.



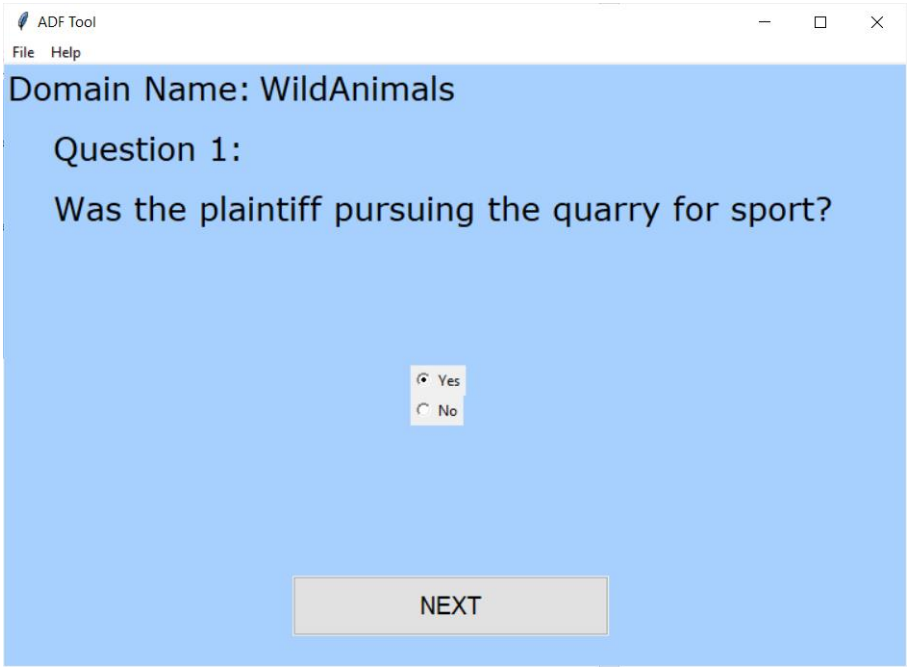
Existing Domain – the user can import a previously created '.xlsx' file in or can select one of the pre-existing domains.



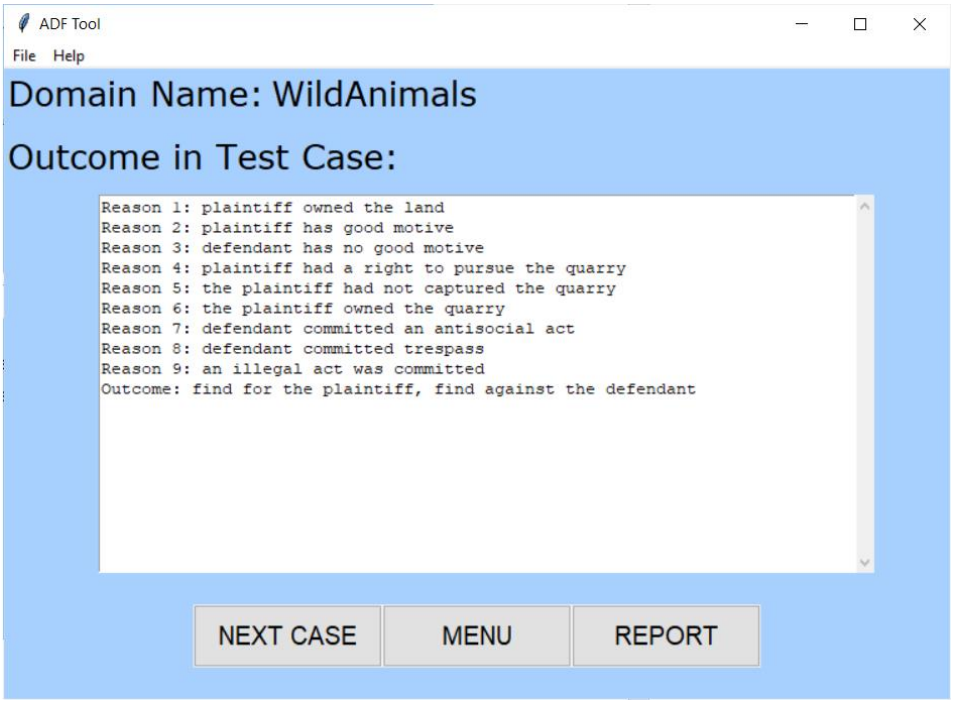
Query Menu – the user can query, edit or visualise an existing domain.



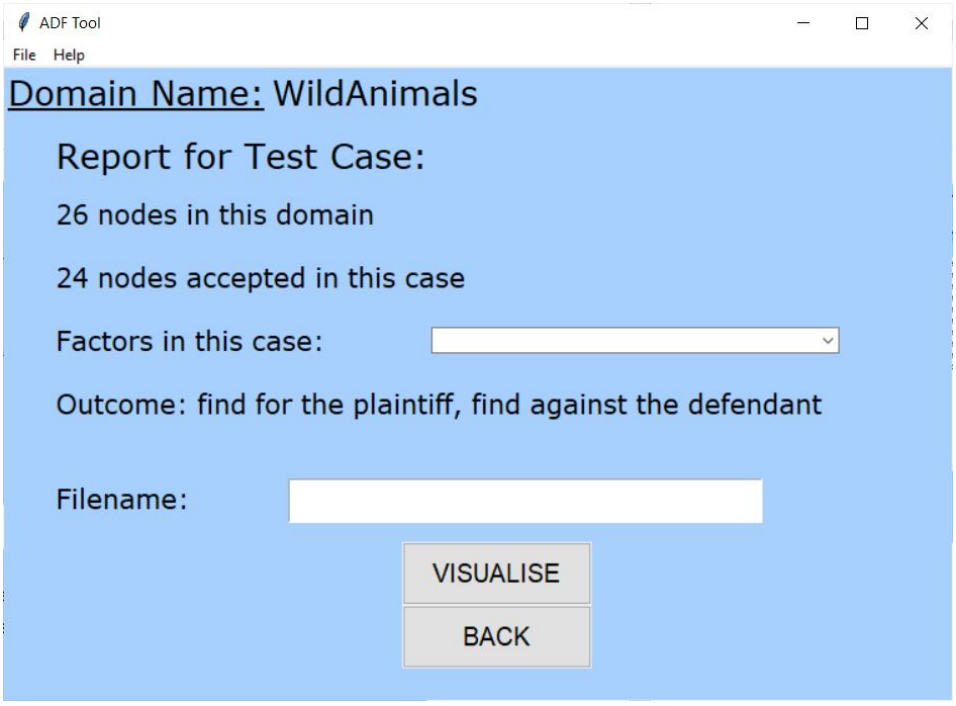
Query Domain – the user can specify the case name or query a predefined case.



Questions – An example of a 'yes' or 'no' question presented to the user to instantiate the base-level factors.



Outcome – this screen shows the reasons the outcome was reached in that particular case.



Report – this screen shows the report features included such as how many nodes are in the domain, how many have been accepted within the case and allows the visualisation to be generated.

APPENDIX C §5.1 Investigation

I noticed when comparing the results of the Prolog with my own implementation in Python that the LegitimatelyObtainable node, referred to in Prolog as 'F120' triggered in the following cases in mine: arco, ferranti, sandlin, yokana, cm1, mbl, mason, scientology but did not trigger in a single case within the Prolog.

The first place to look was the acceptance condition for this node which is one of the simplest in any ADF. Within the Prolog it is constructed as below:

```
getf120(C,Factors):-member(f105,Factors),
                    write([the,information,was,legitimately,obtained]),nl,
                        getf110(C,[f120|Factors]).

getf120(C,Factors):-write([the,information,was,not,legitimately,obtained]),nl,
                        getf110(C,Factors).
```

The acceptance condition here is that if F105 is present then we should accept F120. The odd thing was that F105 was present in each of the cases which triggered F120 in my implementation, so the question became why was this not triggering in the Prolog?

To answer this, I used the trace functionality in Prolog for the case Yokana which gave the following output (this output has been abridged to show only the relevant sections):

Call: (19) getf120(yokana, [f112, f7, f10, f16, f27]) ? creep

Call: (20) lists:member(f105, [f112, f7, f10, f16, f27]) ? creep

Fail: (20) lists:member(f105, [f112, f7, f10, f16, f27]) ? creep

Redo: (19) getf120(yokana, [f112, f7, f10, f16, f27]) ? creep

Call: (20) write([the, information, was, not, legitimately, obtained]) ?creep

[the,information,was,not,legitimately,obtained]

Exit: (20) write([the, information, was, not, legitimately, obtained]) ?creep

---

Call: (23) getf105(yokana, [f108, f112, f7, f10, f16, f27]) ? creep

Call: (24) lists:member(f108, [f108, f112, f7, f10, f16, f27]) ? creep

Exit: (24) lists:member(f108, [f108, f112, f7, f10, f16, f27]) ? creep

Call: (24) write([the, information, was, known, or, available]) ? creep

[the,information,was,known,or,available]

Exit: (24) write([the, information, was, known, or, available]) ? creep

Call: (24) nl ? creep

Exit: (24) nl ? creep

Call: (24) getf104(yokana, [**f105,** f108, f112, f7, f10, f16, f27]) ?

The trace shows that F105 is being evaluated after F120, this means that F120 can never be triggered in the Prolog since it has no knowledge of whether F105 will or will not be accepted. Consequently, I could feel comfortable in the knowledge that my implementation was working correctly since this was a bug in the Prolog and not a fault in outcome of my implementation.

APPENDIX D: Test Data

This section will show the test data, both the case input and the expected outcomes which came from the Prolog. Each line in the input gives the case name and a list of the base-level factors. Each line of the expected outcome gives the case name and a list of the factors, including non-leaf factors, which should be in the case after evaluation. Another aspect of the evaluation test data is that the first item in the list corresponds to the case outcome which would be displayed to the user to quickly check the final outcome in testing.

Wild Animals Test Data

Input

keeble = ['NotCaught','LegalOwner','Malice','Nuisance','DSport','PLiving']

pierson = ['NotCaught','HotPursuit','Impolite','PSport','Vermin']

young = ['NotCaught','HotPursuit','Impolite','PLiving','DLiving']

ghen = ['NotCaught','Convention','NoBlame','PLiving','DLiving']

popov = ['NotCaught','HotPursuit','Assault','NoBlame','PGain','DGain']

Expected Outcome

keeble = ['find for the plaintiff, find against the defendant','IllegalAct','Trespass','AntiSocial','RightToPursue','OwnsLand','PMotive','NotCaught','LegalOwner','Malice','Nuisance','DSport','PLiving']

pierson = ['do not find for the plaintiff, find for the defendant','AntiSocial','RightToPursue','PMotive','NotCaught','HotPursuit','Impolite','PSport','Vermin']

young = ['do not find for the plaintiff, find for the defendant','RightToPursue','DMotive','PMotive','NotCaught','HotPursuit','Impolite','PLiving','DLiving']

ghen = ['find for the plaintiff, find against the defendant','DMotive','PMotive','Ownership','NotCaught','Convention','NoBlame','PLiving','DLiving']

popov = ['do not find for the plaintiff, the defendant did not act illegally, do not find against the defendant','IllegalAct','RightToPursue','DMotive','PMotive','NotCaught','HotPursuit','Assault','NoBlame','PGain','DGain']

Trade Secrets Test Data

Input

arco = ['SecretsDisclosedOutsiders', 'InfoReverseEngineerable','InfoKnownToCompetitors']

boeing = ['AgreedNotToDisclose','SecurityMeasures','OutsiderDisclosuresRestricted','RestrictedMaterialsUsed','KnewInfoConfidential','DisclosureInNegotations','SecretsDisclosedOutsiders']

bryce = ['AgreedNotToDisclose','SecurityMeasures','IdenticalProducts','KnewInfoConfidential','DisclosureInNegotations']

collegeWatercolour = ['UniqueProduct','Deception','DisclosureInNegotations']

denTalEz = ['AgreedNotToDisclose','SecurityMeasures','KnewInfoConfidential','Deception','DisclosureInNegotations']

ecolgix = ['KnewInfoConfidential','DisclosureInNegotations','NoSecurityMeasures','WaiverOfConfidentiality']

emery = ['IdenticalProducts','KnewInfoConfidential','SecretsDisclosedOutsiders']

ferranti = ['BribeEmployee','InfoIndependentlyGenerated','NoSecurityMeasures','InfoKnownToCompetitors','DisclosureInPublicForum']

robinson = ['IdenticalProducts','Deception','DisclosureInNegotations','SecretsDisclosedOutsiders','NoSecurityMeasures']

sandlin = ['DisclosureInNegotations','SecretsDisclosedOutsiders','InfoReverseEngineerable','NoSecurityMeasures','DisclosureInPublicForum']

sheets = ['IdenticalProducts','NoSecurityMeasures','DisclosureInPublicForum']

spaceAero = ['CompetitiveAdvantage','UniqueProduct','IdenticalProducts','DisclosureInNegotations','NoSecurityMeasures']

televation = ['SecurityMeasures','OutsiderDisclosuresRestricted','UniqueProduct','IdenticalProducts','KnewInfoConfidential','SecretsDisclosedOutsiders','InfoReverseEngineerable']

yokana = ['BroughtTools','SecretsDisclosedOutsiders','InfoReverseEngineerable','DisclosureInPublicForum']

cm1 =
['AgreedNotToDisclose','SecurityMeasures','InfoKnownToCompetitors','InfoIndependentlyGenerated','InfoReverseEngineerable','SecretsDisclosedOutsiders','DisclosureInPublicForum']

digitalDevelopment =
['SecurityMeasures','CompetitiveAdvantage','UniqueProduct','IdenticalProducts','KnewInfoConfidential','DisclosureInNegotations']

fmc =
['AgreedNotToDisclose','SecurityMeasures','BroughtTools','OutsiderDisclosuresRestricted','SecretsDisclosedOutsiders','VerticalKnowledge']

forrest =
['SecurityMeasures','UniqueProduct','KnewInfoConfidential','DisclosureInNegotations']

goldberg =
['KnewInfoConfidential','DisclosureInNegotations','SecretsDisclosedOutsiders','DisclosureInPublicForum']

kg =
['SecurityMeasures','RestrictedMaterialsUsed','UniqueProduct','IdenticalProducts','KnewInfoConfidential','InfoReverseEngineerable','InfoReverseEngineered']

laser =
['SecurityMeasures','OutsiderDisclosuresRestricted','KnewInfoConfidential','DisclosureInNegotations','SecretsDisclosedOutsiders']

lewis = ['CompetitiveAdvantage','KnewInfoConfidential','DisclosureInNegotations']

mbl =
['AgreedNotToDisclose','SecurityMeasures','NoncompetitionAgreement','AgreementNotSpecific','SecretsDisclosedOutsiders','InfoKnownToCompetitors']

mason =
['SecurityMeasures','UniqueProduct','KnewInfoConfidential','DisclosureInNegotations','InfoReverseEngineerable']

mineralDeposits =
['IdenticalProducts','RestrictedMaterialsUsed','DisclosureInNegotations','InfoReverseEngineerable','InfoReverseEngineered']

nationalInstruments = ['IdenticalProducts','KnewInfoConfidential','DisclosureInNegotations']

nationalRejectors =
['BroughtTools','UniqueProduct','IdenticalProducts','SecretsDisclosedOutsiders','InfoReverseEngineerable','NoSecurityMeasures','DisclosureInPublicForum']

reinforced =
['AgreedNotToDisclose','SecurityMeasures','CompetitiveAdvantage','UniqueProduct','KnewInfoConfidential','DisclosureInNegotations']

scientology =
['AgreedNotToDisclose','SecurityMeasures','OutsiderDisclosuresRestricted','SecretsDisclosedOutsiders','VerticalKnowledge','InfoKnownToCompetitors']

technicon =
['SecurityMeasures','OutsiderDisclosuresRestricted','RestrictedMaterialsUsed','KnewInfoConfidential','SecretsDisclosedOutsiders','InfoReverseEngineerable','InfoReverseEngineered']

trandes =
['AgreedNotToDisclose','SecurityMeasures','OutsiderDisclosuresRestricted','DisclosureInNegotations','SecretsDisclosedOutsiders']

valcoCincinnati =
['SecurityMeasures','OutsiderDisclosuresRestricted','UniqueProduct','KnewInfoConfidential','DisclosureInNegotations','SecretsDisclosedOutsiders']

Expected Outcome

arco = ['no trade secret was misappropriated, find for
defendant','LegitimatelyObtainable','EffortstoMaintainSecrecy','InfoKnownOrAvailiable','InfoKnown','InfoAvailiableElsewhere','InfoUsed', 'SecretsDisclosedOutsiders',
'InfoReverseEngineerable','InfoKnownToCompetitors']

boeing = ['a trade secret was misappropriated, find for
plaintiff','TradeSecretMisappropriation','EffortstoMaintainSecrecy','InfoValuable','ImproperMeans','QuestionableMeans','InfoUsed','ConfidentialRelationship','NoticeOfConfidentiality','ConfidentialityAgreement','MaintainSecrecyDefendant','MaintainSecrecyOutsiders',
'AgreedNotToDisclose','SecurityMeasures','OutsiderDisclosuresRestricted','RestrictedMaterialsUsed','KnewInfoConfidential','DisclosureInNegotations','SecretsDisclosedOutsiders']

bryce = ['a trade secret was misappropriated, find for
plaintiff','TradeSecretMisappropriation','EffortstoMaintainSecrecy','InfoValuable','InfoUsed','ConfidentialRelationship','NoticeOfConfidentiality','ConfidentialityAgreement','MaintainSecrecyDefendant',
'AgreedNotToDisclose','SecurityMeasures','IdenticalProducts','KnewInfoConfidential','DisclosureInNegotations']

collegeWatercolour = ['a trade secret was misappropriated, find for
plaintiff','TradeSecretMisappropriation','EffortstoMaintainSecrecy','InfoValuable','ImproperMeans','QuestionableMeans','InfoUsed',
'UniqueProduct','Deception','DisclosureInNegotations']

denTalEz = ['a trade secret was misappropriated, find for
plaintiff','TradeSecretMisappropriation','EffortstoMaintainSecrecy','InfoValuable','ImproperMeans','QuestionableMeans','InfoUsed','ConfidentialRelationship','NoticeOfConfidentiality','ConfidentialityAgreement','MaintainSecrecyDefendant',
'AgreedNotToDisclose','SecurityMeasures','KnewInfoConfidential','Deception','DisclosureInNegotations']

ecolgix = ['no trade secret was misappropriated, find for
defendant','InfoValuable','InfoUsed','ConfidentialRelationship','NoticeOfConfidentiality',
'KnewInfoConfidential','DisclosureInNegotations','NoSecurityMeasures','WaiverOfConfidentiality']

emery = ['a trade secret was misappropriated, find for
plaintiff','TradeSecretMisappropriation','EffortstoMaintainSecrecy','InfoValuable','InfoUsed','

ConfidentialRelationship','NoticeOfConfidentiality',
'IdenticalProducts','KnewInfoConfidential','SecretsDisclosedOutsiders']

ferranti = ['no trade secret was misappropriated, find for
defendant','LegitimatelyObtainable','InfoKnownOrAvailiable','InfoKnown',
'BribeEmployee','InfoIndependentlyGenerated','NoSecurityMeasures','InfoKnownToCompeti
tors','DisclosureInPublicForum']

robinson = ['no trade secret was misappropriated, find for
defendant','InfoValuable','ImproperMeans','QuestionableMeans','InfoUsed','IdenticalProducts'
,'Deception','DisclosureInNegotations','SecretsDisclosedOutsiders','NoSecurityMeasures']

sandlin = ['no trade secret was misappropriated, find for
defendant','LegitimatelyObtainable','InfoKnownOrAvailiable','InfoAvailiableElsewhere','Info
Used','DisclosureInNegotations','SecretsDisclosedOutsiders','InfoReverseEngineerable','NoSe
curityMeasures','DisclosureInPublicForum']

sheets = ['no trade secret was misappropriated, find for defendant','InfoValuable','InfoUsed',
'IdenticalProducts','NoSecurityMeasures','DisclosureInPublicForum']

spaceAero = ['no trade secret was misappropriated, find for
defendant','InfoValuable','InfoUsed',
'CompetitiveAdvantage','UniqueProduct','IdenticalProducts','DisclosureInNegotations','NoSec
urityMeasures']

televation = ['a trade secret was misappropriated, find for
plaintiff','TradeSecretMisappropriation','EffortstoMaintainSecrecy','InfoValuable','InfoUsed','
ConfidentialRelationship','NoticeOfConfidentiality','MaintainSecrecyOutsiders',
'SecurityMeasures','OutsiderDisclosuresRestricted','UniqueProduct','IdenticalProducts','Knew
InfoConfidential','SecretsDisclosedOutsiders','InfoReverseEngineerable']

yokana = ['no trade secret was misappropriated, find for
defendant','LegitimatelyObtainable','EffortstoMaintainSecrecy','InfoKnownOrAvailiable','Inf
oAvailiableElsewhere','InfoUsed',
'BroughtTools','SecretsDisclosedOutsiders','InfoReverseEngineerable','DisclosureInPublicFor
um']

cm1 = ['no trade secret was misappropriated, find for
defendant','LegitimatelyObtainable','EffortstoMaintainSecrecy','InfoKnownOrAvailiable','Inf
oKnown','InfoAvailiableElsewhere','ConfidentialRelationship','NoticeOfConfidentiality','Con
fidentialityAgreement','MaintainSecrecyDefendant',
'AgreedNotToDisclose','SecurityMeasures','InfoKnownToCompetitors','InfoIndependentlyGe
nerated','InfoReverseEngineerable','SecretsDisclosedOutsiders','DisclosureInPublicForum']

digitalDevelopment = ['a trade secret was misappropriated, find for
plaintiff','TradeSecretMisappropriation','EffortstoMaintainSecrecy','InfoValuable','InfoUsed','
ConfidentialRelationship','NoticeOfConfidentiality',
'SecurityMeasures','CompetitiveAdvantage','UniqueProduct','IdenticalProducts','KnewInfoCo
nfidential','DisclosureInNegotations']

fmc = ['a trade secret was misappropriated, find for
plaintiff','TradeSecretMisappropriation','EffortstoMaintainSecrecy','InfoValuable','InfoUsed','

ConfidentialRelationship','NoticeOfConfidentiality','ConfidentialityAgreement','MaintainSecrecyDefendant','MaintainSecrecyOutsiders','AgreedNotToDisclose','SecurityMeasures','BroughtTools','OutsiderDisclosuresRestricted','SecretsDisclosedOutsiders','VerticalKnowledge']

forrest = ['a trade secret was misappropriated, find for plaintiff','TradeSecretMisappropriation','EffortstoMaintainSecrecy','InfoValuable','InfoUsed','ConfidentialRelationship','NoticeOfConfidentiality', 'SecurityMeasures','UniqueProduct','KnewInfoConfidential','DisclosureInNegotations']

goldberg = ['a trade secret was misappropriated, find for plaintiff','TradeSecretMisappropriation','EffortstoMaintainSecrecy','InfoValuable','InfoUsed','ConfidentialRelationship','NoticeOfConfidentiality', 'KnewInfoConfidential','DisclosureInNegotations','SecretsDisclosedOutsiders','DisclosureInPublicForum']

kg = ['a trade secret was misappropriated, find for plaintiff','TradeSecretMisappropriation','EffortstoMaintainSecrecy','InfoValuable','InfoUsed','ConfidentialRelationship','NoticeOfConfidentiality', 'SecurityMeasures','RestrictedMaterialsUsed','UniqueProduct','IdenticalProducts','KnewInfoConfidential','InfoReverseEngineerable','InfoReverseEngineered']

laser = ['a trade secret was misappropriated, find for plaintiff','TradeSecretMisappropriation','EffortstoMaintainSecrecy','InfoValuable','InfoUsed','ConfidentialRelationship','NoticeOfConfidentiality','MaintainSecrecyOutsiders', 'SecurityMeasures','OutsiderDisclosuresRestricted','KnewInfoConfidential','DisclosureInNegotations','SecretsDisclosedOutsiders']

lewis = ['a trade secret was misappropriated, find for plaintiff','TradeSecretMisappropriation','EffortstoMaintainSecrecy','InfoValuable','InfoUsed','ConfidentialRelationship','NoticeOfConfidentiality', 'CompetitiveAdvantage','KnewInfoConfidential','DisclosureInNegotations']

mbl = ['no trade secret was misappropriated, find for defendant','LegitimatelyObtainable','EffortstoMaintainSecrecy','InfoKnownOrAvailiable','InfoKnown','InfoUsed','ConfidentialRelationship','NoticeOfConfidentiality','ConfidentialityAgreement','MaintainSecrecyDefendant', 'AgreedNotToDisclose','SecurityMeasures','NoncompetitionAgreement','AgreementNotSpecific','SecretsDisclosedOutsiders','InfoKnownToCompetitors']

mason = ['no trade secret was misappropriated, find for defendant','LegitimatelyObtainable','EffortstoMaintainSecrecy','InfoValuable','InfoKnownOrAvailiable','InfoAvailiableElsewhere','InfoUsed','ConfidentialRelationship','NoticeOfConfidentiality', 'SecurityMeasures','UniqueProduct','KnewInfoConfidential','DisclosureInNegotations','InfoReverseEngineerable']

mineralDeposits = ['a trade secret was misappropriated, find for plaintiff','TradeSecretMisappropriation','EffortstoMaintainSecrecy','InfoValuable','InfoUsed','ConfidentialRelationship','NoticeOfConfidentiality', 'IdenticalProducts','RestrictedMaterialsUsed','DisclosureInNegotations','InfoReverseEngineerable','InfoReverseEngineered']

nationalInstruments = ['a trade secret was misappropriated, find for plaintiff','TradeSecretMisappropriation','EffortstoMaintainSecrecy','InfoValuable','InfoUsed','ConfidentialRelationship','NoticeOfConfidentiality','IdenticalProducts','KnewInfoConfidential','DisclosureInNegotations']

nationalRejectors = ['no trade secret was misappropriated, find for defendant','InfoValuable','InfoUsed','BroughtTools','UniqueProduct','IdenticalProducts','SecretsDisclosedOutsiders','InfoReverseEngineerable','NoSecurityMeasures','DisclosureInPublicForum']

reinforced = ['a trade secret was misappropriated, find for plaintiff','TradeSecretMisappropriation','EffortstoMaintainSecrecy','InfoValuable','InfoUsed','ConfidentialRelationship','NoticeOfConfidentiality','ConfidentialityAgreement','MaintainSecrecyDefendant','AgreedNotToDisclose','SecurityMeasures','CompetitiveAdvantage','UniqueProduct','KnewInfoConfidential','DisclosureInNegotations']

scientology = ['no trade secret was misappropriated, find for defendant','LegitimatelyObtainable','EffortstoMaintainSecrecy','InfoKnownOrAvailiable','InfoKnown','InfoUsed','ConfidentialRelationship','NoticeOfConfidentiality','ConfidentialityAgreement','MaintainSecrecyDefendant','MaintainSecrecyOutsiders','AgreedNotToDisclose','SecurityMeasures','OutsiderDisclosuresRestricted','SecretsDisclosedOutsiders','VerticalKnowledge','InfoKnownToCompetitors']

technicon = ['a trade secret was misappropriated, find for plaintiff','TradeSecretMisappropriation','EffortstoMaintainSecrecy','InfoValuable','InfoUsed','ConfidentialRelationship','NoticeOfConfidentiality','MaintainSecrecyOutsiders','SecurityMeasures','OutsiderDisclosuresRestricted','RestrictedMaterialsUsed','KnewInfoConfidential','SecretsDisclosedOutsiders','InfoReverseEngineerable','InfoReverseEngineered']

trandes = ['a trade secret was misappropriated, find for plaintiff','TradeSecretMisappropriation','EffortstoMaintainSecrecy','InfoValuable','InfoUsed','ConfidentialRelationship','NoticeOfConfidentiality','ConfidentialityAgreement','MaintainSecrecyDefendant','MaintainSecrecyOutsiders','AgreedNotToDisclose','SecurityMeasures','OutsiderDisclosuresRestricted','DisclosureInNegotations','SecretsDisclosedOutsiders']

valcoCincinnati = ['a trade secret was misappropriated, find for plaintiff','TradeSecretMisappropriation','EffortstoMaintainSecrecy','InfoValuable','InfoUsed','ConfidentialRelationship','NoticeOfConfidentiality','MaintainSecrecyOutsiders','SecurityMeasures','OutsiderDisclosuresRestricted','UniqueProduct','KnewInfoConfidential','DisclosureInNegotations','SecretsDisclosedOutsiders']

Fourth Amendment

Coolidge v. New Hampshire (cvnh) excluded as described in §5.1.

Input

cvus =
['car','moving','public_informant','illegal_substance','all_parts','automobile_location','illegal_substance']

cvm = ['car','ft015w','moving','highway','inspection_regulation','robbery','all_parts']

cvd =
['car','public_view','boot','crashed','parked_on_highway','dwelling','inspection_regulation','murder','all_parts','garage'] #pass

sdvo =
['car','paper_bag','glovebox','parked','parking_lot','multiple_parking','illegal_substance','all_parts','automobile_location','illegal_substance'] #pass

usvc =
['car','foot_locker','police_station','boot','parked','parking_lot','public_informant','illegal_substance','car_trunk','police_station_location','illegal_substance','double_locked'] #pass

avs =
['car','goods_container','suitcase','airport','boot','moving','agent_officer','illegal_substance','car_trunk','illegal_substance','closed'] #pass

usvr =
['car','paper_bag','police_station','parked','parking_lot','public_informant','illegal_substance','all_parts','car_trunk','automobile_location','police_station_location','illegal_substance','money','closed'] #pass

cvc =
['mobile_home','paper_bag','near_court','motorhome','police_station','parked','driver_in','parking_lot','downtown','public_informant','the_public','illegal_substance','all_parts','police_station_location','automobile_location','illegal_substance','closed','cab','suitable_accomodation_space','bedroom','kitchen'] #pass

cva =
['car','paper_bag','police_station','boot','moving','highway','public_informant','illegal_substance','car_trunk','automobile_location','illegal_substance','closed'] #pass

Expected Outcome

cvus = ['warantless search did not violate the fourth amendment','car','moving','public_informant','illegal_substance','all_parts','automobile_location','illegal_substance','Exigency','RiskofLosingEvidence','SubjectToInspectionRegulation','Licence','ProbableCauseToSearchVehicle','LegalSearchScope','SearchPlace','WholeVehicle','UrgentReasonToSearch','PublicSafety','AuthorizedOriginOfProbableCause','Information','ExigencyWhenApproached','UrgentStatus','Mobile','Automobile']

cvm = ['warantless search did not violate the fourth amendment','car','ft015w','moving','highway','inspection_regulation','robbery','all_parts','RiskofLosingEvidence','SubjectToInspectionRegulation','Licence','ProbableCauseToSearchVehicle','LegalSearchScope','WholeVehicle','UrgentReasonToSearch','Crime','PublicSafety','Authori

zedOriginOfProbableCause','Procedure','ExigencyWhenApproached','PublicLocation','Urgent Status','Mobile','Automobile','Exigency']

cvd = ['warantless search did not violate the fourth amendment','car','public_view','boot','crashed','parked_on_highway','dwelling','inspection_regulation','murder','all_parts','garage','RiskofLosingEvidence','VisibilityOfItem','CanBeSeen','SubjectToInspectionRegulation','Licence','ProbableCauseToSearchVehicle','LegalSearchScope','WholeVehicle','UrgentReasonToSearch','Crime','PublicSafety','AuthorizedOriginOfProbableCause','Procedure','ExigencyWhenApproached','PublicParking','CapableToMove','Mobile','Automobile','Exigency']

sdvo = ['warantless search did not violate the fourth amendment','car','paper_bag','glovebox','parked','parking_lot','multiple_parking','illegal_substance','all_parts','automobile_location','illegal_substance','RiskofLosingEvidence','CannotBeSeen','SubjectToInspectionRegulation','Licence','ProbableCauseToSearchVehicle','LegalSearchScope','SearchPlace','WholeVehicle','UrgentReasonToSearch','PublicSafety','AuthorizedOriginOfProbableCause','Procedure','ExigencyWhenApproached','PublicParking','CapableToMove','Mobile','MovableContainer','Automobile','Exigency']

usvc = ['warantless search violates the fourth amendment','car','foot_locker','police_station','boot','parked','parking_lot','public_informant','illegal_substance','car_trunk','police_station_location','illegal_substance','double_locked','RiskofLosingEvidence','ProtectionType','CannotBeSeen','RestrictedArea','Licence','OnlyVehicleContainer','UrgentReasonToSearch','PublicSafety','AuthorizedOriginOfProbableCause','Information','ExigencyWhenApproached','PublicParking','CapableToMove','Mobile','LargeContainer','Automobile','Privacy']

avs = ['warantless search violates the fourth amendment','car','goods_container','suitcase','airport','boot','moving','agent_officer','illegal_substance','car_trunk','illegal_substance','closed','RiskofLosingEvidence','CannotBeSeen','RestrictedArea','Licence','OnlyVehicleContainer','UrgentReasonToSearch','PublicSafety','AuthorizedOriginOfProbableCause','Information','ExigencyWhenApproached','UrgentStatus','Mobile','MovableContainer','LargeContainer','Automobile','Privacy']

usvr = ['warantless search did not violate the fourth amendment','car','paper_bag','police_station','parked','parking_lot','public_informant','illegal_substance','all_parts','car_trunk','automobile_location','police_station_location','illegal_substance','money','closed','RiskofLosingEvidence','RestrictedArea','Licence','ProbableCauseToSearchVehicle','LegalSearchScope','SearchPlace','WholeVehicle','UrgentReasonToSearch','PublicSafety','AuthorizedOriginOfProbableCause','Information','ExigencyWhenApproached','PublicParking','CapableToMove','Mobile','MovableContainer','Automobile','Exigency']

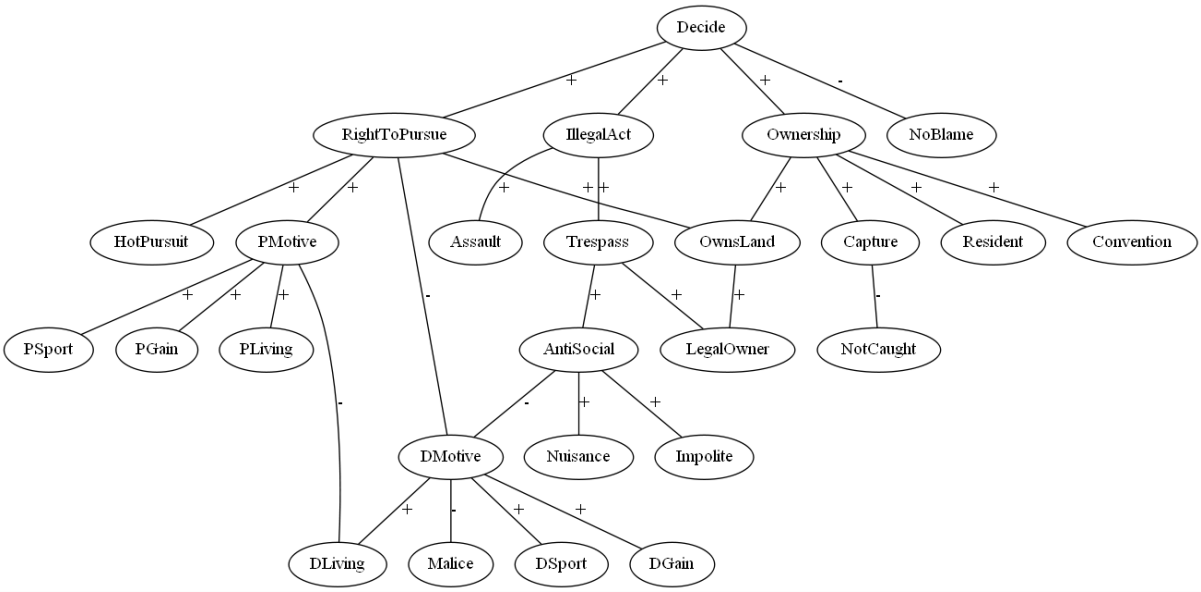
cvc = ['warantless search did not violate the fourth amendment','mobile_home','paper_bag','near_court','motorhome','police_station','parked','driver_in','parking_lot','downtown','public_informant','the_public','illegal_substance','all_parts','police_station_location','automobile_location','illegal_substance','closed','cab','suitable_accomodation_space','bedroom','kitchen','RiskofLosingEvidence','Accomodation','AccomodationSpaces','RestrictedArea','Licence','ProbableCauseToSearchVehicle','LegalSearchScope','SearchPlace','WholeVehicle','UrgentReasonToSearch','PublicSafety','AuthorizedOriginOfProbableCa

use','Information','ExigencyWhenApproached','PublicParking','PublicLocation','CapableToMove','Mobile','MovableContainer','Automobile','Exigency']

cva = ['warantless search violates the fourth amendment','car','paper_bag','police_station','boot','moving','highway','public_informant','illegal_substance','car_trunk','automobile_location','illegal_substance','closed','RiskofLosingEvidence','CannotBeSeen','RestrictedArea','Licence','SearchPlace','OnlyVehicleContainer','UrgentReasonToSearch','PublicSafety','AuthorizedOriginOfProbableCause','Information','ExigencyWhenApproached','PublicLocation','UrgentStatus','Mobile','MovableContainer','Automobile','Privacy']

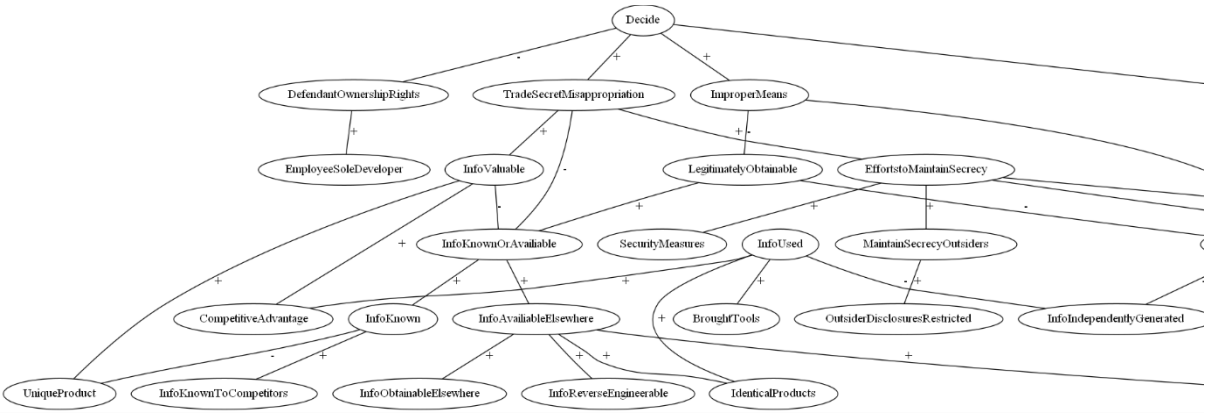
APPENDIX E - Visualisations

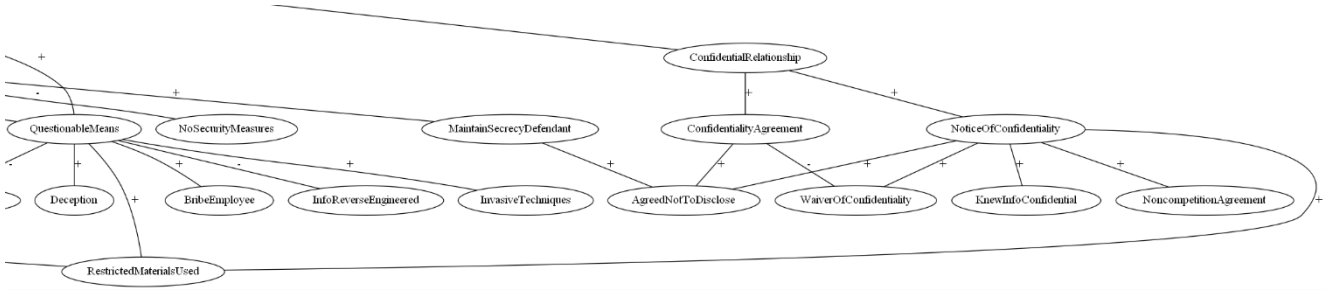
Wild Animals Domain



Trade Secrets Domain

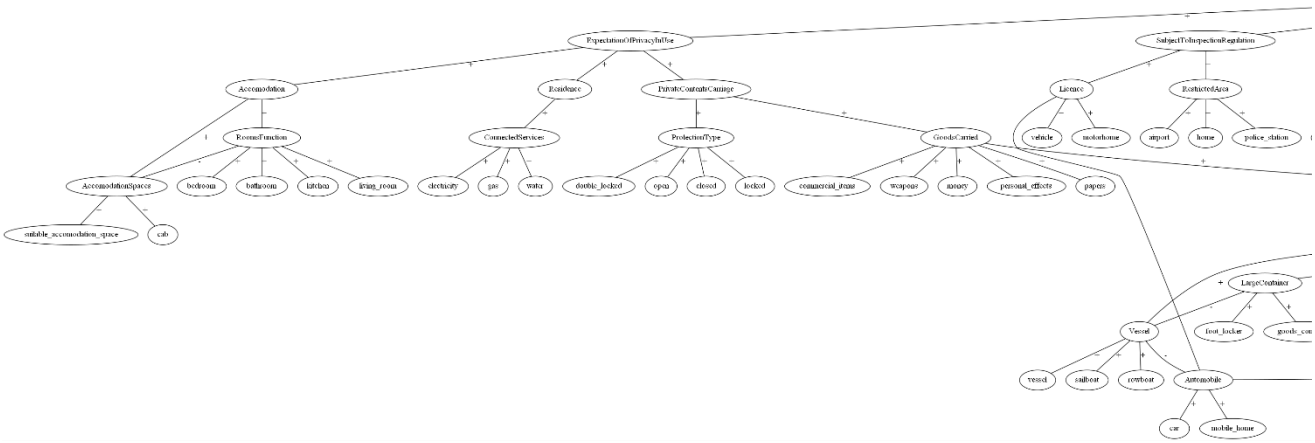
Domain split into two to be read from left to right.

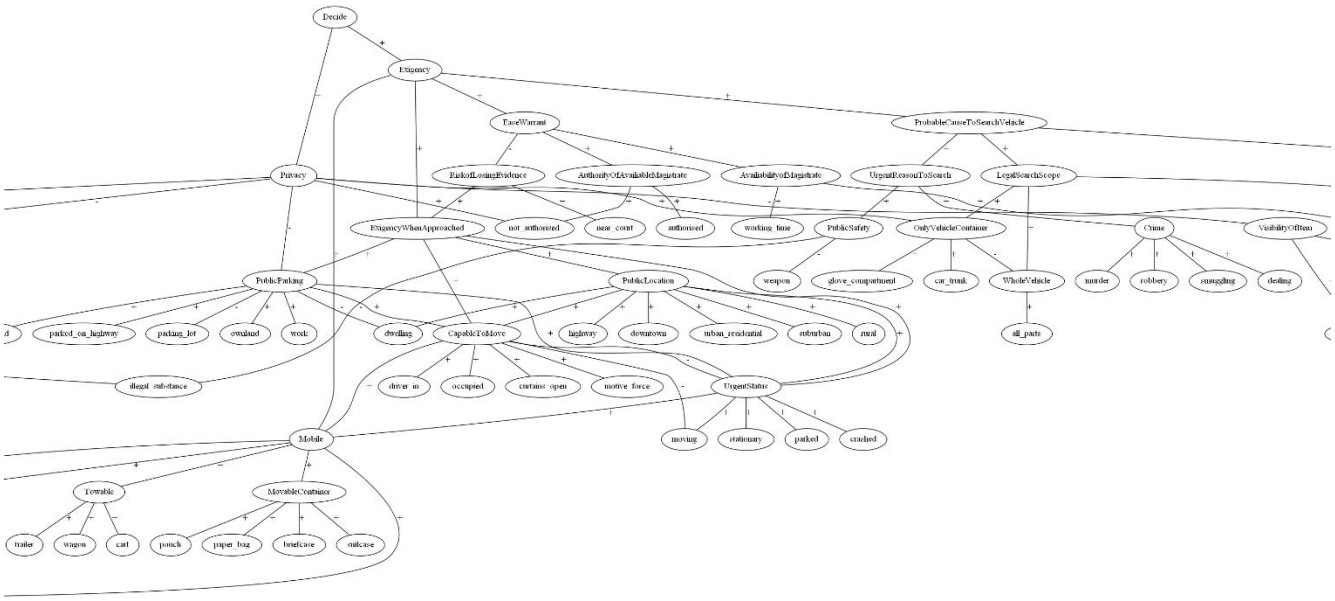


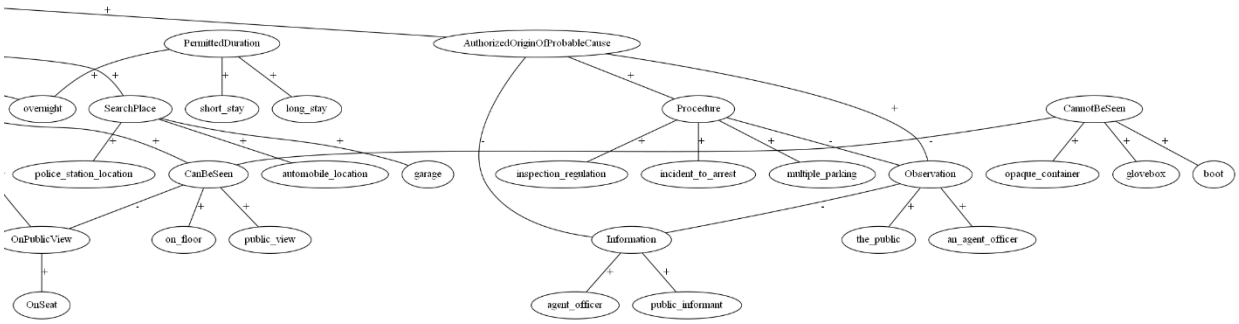


Automobile Exception Domain

Domain split into three to be read from left to right






PermittedDuration
AuthorizedOriginOfProbableCause
overnight
SearchPlace
short_stay
long_stay
Procedure
CannotBeSeen
police_station_location
CanBeSeen
automobile_location
garage
inspection_regulation
incident_to_arrest
multiple_parking
Observation
opaque_container
glovebox
boot
OnPublicView
on_floor
public_view
Information
the_public
an_agent_officer
OnSeat
agent_officer
public_informant

APPENDIX F – User Manual

##
USER MANUAL

##
CONTENTS

##
1.0 - GENERAL INSTRUCTIONS

- Pressing 'START' on the welcome screen will bring you to the main menu and allow you to use the application.

- You can go back to the menu at any time from 'File', but if you are in the middle of querying, creating or editing a domain, etc. your progress in this will be lost.

- You can also acces the user manual at any time by pressing 'Help'.

##
2.0 - CREATE A NEW DOMAIN

- After initialising the creation of a new legal domain you must first specify a name for the domain. This name cannot contain spaces in order to function properly.
e.g. if your domain name is Wild Animals, write it as WildAnimals

2.1 - Create a T/F Node

- Create this kind of node if you are creating a non-leaf factor and you want any questions to instantiate the base-level factors to be answered 'Yes' or 'No'.

- You do not need to create nodes for base-level factors, only non-leaf factors. The base-level factor nodes will be generated for you. The exception is if you

wish the base-level factor to be determined by multiple choice then refer to 2.2.

- First set a name for the node, the name must not contain any spaces. For the visualisation feature to work properly do not use names which are too long, under 25 characters is best.
e.g. if your node name is Visibility of Item, write it as VisibilityOfItem

- You can then set acceptance conditions and statements.

- Each acceptance condition must have an accompanying statement to be printed when it is accepted.

- Each acceptance condition must consist of other nodes, which may have been created previously or not, with the logical operators and, or, not between them.
e.g. Assault and not Trespass

- You may also use brackets in order to make the logical expressions clearer so long as there are spaces between the brackets and the nodes/logical conditions.
e.g. (OwnsLand and Resident) or Convention or Capture

- It is also possible to create a reject condition using the keyword 'reject'.
This keyword means that if the acceptance condition is satisfied the node will not be added to the factors in the case.
e.g. reject Licence and RestrictedArea

- There must always be a node named Decide in the domain which which may be created at any time. This node should give the final outcome of the case.

- Press add condition when you have entered your acceptance condition and statement. This will cause what you have entered to disappear and you can then enter another acceptance condition and statement if needed.

2.1.1 - Default Statement

- This statement will be displayed in the outcome if the acceptance condition is not met.

- Press add statement when you have entered this.

- Press END if you have added all the nodes otherwise add a T/F node or a multi node - 2.3 will detail the next screen.

2.2 - Create a Multiple Choice Node

- Create this kind of node if you are creating a base-level factor which you want to instantiate with a multiple choice question.
e.g. If the vehicle is an automobile, is it a: car, mobile home

- To refer to setting the name, acceptance and statement boxes refer to 2.1.

- The only exception to the above is the setting of the acceptance conditions. The tokens within the acceptance condition, which have not been created as other nodes, will form the answers of the multiple choice question. The choice made by the user will then determine whether the node is accepted or not.
e.g. weapon and illegal_substance

- When adding a token into an acceptance condition which you wish to be part of the multiple choice answer list, the convention is to use underscores instead of leaving no spaces as the program will render underscores as spaces when displaying the answers to the user which allows for easier comprehension.

- Set the question you wish to have answered by multiple choice.

2.3 - Question Creation

- After creating the domain, the program will determine which base-level factors require questions to be set. Set a question for each and then press add question.

- You can only add one question per base-level factors, repeated entry of questions will result in the previous question being overwritten.

2.3.1 - Question Order

- This screen enables the user to create an order for the questions to be presented to the user by highlighting a question and then pressing up or down to move it.

- Press DONE when this has been completed. This will bring you to the edit screen where you can create aditional nodes, delete erroneous nodes, change the question order and save the domain you have created. If you do not go through the save function all progress will be lost upon returning to the menu or exiting the program.

- For details on saving the domain refer to 3.4.2

##
3.0 - EXISTING DOMAIN

- This screen allows you to select one of the included domains: Wild Animals, Trade Secrets, Exception to the Fourth Amendment and Noise-Induced Hearing Loss or to import a domain you previously created and saved through this application. This file would be saved as a '.xlsx' file.

3.1 - Edit Domain Menu

- The first option allows you to query a case in your chosen legal domain.

- The second option allows you to edit a case in your chosen legal domain.

- The third option allows you to visualise your chosen legal domain.

3.2 - Query Domain

- On this screen you can query a predefined case from the drop down menu or query a new case by inputting a case name. Ensure the case name has no spaces between words.

3.2.1 - Question Answering

- For 'Yes' or 'No' questions, answering them is mandatory. Select your answer and then press NEXT for the next question.

- For multiple choice questions answering them is optional. If none of the answers fit your case then leave them blank and press NEXT.

3.3 - Outcome

- The outcome displays the result and reasoning which led to a decision being made.

- NEXT CASE - will allow you to query another case in the same domain.

- REPORT - will give you a more detailed report of the outcome of the case. As detailed in 3.3.1

3.3.1 - Report

- This screen states how many nodes are in the domain, how many have been accepted and a factor list with all the nodes which have been accepted.

- If you input a filename and press visualise this will create and open a png file which shows a tree graph of the domain with the accepted nodes highlighted green and the rejected nodes highlighted red. This shows the pathway the decision making tool used to reach its outcome.

3.4 - Edit Domain

- On this screen you can select a non-leaf node which you can delete by pressing DELETE or you can edit by pressing SEARCH.

- If you wish to edit a leaf node, you can only edit the question associated with it. If you wish to delete a leaf node you must do this by changing any parent node's acceptance conditions which make reference to this node to no longer reference the node.

- For information on CHANGE QUESTION ORDER, refer to 2.3.1

- For information on CREATE NODE, refer to 2.1 or 2.2

3.4.1 - Edit non-leaf node

- The same rules from 2.1 or 2.2 apply for changing the name, statements, questions and acceptance conditions.

- For a multi-node they will appear in both non-leaf and leaf lists. In the leaf list you will be able to edit the question and in the non-leaf list you will be able to edit the acceptance conditions and statements.

- Pressing ADD ACCEPTANCE will add a new acceptance condition and a statement to the screen.

- Press SUBMIT when you have made the required changes and then press DONE. Failure to press SUBMIT will result in the changes not being saved.

3.4.2 - Save

- The filename must have no spaces. Once entered pressing SAVE will create a '.xlsx' file with the data contained. When 'Done' shows this process has been completed. To query or edit the newly created domain, refer to 3.0.

3.5 - Visualise Domain

- This will open a tree graph visualisation of the domain and save this image as a png with the domain name as the filename.

APPENDIX G – Project Log

Project Milestones Achieved

- Completed the reading and background research required 24.06.2021
- Completed the full design 07.07.2021
- Finished implementing the backend systems 08.08.2021
- Completed the frontend systems 22.08.2021
- Completed the testing/evaluation 29.08.2021
- Finished the Final Report 03.09.2021

APPENDIX H – Code Listings

This appendix will reproduce the full code listings and will detail how to run the code. The code for the NIHL domain will not be listed here, due to data privacy concerns. More detail on this can be found in §2.

H.1 INSTALLATION AND RUN GUIDE

This code was tested on an Anaconda distribution of Python 3.8.8.

The additional libraries required to be installed are:

pip install pythonds

pip install openpyxl

conda install graphviz

pip install pydot

Tkinter should be part of the Anaconda distribution but otherwise use:

pip install tk

It is crucial that graphviz and pydot are installed in the order listed as there can sometimes be issues with the graphviz library otherwise, and pydot requires this library to generate the visualisations.

To run the application please ensure all '.py' files and the user_manual.txt are in the same directory and only run the UI.py file.

H.2 MainClasses.py

This file establishes the classes for the ADF(), Node() and MultiNode() classes. The file also contains the function for importing an ADF in importADF().

```
from openpyxl.reader.excel import load_workbook
from pythonds import Stack
import openpyxl as xl
import pydot

class ADF:
    """
    A class used to represent the ADF graph

    Attributes
    ----------
```

```
    name : str
        the name of the ADF
    nodes : dict
        the nodes which constitute the ADF
    reject : bool, default False
        is set to true when the reject keyword is used which lets the software know to reject the node ra-
ther than accep it when the condition is true
    nonLeaf : dict
        the nodes which are non-leaf that have children
    questionOrder : list
        an ordered list which determines which order the questions are asked in
    question : str, optional
        if the node is a base-level factor this stores the question
    statements : list
        the statements to be shown if the node is accepted or rejected
    nodeDone : list
        nodes which have been evaluated
    case : list
        the list of factors forming the case

    Methods
    -------
    addNodes(name, acceptance = None, statement=None, question=None)
        allows nodes to be added to the ADF from the Node() class
    addMulti(name, acceptance, statement, question)
        allows nodes to be added to the ADF from the MultiChoice() class
    nonLeafGen()
        determines what is a non-leaf factor
    evaluateTree(case)
        evaluates the ADF for a specified case
    evaluateNode(node)
        evaluates the acceptance conditions of the node
    postfixEvaluation(acceptance)
        evaluates the individual acceptance conditions which are in postfix notation
    checkCondition(operator, op1, op2 = None):
        checks the logical conditions for the acceptance condition, returning a boolean
    checkNonLeaf(node)
        checks if a node has children which need to be evaluated before it is evaluated
    questionAssignment()
        checks if any node requires a question to be assigned
    visualiseNetwork(case=None)
        allows visualisation of the ADF
    saveNew(name)
        allows the ADF to be saved as a .xlsx file
    saveHelper(wb,name)
        helper class for saveNew which provides core functionality
    """
```

```
def __init__(self, name):
    """
    Parameters
    ----------
    name : str
        the name of the ADF
    """

    self.name = name

    #dictionary of nodes --> 'name': 'node object
    self.nodes = {}

    self.reject = False

    #dictionary of nodes which have children
    self.nonLeaf = {}

    self.questionOrder = []

def addNodes(self, name, acceptance = None, statement=None, question=None):
    """
    adds nodes to ADF

    Parameters
    ----------
    name : str
        the name of the node
    acceptance : list
        a list of the acceptance conditions each of which should be a string
    statement : list
        a list of the statements which will be shown if a condition is accepted or rejected
    question : str
        the question to determine whether a node is absent or present

    """

    node = Node(name, acceptance, statement, question)

    self.nodes[name] = node

    self.question = question

    #creates children nodes
    if node.children != None:
        for childName in node.children:
            if childName not in self.nodes:
                node = Node(childName)
```

```
            self.nodes[childName] = node

    def addMulti(self,name, acceptance, statement, question):
        """
        adds MultiChoice() nodes to the ADF

        Parameters
        ----------
        name : str
            the name of the node
        acceptance : list
            a list of the acceptance conditions each of which should be a string
        statement : list
            a list of the statements which will be shown if a condition is accepted or rejected
        question : str
            the question to determine whether a node is absent or present

        """

        node = MultiChoice(name, acceptance, statement, question)

        self.nodes[name] = node

    def nonLeafGen(self):
        """
        determines which of the nodes is non-leaf
        """

        #sets it back to an empty dictionary
        self.nonLeaf = {}

        #checks each node and determines if it is a non-leaf node (one with children)
        for name,node in zip(self.nodes,self.nodes.values()):

            #adds node to dict of nodes with children
            if node.children != None and node.children != []:
                self.nonLeaf[name] = node
            else:
                pass

    def evaluateTree(self, case):
        """
        evaluates the ADF for a given case

        Parameters
        ----------
        case : list
            the list of factors forming the case
```

```
        """
        #keep track of print statements
        self.statements = []
        #list of non-leaf nodes which have been evaluated
        self.nodeDone = []
        self.case = case
        #generates the non-leaf nodes
        self.nonLeafGen()
        #while there are nonLeaf nodes which have not been evaluated, evaluate a node in this list in ascend-
ing order
        while self.nonLeaf != {}:
            for name,node in zip(self.nonLeaf,self.nonLeaf.values()):
                #checks if the node's children are non-leaf nodes
                if name == 'Decide' and len(self.nonLeaf) != 1:
                    pass
                elif self.checkNonLeaf(node):
                    #adds to list of evaluated nodes
                    self.nodeDone.append(name)
                    #checks candidate node's acceptance conditions
                    if self.evaluateNode(node):
                        #enables rejection clauses - handy for automobile
                        if self.reject != True:
                            #adds factor to case if present
                            self.case.append(name)
                        #deletes node from nonLeaf nodes
                        self.nonLeaf.pop(name)
                        self.statements.append(node.statement[self.counter])
                        self.reject = False
                        break
                    #if node's acceptance conditions are false
                    else:
                        #deletes node from nonLeaf nodes but doesn't add to case
                        self.nonLeaf.pop(name)
                        #the last statement is always the rejection statemenr
                        self.statements.append(node.statement[-1])
                        self.reject = False
                        break

        return self.statements

    def evaluateNode(self, node):
        """
        evaluates a node in respect to its acceptance conditions

        x will be always be a boolean value

        Parameters
```

```
        ----------
        node : class
            the node class to be evaluated

        """

        #for visualisation purposes - this tracks the attacking nodes
        self.vis = []

        #counter to index the statements to be shown to the user
        self.counter = -1

        #checks each acceptance condition seperately
        for i in node.acceptance:
            self.reject = False
            self.counter+=1
            x = self.postfixEvaluation(i)
            if x == True:
                return x

        return x

    def postfixEvaluation(self,acceptance):
        """
        evaluates the given acceptance condition

        Parameters
        ----------
        acceptance : str
            a string with the names of nodes seperated by logical operators

        """
        #initialises stack of operands
        operandStack = Stack()
        #list of tokens from acceptance conditions
        tokenList = acceptance.split()
        #checks each token's acceptance conditions
        for token in tokenList:
            #checks if something is a rejection condition
            if token == 'reject':
                self.reject = True
                try:
                    if x in self.case:
                        return True
                    else:
                        return False
                except:
                    pass
```

```
        elif token == 'not':
            operand1 = operandStack.pop()
            result = self.checkCondition(token,operand1)
            operandStack.push(result)
            self.vis.append(operand1)

        elif token == 'and' or token == 'or':
            operand2 = operandStack.pop()
            operand1 = operandStack.pop()
            result = self.checkCondition(token,operand1,operand2)
            operandStack.push(result)

        #for an acceptance condition with no operator
        elif len(tokenList) == 1 or (len(tokenList) ==2 and 'reject' in tokenList):
            if 'reject' in tokenList and 'reject' != token:
                x = token
            else:
                if token in self.case:
                    return True
                else:
                    return False
        else:
            #adds the operand to the stack
            operandStack.push(token)

    return operandStack.pop()

def checkCondition(self, operator, op1, op2 = None):
    """
    checks the logical condition and returns a boolean

    Parameters
    ----------
    operator : str
        the logical operator such as or, and, not
    op1 : str
        the first operand
    op2 : str, optional
        the second operand
    """

    if operator == "or":
        if op1 in self.case or op2 in self.case or op1 == True or op2 == True:
            return True
        else:
            return False

    elif operator == "and":
```

```
            if op1 == True or op1 in self.case:
                if op2 in self.case or op2 == True:
                    return True
                else:
                    return False
            elif op2 == True or op2 in self.case:
                if op1 in self.case or op1 == True:
                    return True
                else:
                    return False
            else:
                return False

        elif operator == "not":
            if op1 == True:
                return False
            if op1 == False:
                return True
            elif op1 not in self.case:
                return True
            else:
                return False

    def checkNonLeaf(self, node):
        """
        checks if a given node has children which need to be evaluated
        before it can be evaluated

        Parameters
        ----------
        node : class
            the node class to be evaluated

        """
        for j in node.children:

            if j in self.nonLeaf:

                if j in self.nodeDone:
                    pass

                else:
                    return False

            else:
                pass

        return True
```

```
def questionAssignment(self):
    """
    used by the user interface to determine whether a node needs a
    question assigning to it
    """
    for i in self.nodes.values():

        if i.children == None and i.question == None:

            return i.name

    return None

def visualiseNetwork(self,case=None):
    """
    allows the ADF to be visualised as a graph

    can be for the domain with or without a case

    if there is a case it will highlight the nodes green which have been
    accepted and red the ones which have been rejected

    Parameters
    ----------
    case : list, optional
        the list of factors constituting the case
    """

    #initialises the graph
    G = pydot.Dot('{}'.format(self.name), graph_type='graph')

    if case != None:

        #checks each node
        for i in self.nodes.values():

            #checks if node is already in the graph
            if i not in G.get_node_list():

                #checks if the node was accepted in the case
                if i.name in case:
                    a = pydot.Node(i.name,label=i.name,color='green')
                else:
                    a = pydot.Node(i.name,label=i.name,color='red')

                G.add_node(a)
```

```
        #creates edges between a node and its children
        if i.children != None and i.children != []:

            self.evaluateNode(i)

            for j in i.children:


                if j not in G.get_node_list():

                    if j in case:
                        a = pydot.Node(j,label=j,color='green')
                    else:
                        a = pydot.Node(j,label=j,color='red')

                    G.add_node(a)

                #self.vis is a list which tracks whether a node is an attacking or defending node
                if j in self.vis:
                    if j in case:
                        my_edge = pydot.Edge(i.name, j, color='green',label='-')
                    else:
                        my_edge = pydot.Edge(i.name, j, color='red',label='-')
                else:
                    if j in case:
                        my_edge = pydot.Edge(i.name, j, color='green',label='+')
                    else:
                        my_edge = pydot.Edge(i.name, j, color='red',label='+')

                G.add_edge(my_edge)

    else:

        #creates self.vis if not already created
        self.evaluateTree([])

        #checks each node
        for i in self.nodes.values():

            #checks if node is already in the graph
            if i not in G.get_node_list():

                a = pydot.Node(i.name,label=i.name,color='black')

                G.add_node(a)

            #creates edges between a node and its children
            if i.children != None and i.children != []:
```

```
            self.evaluateNode(i)

            for j in i.children:

                if j not in G.get_node_list():

                    a = pydot.Node(j,label=j,color='black')

                    G.add_node(a)

                #self.vis is a list which tracks whether a node is an attacking or defending node
                if j in self.vis:
                    my_edge = pydot.Edge(i.name, j, color='black',label='-')

                else:
                    my_edge = pydot.Edge(i.name, j, color='black',label='+')

                G.add_edge(my_edge)

    return G

def saveNew(self,filename):
    """
    enables an ADF to be saved as a .xlsx file

    Parameters
    ----------
    adf : class
        the instance of the ADF() class
    name : str
        the filename
    """

    #create the excel workbook
    wb = xl.Workbook()

    #saves the workbook
    wb.save('{}.xlsx'.format(filename))

    #saves the ADF in the workbook
    self.saveHelper(wb, filename)

def saveHelper(self,wb,filename):
    """
    helper class for saveNew which provides the ability to save the
    ADF in an excel file
```

```
    Parameters
    ----------
    adf : class
        the instance of the ADF() class
    name : str
        the filename
    wb : xl.Workbook()
        the excel workbook to save the adf in
    """

    #checks to see if there is a file with the same name to overwrite or not
    try:
        wb['{}'.format(filename)]
        ws = wb['{}'.format(filename)]

    except:
        wb.create_sheet('{}'.format(filename))
        ws = wb['{}'.format(filename)]

    for i in self.nodes.values():

        #duplicate flag
        dupeFlag = True

        #checks whether the node has already been saved or not to prevent duplicates
        for row in ws.iter_rows(values_only = True):
            if row[0] == i.name:
                dupeFlag = False
            else:
                pass

        if dupeFlag == True:

            #sets node name
            nodeName = [i.name]

            try:
                #adds acceptance conditions and corresponding statements
                for x,y in zip(i.acceptanceOriginal,i.statement):
                    nodeName.append(x)
                    nodeName.append(y)

                #adds the rejection statement
                nodeName.append(i.statement[-1])

            except:
                #no acceptance condition i.e. base level factor
                pass
```

```
            ws.append(nodeName)

            #adds the question in the row below the other node info
            if i.question != None:
                ws.append([i.question])

            else:
                ws.append(['None'])

        #inserts a row at the end with the question order
        if self.questionOrder != []:
            self.questionOrder.insert(0,"QUESTION!")
            ws.append(self.questionOrder)

        wb.save('{}.xlsx'.format(filename))

        #removes from the question order as only temporary for data file
        self.questionOrder.remove("QUESTION!")

class Node:
    """
    A class used to represent an individual node, whose acceptance conditions
    are instantiated by 'yes' or 'no' questions

    Attributes
    ----------
    name : str
        the name of the node
    question : str, optional
        the question which will instantiate the blf
    answers :
        set to None type to indicate to other methods the Node is not from MultiChoice()
    acceptanceOriginal : str
        the original acceptance condition before being converted to postfix notation
    statement : list
        the statements which will be output depending on whether the node is accepted or rejected
    acceptance : list
        the acceptance condition in postfix form
    children : list
        a list of the node's children nodes

    Methods
    -------
    attributes(acceptance)
        sets the acceptance conditions and determines the children nodes

    logicConverter(expression)
```

```
    converts the acceptance conditions into postfix notation

    """
    def __init__(self, name, acceptance=None, statement=None, question=None):
        """
        Parameters
        ----------
        name : str
            the name of the node
        statement : list, optional
            the statements which will be output depending on whether the node is accepted or rejected
        acceptance : list, optional
            the acceptance condition in postfix form
        question : str, optional
            the question which will instantiate the blf
        """
        #name of the node
        self.name = name

        #question for base leve factor
        self.question = question

        self.answers = None

        self.acceptanceOriginal = acceptance

        #sets postfix acceptance conditions and children nodes
        try:
            self.attributes(acceptance)
            self.statement = statement
        except:
            self.acceptance = None
            self.children = None
            self.statement = None

    def attributes(self, acceptance):
        """
        sets the acceptance condition and children for the node

        Parameters
        ----------
        acceptance : list
            the acceptance condition in postfix form
        """

        #sets acceptance condition to postfix if acceptance condition specified
        self.acceptance = []
        self.children = []
```

```
        for i in acceptance:
            self.acceptance.append(self.logicConverter(i))

        for i in self.acceptance:
            splitAcceptance = i.split()

            #sets the children nodes
            for token in splitAcceptance:

                if token not in ['and','or','not','reject'] and token not in self.children:

                    self.children.append(token)

    def logicConverter(self, expression):
        """
        converts a logical expression from infix to postfix notation

        Parameters
        ----------
        expression : list
            the acceptance condition to be converted into postfix form
        """

        #precedent dictionary of logical operators and reject keyword
        precedent = {'(':1,'or':2,'and':3,'not':4,'reject':5}

        #creates the stack
        operatorStack = Stack()

        #splits the tokens in the logical expression
        tokenList = expression.split()

        #stores the postfix expression
        postfixList = []

        #checks each token in the expression and pushes or pops on the stack accordingly
        for token in tokenList:

            if token == '(':
                operatorStack.push(token)
            elif token == ')':
                topToken = operatorStack.pop()
                while topToken != '(':
                    postfixList.append(topToken)
                    topToken = operatorStack.pop()
```

```
        elif token == 'and' or token == 'or' or token == 'not' or token == 'reject':
            while (not operatorStack.isEmpty()) and (precedent[operatorStack.peek()] >= precedent[token]):
                postfixList.append(operatorStack.pop())
            operatorStack.push(token)

        else:
            postfixList.append(token)

    #while operator stack not empty pop the operators to the postfix list
    while not operatorStack.isEmpty():
        postfixList.append(operatorStack.pop())

    #returns the post fix expression as a string
    return " ".join(postfixList)

class MultiChoice(Node):
    """
    for the creation of multiple choice base level factors, especially to
    facilitate the exception to the 4th amendment and NIHL domains

    Methods inherited from Node()

    Attributes
    ----------
    name : str
        the name of the node
    question : str, optional
        the question which will instantiate the blf
    answers : list
        the multiple choice answers to be selected from
    acceptanceOriginal : str
        the original acceptance condition before being converted to postfix notation
    statement : list
        the statements which will be output depending on whether the node is accepted or rejected
    acceptance : list
        the acceptance condition in postfix form
    children : list
        a list of the node's children nodes

    Methods
    -------
    attributes(acceptance)
        sets the acceptance conditions and determines the children nodes

    logicConverter(expression)
        converts the acceptance conditions into postfix notation

    """
```

```
    def __init__(self, name, acceptance, statement, question=None):
        """
        Parameters
        ----------
        name : str
            the name of the node
        statement : list, optional
            the statements which will be output depending on whether the node is accepted or rejected
        acceptance : list, optional
            the acceptance condition in postfix form
        question : str, optional
            the question which will instantiate the blf
        """

        #name of the node
        self.name = name

        #quetion for base level factor
        self.question = question

        self.acceptanceOriginal = acceptance

        #sets postfix acceptance conditions and children nodes
        try:
            self.attributes(acceptance)
            self.statement = statement
        except:
            self.acceptance = None
            self.children = None
            self.statement = None

        self.answers = self.children

def importADF(file,name):
    """
    enables a .xlsx file containing the ADF to be loaded into the tool

    Parameters
    ----------
    file : file
        the file address on the computer
    name : str
        the filename
    """

    try:

        #loads workbook
```

```
    wb = load_workbook(file)

    #creates ADF
    ws = wb[name]
    adf = ADF(name)

    #identifies question order row and converts to questionOrder attribute
    if ws[ws.max_row][0].value == 'QUESTION!':

        max_range = ws.max_row - 1
        adf.questionOrder = [x.value for x in ws[ws.max_row] if x.value is not None]
        adf.questionOrder.remove("QUESTION!")

    else:
        max_range = ws.max_row

    for row in range(1, max_range,2):

        #each node has two rows with its data
        firstrow = ws[row]
        secondrow = ws[row+1]

        row = [x.value for x in firstrow if x.value is not None]
        row1 = [x.value for x in secondrow if x.value is not None]

        nameNode = row[0]
        acceptance = row[1:-1:2]
        slice = (row[0::2])
        statement = slice[1:]
        if statement != []:
            statement.append(row[-1])
        question = row1[0]

        #checks whether a multiChoice() or Node() is more appropriate
        if row1[0] == 'None':
            adf.addNodes(nameNode,acceptance,statement)

        else:

            try:
                #if row[1] is present this indicates there is an acceptance condition so it is a multi-
ple choice node since there is a question
                row[1]
                adf.addMulti(nameNode,acceptance,statement,question)

            except:
                adf.addNodes(nameNode,acceptance,statement,question)
```

```
        return adf

    except IOError:
        print('\nFilename does not exist\n')
```

H.3 UI.py

This file initialises the user interface and is the only file the user should run to begin using the tool.

```
from tkinter import filedialog as fd
from tkinter import tix
from tkinter.constants import RIDGE
from tkinter.tix import *
import tkinter as tk
from tkinter import ttk
import openpyxl as xl
from MainClasses import *
import WildAnimals
import TradeSecrets
import FourthAmendment
import os



#excluded due to data privacy concerns
#import NIHL

#General Functionality and Menus
class Information():

    """
    info that is common between all classes which process the creation of an ADF
    """
    def __init__(self):

        self.name = None
        self.acceptance = []
        self.statement = []
        self.question = None
        self.adf = None
        self.multiFlag = False
        self.case = []
        self.node = None
        self.editFlag = False
        self.caseName = None
        self.caseList = {}
```

```
class UI(tix.Tk):

    def __init__(self,*args,**kwargs):

        tix.Tk.__init__(self,*args,**kwargs)
        tix.Tk.title(self,'ADF Tool')
        tix.Tk.geometry(self,"750x500")

        menubar = tk.Menu(self)
        filemenu = tk.Menu(menubar,tearoff=0)
        self.config(menu=menubar)

        filemenu.add_command(label="Menu",command=lambda:self.menu())
        filemenu.add_command(label="Exit", command=lambda:self.quit())
        menubar.add_cascade(label="File", menu=filemenu)
        menubar.add_command(label="Help", command=lambda:self.help())

        self.container = tk.Frame(self)
        self.container.pack(side="top",fill="both",expand=True)
        self.container.grid_rowconfigure(0,weight=1)
        self.container.grid_columnconfigure(0,weight=1)

        self.frames = {}
        frame = Welcome
        self.info = Information()
        self.frameCreation(frame,self.info)
        self.show_frame(frame)

        #styles
        s = ttk.Style()
        s.configure('my.TButton', font=('Helvetica', 15))

    def show_frame(self, page):
        frame = self.frames[page]
        frame.tkraise()

    def frameCreation(self, F, info):

        frame = F(self.container,self,info)

        self.frames[F] = frame
        frame.grid(row=0, column=0, sticky='nsew')

    def menu(self):
        self.frameCreation(MainMenu,self.info)
        self.show_frame(MainMenu)
```

```
    def help(self):
        os.startfile('User_Manual.txt')

    def quit(self):
        exit()

class Background(tk.Frame):

    def __init__(self,parent):
        tk.Frame.__init__(self,parent,bg='#A7D0FF')
        self.acc = []

class Welcome(Background):

    def __init__(self, parent, controller, info):

        Background.__init__(self,parent)

        self.info = info
        self.controller = controller

        var = tk.StringVar()
        var.set("Welcome to the Legal Decision-Making Tool")
        label = tk.Label(self, textvariable=var,font="Verdana 20 underline",bg='#A7D0FF')
        label.pack(pady=10,padx=10)

        T = tk.Text(self, height = 11, width = 45, bg='#ddeded',font="Verdana 18" )

        message = """This is a tool to help assist in legal decision-making.\n\nThis tool will predict the out-
come of a legal case, in a\nspecified legal domain, after asking the user a number\nof ques-
tions. You can also create new ADFs for legal\ndomains in this tool and edit exist-
ing ones in light\nof new precedents or do-
main knowledge.\n\nPress START to begin the tool or press MANUAL to\ncheck the user manual be-
fore you begin."""

        T.place(relx=0.05,rely=0.13)

        T.insert(tk.END, message )

        button = ttk.Button(self, text = 'START',style='my.TButton',command=lambda: self.start())
        button.place(relheight= 0.1, relwidth = 0.15, relx=0.6, rely=0.85)

        button2 = ttk.Button(self, text = 'MANUAL',style='my.TButton',command=lambda: self.manual())
        button2.place(relheight= 0.1, relwidth = 0.15, relx=0.2, rely=0.85)

    def start(self):
        self.controller.frameCreation(MainMenu, self.info)
        self.controller.show_frame(MainMenu)
```

```
    def manual(self):
        os.startfile('User_Manual.txt')

class MainMenu(Background):

    def __init__(self, parent, controller, info):

        Background.__init__(self,parent)

        self.controller = controller
        self.info = info
        self.info.case = []

        var = tk.StringVar()
        var.set("Main Menu")
        label = tk.Label(self, textvariable=var,font="Verdana 20 underline",bg='#A7D0FF')
        label.pack(pady=10,padx=10)
        button = ttk.Button(self, text = 'Existing domain',style='my.TButton',command=lambda: self.existing-
Domain())
        button.place(relheight= 0.25 , relwidth = 0.5, relx=0.25, rely=0.6)
        button2 = ttk.Button(self, text = 'Create a new domain',style='my.TButton',com-
mand=lambda: self.CreateDomain())
        button2.place(relheight= 0.25 , relwidth = 0.5, relx = 0.25, rely=0.2 )

    def CreateDomain(self):

        self.controller.frameCreation(CreateDomain, self.info)
        self.controller.show_frame(CreateDomain)

    def existingDomain(self):
        self.controller.frameCreation(ExistingDomain, self.info)
        self.controller.show_frame(ExistingDomain)

#Existing Domain Screens
class ExistingDomain(Background):
    def __init__(self,parent,controller, info):

        Background.__init__(self,parent)

        self.controller = controller
        self.info = info
        self.adf = None

        var = tk.StringVar()
        var.set("Existing Domain")
        var2 = tk.StringVar()
        var2.set("Select Domain:")
```

```
        label = tk.Label(self, textvariable=var,bg='#A7D0FF',font="Verdana 20 underline")
        label.pack()
        label2 = tk.Label(self, textvariable=var2,bg='#A7D0FF',font="Verdana 12")
        label2.place(relx = 0.02,rely = 0.2)

        var3 = tk.StringVar()
        var3.set("Import Domain:")
        label3 = tk.Label(self, textvariable=var3,bg='#A7D0FF',font="Verdana 12")
        label3.place(relx = 0.02,rely = 0.45)

        button = ttk.Button(self, text = 'OK',style= 'my.TButton',command= lambda: self.OK())
        button.place(relheight= 0.1 , relwidth = 0.1, relx = 0.45, rely=0.8 )

        help = tk.StringVar()
        help.set('?')

        helpLabel = tk.Label(self, textvariable=help,bg='#A7D0FF',fg='#FF001F', relief=RIDGE,font="Ver-
dana 12 bold")
        helpLabel.place(relx = 0.2,rely = 0.45)

        box = Balloon(self)

        box.bind_widget(helpLabel,balloonmsg="Please select a .xlsx file which has previously been cre-
ated through this app")

        button1 = ttk.Button(self, text = 'BROWSE',style= 'my.TButton',command= lambda: self.browse())
        button1.place(relheight= 0.1 , relwidth = 0.2, relx = 0.4, rely=0.55 )

        n = tk.StringVar()
        self.combo = ttk.Combobox(self, width = 50, textvariable = n, state='readonly')

        # Adding combobox drop down list
        self.combo['values'] = ('Wild Animals','Trade Secrets','Automobile Exception to the 4th Amend-
ment')#'Noise-Induced Hearing Loss')

        self.combo.place(relx = 0.3, rely=0.3)

    def OK(self):

        if self.adf != None:
            self.info.adf = self.adf
            self.info.caseList = {}
            self.controller.frameCreation(DomainMenu, self.info)
            self.controller.show_frame(DomainMenu)

        elif self.combo.get() == 'Wild Animals':
            self.info.adf = WildAnimals.adf()
```

```
            self.info.caseList = WildAnimals.cases()
            self.controller.frameCreation(DomainMenu, self.info)
            self.controller.show_frame(DomainMenu)

        elif self.combo.get() == 'Trade Secrets':
            self.info.adf = TradeSecrets.adf()
            self.info.caseList = TradeSecrets.cases()
            self.controller.frameCreation(DomainMenu, self.info)
            self.controller.show_frame(DomainMenu)

        elif self.combo.get() == 'Automobile Exception to the 4th Amendment':
            self.info.adf = FourthAmendment.adf()
            self.info.caseList = FourthAmendment.cases()
            self.controller.frameCreation(DomainMenu, self.info)
            self.controller.show_frame(DomainMenu)

        # elif self.combo.get() == 'Noise-Induced Hearing Loss':
        #     self.info.adf = NIHL.adf()
        #     self.info.caseList = NIHL.cases()
        #     self.controller.frameCreation(DomainMenu, self.info)
        #     self.controller.show_frame(DomainMenu)

        else:
            pass

    def browse(self):
        try:
            file = fd.askopenfilename(title='BROWSE',filetypes=[("Excel files", "*.xlsx")])
            name = os.path.basename(file)
            name = name.split('.')[0]
            self.adf = importADF(file,name)
            file.close()
        except:
            pass

class DomainMenu(Background):

    def __init__(self,parent,controller, info):

        Background.__init__(self,parent)

        self.controller = controller
        self.info = info

        button = ttk.Button(self, text = 'Edit domain',style='my.TButton', command=lambda:self.editDo-
main())
        button.place(relheight= 0.2 , relwidth = 0.5, relx=0.25, rely=0.4)
```

```
        button2 = ttk.Button(self, text = 'Query domain',style='my.TButton',command=lambda: self.queryDo-
main())
        button2.place(relheight= 0.2 , relwidth = 0.5, relx = 0.25, rely=0.15 )
        button3 = ttk.Button(self, text = 'Visualise domain',style='my.TButton',command=lambda: self.visual-
ise())
        button3.place(relheight= 0.2, relwidth = 0.5, relx = 0.25, rely=0.65 )

    def queryDomain(self):
        self.controller.frameCreation(CaseName, self.info)
        self.controller.show_frame(CaseName)

    def editDomain(self):
        self.controller.frameCreation(EditDomain, self.info)
        self.controller.show_frame(EditDomain)

    def visualise(self):
        self.graph = self.info.adf.visualiseNetwork()

        self.graph.write_png('{}.png'.format(self.info.adf.name))

        os.startfile('{}.png'.format(self.info.adf.name))

class CaseName(Background):
    def __init__(self,parent,controller, info):

        Background.__init__(self,parent)

        self.controller = controller
        self.info = info
        var = tk.StringVar()
        var.set("Query Domain")
        var2 = tk.StringVar()
        var2.set("Specify Case Name:")
        var3 = tk.StringVar()
        var3.set("Query a predefined case:")

        label = tk.Label(self, textvariable=var,bg='#A7D0FF',font="Verdana 20 underline")
        label.pack()
        label2 = tk.Label(self, textvariable=var2,bg='#A7D0FF',font="Verdana 12 underline")
        label2.place(relx = 0.02,rely = 0.2)
        label3 = tk.Label(self, textvariable=var3,bg='#A7D0FF',font="Verdana 12 underline")
        label3.place(relx = 0.02,rely = 0.5)

        n = tk.StringVar()
        self.combo = ttk.Combobox(self, width = 50, textvariable = n, state='readonly')
        if self.info.case != {}:
            self.combo['values'] = list(self.info.caseList.keys())
        self.combo.place(relx = 0.3 , rely = 0.62 )
```

```
        self.entry = tk.Entry(self,font='Verdana 12')
        self.entry.place(relx = 0.32,rely = 0.3,relheight=0.05,relwidth=0.35)
        button = ttk.Button(self, text = 'OK',style= 'my.TButton',command= lambda: self.OK())
        button.place(relheight= 0.1 , relwidth = 0.1, relx = 0.45, rely=0.8 )

    def OK(self):

        if self.entry.get() != '':
            self.info.caseName = self.entry.get()
            self.entry.delete(0,10000000)

            self.controller.frameCreation(QueryDomain, self.info)
            self.controller.show_frame(QueryDomain)

        elif self.combo.get() != '':
            self.info.caseName = self.combo.get()
            self.info.case = self.info.caseList[self.combo.get()]

            self.controller.frameCreation(Outcome, self.info)
            self.controller.show_frame(Outcome)


class QueryDomain(Background):
    def __init__(self,parent,controller, info):

        Background.__init__(self,parent)

        self.controller = controller
        self.info = info
        self.info.case = []

        self.number = 1

        self.stringDict = {}
        self.checkList = []

        self.nodes = self.info.adf.nodes.copy()

        self.questionOrder = self.info.adf.questionOrder.copy()
        self.orderFlag = False

        var = tk.StringVar()
        var.set("Domain Name:")

        var2 = tk.StringVar()
        var2.set(self.info.adf.name)
```

```
        self.var3 = tk.StringVar()
        self.var3.set("Question {}:".format(self.number))

        label = tk.Label(self, textvariable=var,bg='#A7D0FF',font="Verdana 20")
        label.pack(side = 'left',anchor='nw')

        label2 = tk.Label(self, textvariable=var2,bg='#A7D0FF',font="Verdana 20")
        label2.pack(side = 'left',anchor='nw')

        label3 = tk.Label(self, textvariable=self.var3,bg='#A7D0FF',font="Verdana 20")
        label3.place( relx = 0.05, rely=0.1)

        self.var4 = tk.StringVar()
        self.questionGen()

        label4 = tk.Label(self, textvariable=self.var4,bg='#A7D0FF',font="Verdana 20",wraplength=700)
        label4.place( relx = 0.05,rely=0.2)

        next = ttk.Button(self, text = 'NEXT',style= 'my.TButton',command= lambda: self.next())
        next.place(relheight= 0.1 , relwidth = 0.35, relx = 0.32, rely=0.85 )

        if self.multiFlag == False:
            self.trueFalse()

        else:
            self.multi()

    def next(self):
        if self.multiFlag == False:
            if self.i.get() == 'Yes':
                self.info.case.append(self.name)

            else:
                pass
        else:
            for key, value in self.stringDict.items():

                if value.get() == '1':
                    key = key.replace(' ','_')
                    self.info.case.append(key)

        self.forget()

        self.questionGen()

        if self.status == False:
            self.controller.frameCreation(Outcome, self.info)
            self.controller.show_frame(Outcome)
```

```
        else:
            pass

        if self.multiFlag:
            self.multi()
        else:
            self.trueFalse()

        self.number += 1
        self.var3.set("Question {}:".format(self.number))

    def questionGen(self):

        self.status = False
        self.multiFlag = False
        self.answers = []
        self.counter = 0

        if self.questionOrder != []:
            self.orderFlag = True

            for i in self.questionOrder:
                node = self.nodes[i]
                x = self.questionHelper(node)
                if x == 'Done':
                    break

            self.questionOrder.remove(i)
            self.name = i
            return

        elif self.orderFlag == False:
            for i in self.nodes.values():
                x = self.questionHelper(i)
                if x == 'Done':
                    break
            self.nodes.pop(i.name)
            self.name = i.name
            return

        return

    def questionHelper(self,i):
        #multipe choice question
        if i.answers != None:
            #prunes answers which are themselves non-leaf factors in the case of a multiple choice question
            for j in i.answers:
```

```
            try:
                self.info.adf.nodes[j]
                try:
                    #solves an issue in the fourth amendment domain with not_authorised sub-blf
                    if self.info.adf.nodes[j].question == '<>':
                        self.answers.append(j)
                except:
                    pass
            except:
                self.answers.append(j)

        question = i.question
        self.var4.set(question)
        self.multiFlag = True
        self.status = True
        return 'Done'

    #true/false question
    else:
        # '<>' is to fix a problem in the 4th amendment domain in which not_authorised was be-
ing prompted to set a question for it
        if i.question != None and i.question != '<>':
            self.question = i.question
            self.var4.set(self.question)
            self.status = True
            return 'Done'

def trueFalse(self):

    self.i = tk.StringVar()
    self.i.set("Yes")

    self.r1= tk.Radiobutton(self, text="Yes", value='Yes', variable =self.i)
    self.r1.place( relx = 0.45,rely=0.5)
    self.r2= tk.Radiobutton(self, text="No", value='No', variable=self.i)
    self.r2.place( relx = 0.45,rely=0.55)

def multi(self):

    x = 0.4
    y = 0.4

    counter = 0

    for i in self.answers:

        if len(self.answers)>5:
            x=0.2
```

```
            if len(self.checkList)>4:
                x=0.6
                y=0.4 + (0.05*counter)
                counter+=1

        i = i.replace('_',' ')
        y += 0.05
        self.stringDict[i] = tk.StringVar()
        c = tk.Checkbutton(self, text = i, variable=self.stringDict[i],tristatevalue=0)
        c.place(relx = x, rely = y)
        self.checkList.append(c)

    def forget(self):

        #multi check boxes
        try:
            # remove previous StringVars
            self.stringDict.clear()

            # remove previous Checkboxes
            for cb in self.checkList:
                cb.destroy()
            self.checkList.clear()

        except:
            pass

        try:
            self.r1.destroy()
            self.r2.destroy()
        except:
            pass

class Outcome(Background):

    def __init__(self,parent,controller, info):

        Background.__init__(self,parent)

        self.controller = controller
        self.info = info

        output = self.info.adf.evaluateTree(self.info.case)

        var = tk.StringVar()
        var.set("Domain Name:")
```

```
        var2 = tk.StringVar()
        var2.set(self.info.adf.name)

        self.var3 = tk.StringVar()
        self.var3.set("Outcome in {}:".format(self.info.caseName))

        label = tk.Label(self, textvariable=var,bg='#A7D0FF',font="Verdana 20")
        label.pack(side = 'left',anchor='nw')

        label2 = tk.Label(self, textvariable=var2,bg='#A7D0FF',font="Verdana 20")
        label2.pack(side = 'left',anchor='nw')

        label3 = tk.Label(self, textvariable=self.var3,bg='#A7D0FF',font="Verdana 20")
        label3.place( relx = 0, rely=0.1)

        anotherCase = ttk.Button(self, text = 'NEXT CASE',style= 'my.TButton',command= lambda: self.an-
otherCase())
        anotherCase.place(relheight= 0.1 , relwidth = 0.2, relx = 0.2, rely=0.85 )

        menu = ttk.Button(self, text = 'MENU',style= 'my.TButton',command= lambda: self.menu())
        menu.place(relheight= 0.1 , relwidth = 0.2, relx = 0.4, rely=0.85 )

        report = ttk.Button(self, text = 'REPORT',style= 'my.TButton',command= lambda: self.report())
        report.place(relheight= 0.1 , relwidth = 0.2, relx = 0.6, rely=0.85 )

        outcome = tk.Text(self)
        outcome.place(relheight=0.6,relwidth=0.8, relx = 0.1, rely=0.2)

        scrollbar = tk.Scrollbar(self)
        scrollbar.place(relheight=0.6,relwidth=0.02,relx=0.9,rely=0.2)
        scrollbar.config(command=outcome.yview)
        outcome.config(yscrollcommand=scrollbar.set)

        counter = 1

        for i in range(0,len(output)-1):
            x = output[i]

            outcome.insert(tk.END,"Reason {}: ".format(counter))
            outcome.insert(tk.END,x+'\n' )
            counter += 1

        outcome.insert(tk.END,"Outcome: ")
        outcome.insert(tk.END,output[-1])

    def anotherCase(self):
        self.controller.frameCreation(CaseName, self.info)
        self.controller.show_frame(CaseName)
```

```
    def menu(self):
        self.controller.show_frame(MainMenu)

    def report(self):
        self.controller.frameCreation(Report, self.info)
        self.controller.show_frame(Report)

class EditDomain(Background):
    def __init__(self,parent,controller, info):

        Background.__init__(self,parent)

        self.controller = controller
        self.info = info

        var = tk.StringVar()
        var.set("Domain Name:")

        var2 = tk.StringVar()
        var2.set(self.info.adf.name)

        self.var3 = tk.StringVar()
        self.var3.set("Edit Node")

        label = tk.Label(self, textvariable=var,bg='#A7D0FF',font="Verdana 20")
        label.pack(side = 'left',anchor='nw')

        label2 = tk.Label(self, textvariable=var2,bg='#A7D0FF',font="Verdana 20")
        label2.pack(side = 'left',anchor='nw')

        label3 = tk.Label(self, textvariable=self.var3,bg='#A7D0FF',font="Verdana 20 underline")
        label3.place( relx = 0.05, rely=0.1)

        var4 = tk.StringVar()
        var4.set('Edit or delete a non-leaf node')

        name = tk.Label(self, textvariable=var4,bg='#A7D0FF',font="Verdana 20 ")
        name.place( relx = 0.05, rely=0.2 )

        var5 = tk.StringVar()
        var5.set('Edit the question of a leaf node')

        question = tk.Label(self, textvariable=var5,bg='#A7D0FF',font="Verdana 20 ")
        question.place( relx = 0.05, rely=0.4 )

        n = tk.StringVar()
        self.combo = ttk.Combobox(self, width = 50, textvariable = n, state='readonly')
```

```
        m = tk.StringVar()
        self.combo2 = ttk.Combobox(self, width = 50, textvariable = m, state='readonly')

        nodeList = []
        nodeList2 = []

        self.info.adf.nonLeafGen()

        # Adding combobox drop down list
        for key in self.info.adf.nonLeaf:
            nodeList.append(key)

        for name in self.info.adf.nodes:
            node = self.info.adf.nodes[name]
            if node.question != None:
                nodeList2.append(node.name)

        self.combo['values'] = nodeList
        self.combo.place(relx = 0.3, rely=0.32)

        self.combo2['values'] = nodeList2
        self.combo2.place(relx = 0.3, rely=0.52)

        create = ttk.Button(self, text = 'CREATE NODE',style= 'my.TButton',command= lambda: self.create-
Node())
        create.place(relheight= 0.1 , relwidth = 0.4, relx = 0.1, rely=0.65 )

        search = ttk.Button(self, text = 'SEARCH',style= 'my.TButton',command= lambda: self.searchNode())
        search.place(relheight= 0.1 , relwidth = 0.4, relx = 0.3, rely=0.85 )

        save = ttk.Button(self, text = 'SAVE',style= 'my.TButton',command= lambda: self.saveNode())
        save.place(relheight= 0.1 , relwidth = 0.4, relx = 0.5, rely=0.75 )

        question = ttk.Button(self, text = 'CHANGE QUESTION ORDER',style= 'my.TButton',com-
mand= lambda: self.questionOrder())
        question.place(relheight= 0.1 , relwidth = 0.4, relx = 0.5, rely=0.65 )

        delete = ttk.Button(self, text = 'DELETE',style= 'my.TButton',command= lambda: self.delete())
        delete.place(relheight= 0.1 , relwidth = 0.4, relx = 0.1, rely=0.75 )


    def delete(self):

        if self.combo.get() != '':

            self.info.adf.nodes.pop(self.combo.get())
```

```
            self.combo.set('')
            self.combo['values'] = []

            nodeList = []

            self.info.adf.nonLeafGen()

            # Adding combobox drop down list
            for key in self.info.adf.nonLeaf:
                nodeList.append(key)

            self.combo['values'] = nodeList

    def searchNode(self):

        if self.combo.get() != '':
            node = self.info.adf.nodes[self.combo.get()]
            self.info.node = node
            self.controller.frameCreation(EditNode, self.info)
            self.controller.show_frame(EditNode)
        elif self.combo2.get() != '':
            node = self.info.adf.nodes[self.combo2.get()]
            self.info.node = node
            self.controller.frameCreation(EditQuestion, self.info)
            self.controller.show_frame(EditQuestion)

    def questionOrder(self):
        self.controller.frameCreation(QuestionOrder, self.info)
        self.controller.show_frame(QuestionOrder)

    def createNode(self):
        self.info.editFlag = True
        self.controller.frameCreation(NodeSelection, self.info)
        self.controller.show_frame(NodeSelection)

    def saveNode(self):
        self.controller.frameCreation(SaveScreen, self.info)
        self.controller.show_frame(SaveScreen)

class EditNode(Background):

    def __init__(self,parent,controller, info):

        Background.__init__(self,parent)

        self.controller = controller
        self.info = info
```

```
        var = tk.StringVar()
        var.set("Domain Name:")

        var2 = tk.StringVar()
        var2.set(self.info.adf.name)

        self.var3 = tk.StringVar()
        self.var3.set("Node:")

        self.var4 = tk.StringVar()
        self.var4.set("{}".format(self.info.node.name))

        label = tk.Label(self, textvariable=var,bg='#A7D0FF',font="Verdana 20")
        label.pack(side = 'left',anchor='nw')

        label2 = tk.Label(self, textvariable=var2,bg='#A7D0FF',font="Verdana 20")
        label2.pack(side = 'left',anchor='nw')

        label3 = tk.Label(self, textvariable=self.var3,bg='#A7D0FF',font="Verdana 20 ")
        label3.place( relx = 0, rely=0.1)

        label4 = tk.Label(self, textvariable=self.var4,bg='#A7D0FF',font="Verdana 20 ")
        label4.place( relx = 0.15, rely=0.1)

        save = ttk.Button(self, text = 'SUBMIT',style= 'my.TButton',command= lambda: self.submit())
        save.place(relheight= 0.1 , relwidth = 0.3, relx = 0.35, rely=0.75 )

        done = ttk.Button(self, text = 'DONE',style= 'my.TButton',command= lambda: self.done())
        done.place(relheight= 0.1 , relwidth = 0.3, relx = 0.35, rely=0.85 )

        back = ttk.Button(self, text = 'BACK',style= 'my.TButton',command= lambda: self.back())
        back.place(relheight= 0.1 , relwidth = 0.3, relx = 0.02, rely=0.75 )

        addAcceptance = ttk.Button(self, text = 'ADD ACCEPTANCE',style= 'my.TButton',com-
mand= lambda: self.accept())
        addAcceptance.place(relheight= 0.1 , relwidth = 0.3, relx = 0.66, rely=0.75 )

        self.acceptDict = {}
        self.statementDict = {}
        self.acceptList = []
        self.statementList = []

        nameVar = tk.StringVar()
        nameVar.set("Name:")
        name = tk.Label(self, textvariable=nameVar,bg='#A7D0FF',font="Verdana 12 ")
        name.place(relx = 0, rely=0.2)

        self.nameEntry = tk.Entry(self,font='Verdana 12')
```

```
        self.nameEntry.place(relx = 0.2,rely = 0.2,relheight=0.05,relwidth=0.75)
        self.nameEntry.insert(tk.END, self.info.node.name)

        self.y=0.2

        #ACCEPTANCE CONDITIONS
        counter = 1

        for i in self.info.node.acceptanceOriginal:
            self.y+=0.05

            self.acceptDict[i] = tk.StringVar()
            self.acceptDict[i].set("Acceptance {}:".format(counter))

            labelAccept = tk.Label(self, textvariable=self.acceptDict[i],bg='#A7D0FF',font="Verdana 12")
            labelAccept.place(x=0.05, rely=self.y)

            acceptance = tk.Entry(self,font='Verdana 12')
            acceptance.place(relx = 0.2,rely = self.y,relheight=0.05,relwidth=0.75)
            acceptance.insert(tk.END, i)

            self.acceptList.append(acceptance)

            counter += 1

        #STATEMENTS
        counter = 1

        for j in self.info.node.statement:
            self.y+=0.05

            self.statementDict[j] = tk.StringVar()
            self.statementDict[j].set("Statement {}:".format(counter))

            labelStatement = tk.Label(self, textvariable=self.statementDict[j],bg='#A7D0FF',font="Ver-
dana 12")
            labelStatement.place(x=0.05, rely=self.y)

            statement = tk.Entry(self,font='Verdana 12')
            statement.place(relx = 0.2,rely = self.y,relheight=0.05,relwidth=0.75)
            statement.insert(tk.END, j)

            self.statementList.append(statement)

            counter+= 1

    def accept(self):
        self.y+=0.05
```

```
        stringVar = tk.StringVar()
        stringVar.set("Acceptance:")

        labelAccept = tk.Label(self, textvariable=stringVar,bg='#A7D0FF',font="Verdana 12")
        labelAccept.place(x=0.05, rely=self.y)

        acceptance = tk.Entry(self,font='Verdana 12')
        acceptance.place(relx = 0.2,rely = self.y,relheight=0.05,relwidth=0.75)

        #adds the entry box to the list of acceptance conditions
        self.acceptList.append(acceptance)

        self.y += 0.05

        stringVar2 = tk.StringVar()
        stringVar2.set("Statement:")

        labelStatement = tk.Label(self, textvariable=stringVar2,bg='#A7D0FF',font="Verdana 12")
        labelStatement.place(x=0.05, rely=self.y)

        statement = tk.Entry(self,font='Verdana 12')
        statement.place(relx = 0.2,rely = self.y,relheight=0.05,relwidth=0.75)
        self.statementList.append(statement)

    def submit(self):

        self.acceptance = []
        self.statements = []

        self.name = self.nameEntry.get()
        self.nameEntry.delete(0,10000000)

        for i in self.acceptList:
            self.acceptance.append(i.get())
            i.delete(0,10000000)

        for j in self.statementList:
            self.statements.append(j.get())
            j.delete(0,10000000)

        self.info.adf.nodes.pop(self.info.node.name)
        self.info.adf.addNodes(self.name, self.acceptance, self.statements)

    def back(self):

        self.controller.frameCreation(EditDomain, self.info)
        self.controller.show_frame(EditDomain)
```

```
  def done(self):

      questionGeneration = False

      for i in self.info.adf.nodes.values():

          if i.children == None and i.question == None:
              questionGeneration = True

      if questionGeneration == True:
          self.controller.frameCreation(QuestionCreation, self.info)
          self.controller.show_frame(QuestionCreation)
      else:
          self.controller.frameCreation(EditDomain, self.info)
          self.controller.show_frame(EditDomain)

class EditQuestion(Background):

  def __init__(self,parent,controller, info):

      Background.__init__(self,parent)

      self.controller = controller
      self.info = info

      var = tk.StringVar()
      var.set("Domain Name:")

      var2 = tk.StringVar()
      var2.set(self.info.adf.name)

      self.var3 = tk.StringVar()
      self.var3.set("Node:")

      self.var4 = tk.StringVar()
      self.var4.set("{}".format(self.info.node.name))

      label = tk.Label(self, textvariable=var,bg='#A7D0FF',font="Verdana 20")
      label.pack(side = 'left',anchor='nw')

      label2 = tk.Label(self, textvariable=var2,bg='#A7D0FF',font="Verdana 20")
      label2.pack(side = 'left',anchor='nw')

      label3 = tk.Label(self, textvariable=self.var3,bg='#A7D0FF',font="Verdana 20 ")
      label3.place( relx = 0, rely=0.1)

      label4 = tk.Label(self, textvariable=self.var4,bg='#A7D0FF',font="Verdana 20 ")
```

```
        label4.place( relx = 0.15, rely=0.1)

        save = ttk.Button(self, text = 'SUBMIT',style= 'my.TButton',command= lambda: self.submit())
        save.place(relheight= 0.1 , relwidth = 0.35, relx = 0.32, rely=0.65 )

        done = ttk.Button(self, text = 'DONE',style= 'my.TButton',command= lambda: self.done())
        done.place(relheight= 0.1 , relwidth = 0.35, relx = 0.32, rely=0.75 )

        back = ttk.Button(self, text = 'BACK',style= 'my.TButton',command= lambda: self.back())
        back.place(relheight= 0.1 , relwidth = 0.35, relx = 0.32, rely=0.85 )

        questionVar = tk.StringVar()
        questionVar.set("Question:")
        question = tk.Label(self, textvariable=questionVar,bg='#A7D0FF',font="Verdana 12 ")
        question.place(relx=0.01,rely=0.2)

        self.questionEntry = tk.Entry(self,font='Verdana 12')
        self.questionEntry.place(relx=0.2,rely=0.2,relheight=0.05,relwidth=0.75)
        try:
            self.questionEntry.insert(tk.END, self.info.node.question)
        except:
            pass

    def submit(self):

        self.question = self.questionEntry.get()
        self.info.adf.nodes[self.info.node.name].question = self.question
        self.questionEntry.delete(0,10000000)

    def back(self):

        self.controller.frameCreation(EditDomain, self.info)
        self.controller.show_frame(EditDomain)

    def done(self):
        self.controller.frameCreation(SaveScreen, self.info)
        self.controller.show_frame(SaveScreen)

#Create Domain Screens
class CreateDomain(Background):
    def __init__(self,parent,controller, info):

        Background.__init__(self,parent)

        self.controller = controller
        self.info = info
        var = tk.StringVar()
        var.set("Create Domain")
```

```
        var2 = tk.StringVar()
        var2.set("Specify Domain Name:")
        label = tk.Label(self, textvariable=var,bg='#A7D0FF',font="Verdana 20 underline")
        label.pack()
        label2 = tk.Label(self, textvariable=var2,bg='#A7D0FF',font="Verdana 12 underline")
        label2.place(relx = 0.02,rely = 0.2)

        help = tk.StringVar()
        help.set('?')

        helpLabel = tk.Label(self, textvariable=help,bg='#A7D0FF',fg='#FF001F',relief=RIDGE,font="Ver-
dana 12 bold")
        helpLabel.place(relx = 0.3,rely = 0.2)

        box = Balloon(self)

        box.bind_widget(helpLabel,balloonmsg="Please ensure your domain name has no spaces be-
tween words")



        self.entry = tk.Entry(self,font='Calibri 20')
        self.entry.place(relx = 0.32,rely = 0.35,relheight=0.1,relwidth=0.35)
        button = ttk.Button(self, text = 'OK',style= 'my.TButton',command= lambda: self.OK())
        button.place(relheight= 0.1 , relwidth = 0.1, relx = 0.45, rely=0.8 )

    def OK(self):

        self.info.adf = ADF(self.entry.get())
        self.entry.delete(0,10000000)

        self.controller.frameCreation(NodeSelection, self.info)
        self.controller.show_frame(NodeSelection)

class NodeSelection(Background):

    def __init__(self,parent,controller, info):

        Background.__init__(self,parent)

        self.controller = controller
        self.info = info

        var = tk.StringVar()
        var.set("Node Creation")
        label = tk.Label(self, textvariable=var,font="Verdana 20 underline",bg='#A7D0FF')
        label.pack(pady=10,padx=10)

        help = tk.StringVar()
```

```
        help.set('?')

        helpLabel = tk.Label(self, textvariable=help,bg='#A7D0FF',fg='#FF001F',relief=RIDGE,font="Ver-
dana 20 bold")
        helpLabel.place(relx = 0.8,rely = 0.3)

        helpLabel1 = tk.Label(self, textvariable=help,bg='#A7D0FF',fg='#FF001F',relief=RIDGE,font="Ver-
dana 20 bold")
        helpLabel1.place(relx = 0.8,rely = 0.7)

        box = Balloon(self)

        box.bind_widget(helpLabel,balloonmsg="Create this type of node if you are creating non-leaf nodes")
        box.bind_widget(helpLabel1,balloonmsg="Create this type of node if you are creat-
ing a base level factor with multiple sub-factors")

        button = ttk.Button(self, text = 'Create a T/F Node',style='my.TButton',com-
mand = lambda: self.standardNode() )
        button.place(relheight= 0.25 , relwidth = 0.5, relx=0.25, rely=0.2 )
        button2 = ttk.Button(self, text = 'Create a Multiple Choice Node',style='my.TButton',com-
mand=lambda: self.multiNode())
        button2.place(relheight= 0.25 , relwidth = 0.5, relx = 0.25, rely=0.6 )



    def standardNode(self):
        self.controller.frameCreation(NodeCreation, self.info)
        self.controller.show_frame(NodeCreation)

    def multiNode(self):
        self.controller.frameCreation(MultiCreation, self.info)
        self.controller.show_frame(MultiCreation)

class NodeCreation(Background):

    def __init__(self,parent,controller, info):

        Background.__init__(self,parent)

        self.name = ''
        self.acceptance = []
        self.statement = []

        self.controller = controller
        self.info = info

        var = tk.StringVar()
        var.set("Domain Name:")
```

```
        var2 = tk.StringVar()
        var2.set(self.info.adf.name)

        var3 = tk.StringVar()
        var3.set('Create Node')

        var4 = tk.StringVar()
        var4.set('Name:')

        var5 = tk.StringVar()
        var5.set('Acceptance:')

        var6 = tk.StringVar()
        var6.set('Statement:')

        label = tk.Label(self, textvariable=var,bg='#A7D0FF',font="Verdana 20")
        label.pack(side = 'left',anchor='nw')

        label2 = tk.Label(self, textvariable=var2,bg='#A7D0FF',font="Verdana 20")
        label2.pack(side = 'left',anchor='nw')

        label3 = tk.Label(self, textvariable=var3,bg='#A7D0FF',font="Verdana 20 underline")
        label3.place( relx = 0, rely=0.1 )

        name = tk.Label(self, textvariable=var4,bg='#A7D0FF',font="Verdana 20 ")
        name.place( relx = 0, rely=0.2 )

        help = tk.StringVar()
        help.set('?')

        helpLabel = tk.Label(self, textvariable=help,bg='#A7D0FF',fg='#FF001F',relief=RIDGE,font="Ver-
dana 20 bold")
        helpLabel.place(relx = 0.8,rely = 0.2)

        helpLabel1 = tk.Label(self, textvariable=help, bg='#A7D0FF',fg='#FF001F',relief=RIDGE,font="Ver-
dana 20 bold")
        helpLabel1.place(relx = 0.8,rely = 0.3)

        helpLabel2 = tk.Label(self, textvariable=help,bg='#A7D0FF',fg='#FF001F',relief=RIDGE,font="Ver-
dana 20 bold")
        helpLabel2.place(relx = 0.8,rely = 0.4)

        box = Balloon(self)

        box.bind_widget(helpLabel,balloonmsg="Please ensure your node name has no spaces")
        box.bind_widget(helpLabel1,balloonmsg="Please use the logical opera-
tors and, or, not\nYou can use brackets but there must be a space before and af-
ter them \nIf you want to create a reject condition use the keyword reject at the beginning ")
```

```
        box.bind_widget(helpLabel2,balloonmsg="This statement will be printed when the node is accepted")

        acceptance = tk.Label(self, textvariable=var5,bg='#A7D0FF',font="Verdana 20 ")
        acceptance.place( relx = 0, rely=0.3)

        statement = tk.Label(self, textvariable=var6,bg='#A7D0FF',font="Verdana 20 ")
        statement.place( relx = 0, rely=0.4)

        self.nameEntry = tk.Entry(self,font='Calibri 20')
        self.nameEntry.place(relx = 0.25,rely = 0.2,relheight=0.1,relwidth=0.5)

        self.acceptanceEntry = tk.Entry(self,font='Calibri 20')
        self.acceptanceEntry.place(relx = 0.25,rely = 0.3,relheight=0.1,relwidth=0.5)

        self.statementEntry = tk.Entry(self,font='Calibri 20')
        self.statementEntry.place(relx = 0.25,rely = 0.4,relheight=0.1,relwidth=0.5)

        addDefault = ttk.Button(self, text = 'NEXT',style= 'my.TButton',command= lambda: self.addDe-
fault())
        addDefault.place(relheight= 0.1 , relwidth = 0.35, relx = 0.32, rely=0.85 )

        addCondition = ttk.Button(self, text = 'ADD CONDITION',style= 'my.TButton',com-
mand= lambda: self.addCondition())
        addCondition.place(relheight= 0.1 , relwidth = 0.35, relx = 0.32, rely=0.75 )

    def addDefault(self):

        self.info.name = self.name
        self.info.acceptance = self.acceptance
        self.info.statement = self.statement

        self.acceptanceEntry.delete(0,10000000)
        self.statementEntry.delete(0,10000000)
        self.nameEntry.delete(0,10000000)

        self.controller.frameCreation(DefaultCreation, self.info)
        self.controller.show_frame(DefaultCreation)

    def addCondition(self):

        if self.nameEntry.get() != '':
            self.name = self.nameEntry.get()
        acceptance = self.acceptanceEntry.get()
        statement = self.statementEntry.get()

        if acceptance != '':
            self.acceptance.append(acceptance)
            self.statement.append(statement)
```

```
        self.acceptanceEntry.delete(0,10000000)
        self.statementEntry.delete(0,10000000)

class MultiCreation(Background):

    def __init__(self,parent,controller, info):

        Background.__init__(self,parent)

        self.name = ''
        self.acceptance = []
        self.statement = []
        self.question = ''

        self.controller = controller
        self.info = info

        var = tk.StringVar()
        var.set("Domain Name:")

        var2 = tk.StringVar()
        var2.set(self.info.adf.name)

        var3 = tk.StringVar()
        var3.set('Create Node')

        var4 = tk.StringVar()
        var4.set('Name:')

        var5 = tk.StringVar()
        var5.set('Acceptance:')

        var6 = tk.StringVar()
        var6.set('Statement:')

        var7 = tk.StringVar()
        var7.set('Question:')

        help = tk.StringVar()
        help.set('?')

        helpLabel = tk.Label(self, textvariable=help,bg='#A7D0FF',fg='#FF001F',relief=RIDGE,font="Ver-
dana 20 bold")
        helpLabel.place(relx = 0.8,rely = 0.2)

        helpLabel1 = tk.Label(self, textvariable=help,bg='#A7D0FF',fg='#FF001F',relief=RIDGE,font="Ver-
dana 20 bold")
```

```
        helpLabel1.place(relx = 0.8,rely = 0.3)

        helpLabel2 = tk.Label(self, textvariable=help,bg='#A7D0FF',fg='#FF001F',relief=RIDGE,font="Ver-
dana 20 bold")
        helpLabel2.place(relx = 0.8,rely = 0.4)

        helpLabel3 = tk.Label(self, textvariable=help,bg='#A7D0FF',fg='#FF001F',relief=RIDGE,font="Ver-
dana 20 bold")
        helpLabel3.place(relx = 0.8,rely = 0.5)

        box = Balloon(self)

        box.bind_widget(helpLabel,balloonmsg="Please ensure your node name has no spaces")
        box.bind_widget(helpLabel1,balloonmsg="Please use the logical opera-
tors and, or, not\nYou can use brackets but there must be a space before and af-
ter them \nIf you want to create a reject condition use the keyword reject at the beginning ")
        box.bind_widget(helpLabel2,balloonmsg="This statement will be printed when the node is accepted")
        box.bind_widget(helpLabel3,balloonmsg="Set the question which will be asked to determine this ac-
ceptance condition\nThe possible answers will be taken from the acceptance conditions")


        label = tk.Label(self, textvariable=var,bg='#A7D0FF',font="Verdana 20")
        label.pack(side = 'left',anchor='nw')

        label2 = tk.Label(self, textvariable=var2,bg='#A7D0FF',font="Verdana 20")
        label2.pack(side = 'left',anchor='nw')

        label3 = tk.Label(self, textvariable=var3,bg='#A7D0FF',font="Verdana 20 underline")
        label3.place( relx = 0, rely=0.1 )

        name = tk.Label(self, textvariable=var4,bg='#A7D0FF',font="Verdana 20 ")
        name.place( relx = 0, rely=0.2 )

        acceptance = tk.Label(self, textvariable=var5,bg='#A7D0FF',font="Verdana 20 ")
        acceptance.place( relx = 0, rely=0.3)

        statement = tk.Label(self, textvariable=var6,bg='#A7D0FF',font="Verdana 20 ")
        statement.place( relx = 0, rely=0.4)

        question = tk.Label(self, textvariable=var7,bg='#A7D0FF',font="Verdana 20 ")
        question.place( relx = 0, rely=0.5)

        self.nameEntry = tk.Entry(self,font='Calibri 20')
        self.nameEntry.place(relx = 0.25,rely = 0.2,relheight=0.1,relwidth=0.5)

        self.acceptanceEntry = tk.Entry(self,font='Calibri 20')
        self.acceptanceEntry.place(relx = 0.25,rely = 0.3,relheight=0.1,relwidth=0.5)
```

```
        self.statementEntry = tk.Entry(self,font='Calibri 20')
        self.statementEntry.place(relx = 0.25,rely = 0.4,relheight=0.1,relwidth=0.5)

        self.questionEntry = tk.Entry(self,font='Calibri 20')
        self.questionEntry.place(relx = 0.25,rely = 0.5,relheight=0.1,relwidth=0.5)

        addDefault = ttk.Button(self, text = 'NEXT',style= 'my.TButton',command= lambda: self.addDe-
fault())
        addDefault.place(relheight= 0.1 , relwidth = 0.35, relx = 0.32, rely=0.85 )

        addCondition = ttk.Button(self, text = 'ADD CONDITION',style= 'my.TButton',com-
mand= lambda: self.addCondition())
        addCondition.place(relheight= 0.1 , relwidth = 0.35, relx = 0.32, rely=0.65 )

        addQuestion = ttk.Button(self, text = 'ADD QUESTION',style= 'my.TButton',com-
mand= lambda: self.addQuestion())
        addQuestion.place(relheight= 0.1 , relwidth = 0.35, relx = 0.32, rely=0.75 )

    def addQuestion(self):

        if self.questionEntry.get() != '':
            self.question = self.questionEntry.get()

            if self.info.editFlag:
                self.info.adf.questionOrder.append(self.nameEntry.get())

        self.questionEntry.delete(0,10000000)

    def addDefault(self):

        self.info.name = self.name
        self.info.acceptance = self.acceptance
        self.info.statement = self.statement
        self.info.question = self.question
        self.info.multiFlag = True

        self.acceptanceEntry.delete(0,10000000)
        self.statementEntry.delete(0,10000000)
        self.nameEntry.delete(0,10000000)
        self.questionEntry.delete(0,10000000)

        self.controller.frameCreation(DefaultCreation, self.info)
        self.controller.show_frame(DefaultCreation)

    def addCondition(self):

        if self.nameEntry.get() != '':
            self.name = self.nameEntry.get()
```

```
        acceptance = self.acceptanceEntry.get()
        statement = self.statementEntry.get()

        if acceptance != '':
            self.acceptance.append(acceptance)
            self.statement.append(statement)

        self.acceptanceEntry.delete(0,10000000)
        self.statementEntry.delete(0,10000000)

class DefaultCreation(Background):

    def __init__(self,parent,controller, info):

        Background.__init__(self,parent)

        self.info = info

        self.name = self.info.name
        self.acceptance = self.info.acceptance
        self.statement = self.info.statement
        self.question = self.info.question
        self.controller = controller

        var = tk.StringVar()
        var.set("Domain Name:")

        var2 = tk.StringVar()
        var2.set(self.info.adf.name)

        var3 = tk.StringVar()
        var3.set('Create Node')

        var6 = tk.StringVar()
        var6.set('Default Statement:')

        help = tk.StringVar()
        help.set('?')

        helpLabel = tk.Label(self, textvariable=help,bg='#A7D0FF',fg='#FF001F',relief=RIDGE,font="Ver-
dana 20 bold")
        helpLabel.place(relx = 0.8,rely = 0.3)

        box = Balloon(self)

        box.bind_widget(helpLabel,balloonmsg="This statement will print if the node is not accepted")

        label = tk.Label(self, textvariable=var,bg='#A7D0FF',font="Verdana 20")
```

```
        label.pack(side = 'left',anchor='nw')

        label2 = tk.Label(self, textvariable=var2,bg='#A7D0FF',font="Verdana 20")
        label2.pack(side = 'left',anchor='nw')

        label3 = tk.Label(self, textvariable=var3,bg='#A7D0FF',font="Verdana 20 underline")
        label3.place( relx = 0, rely=0.1 )

        default = tk.Label(self, textvariable=var6,bg='#A7D0FF',font="Verdana 20 ")
        default.place( relx = 0, rely=0.2)

        self.defaultEntry = tk.Entry(self,font='Calibri 20')
        self.defaultEntry.place(relx = 0.25,rely = 0.3,relheight=0.1,relwidth=0.5)

        end = ttk.Button(self, text = 'END',style= 'my.TButton',command= lambda: self.end())
        end.place(relheight= 0.1 , relwidth = 0.1, relx = 0.46, rely=0.8 )

        addStatement = ttk.Button(self, text = 'ADD STATEMENT',style= 'my.TButton',com-
mand= lambda: self.addStatement())
        addStatement.place(relheight= 0.1 , relwidth = 0.35, relx = 0.32, rely=0.5 )

        addNode = ttk.Button(self, text = 'ADD T/F NODE',style= 'my.TButton',com-
mand= lambda: self.standardNode())
        addNode.place(relheight= 0.1 , relwidth = 0.35, relx = 0.32, rely=0.6 )

        addMulti = ttk.Button(self, text = 'ADD MULTI NODE',style= 'my.TButton',com-
mand= lambda: self.multiNode())
        addMulti.place(relheight= 0.1 , relwidth = 0.35, relx = 0.32, rely=0.7 )

    def addStatement(self):

        statement = self.defaultEntry.get()

        if statement != '':
            self.statement.append(statement)

        self.defaultEntry.delete(0,10000000)

    def end(self):

        self.addNodeADF()

        self.info.multiFlag = False

        questionGeneration = False

        for i in self.info.adf.nodes.values():
```

```
            if i.children == None and i.question == None:
                questionGeneration = True

        if questionGeneration == True:
            self.controller.frameCreation(QuestionCreation, self.info)
            self.controller.show_frame(QuestionCreation)

        elif self.info.editFlag:
            self.controller.frameCreation(EditDomain, self.info)
            self.controller.show_frame(EditDomain)
            self.info.editFlag = False

        else:
            self.controller.frameCreation(SaveScreen, self.info)
            self.controller.show_frame(SaveScreen)

    def standardNode(self):

        self.addNodeADF()

        self.controller.frameCreation(NodeCreation, self.info)
        self.controller.show_frame(NodeCreation)

        self.info.multiFlag = False

    def multiNode(self):

        self.addNodeADF()

        self.controller.frameCreation(MultiCreation, self.info)
        self.controller.show_frame(MultiCreation)

    def addNodeADF(self):

        if self.info.multiFlag == True:
            self.info.adf.addMulti(self.name, self.acceptance, self.statement, self.question)
        else:
            self.info.adf.addNodes(self.name, self.acceptance, self.statement)

class QuestionCreation(Background):

    def __init__(self,parent,controller, info):

        Background.__init__(self,parent)

        self.info = info
        self.controller = controller
```

```
        name = self.info.adf.questionAssignment()

        self.nameVar = tk.StringVar()
        self.nameVar.set(name)
        self.name = name

        self.nameLabel = tk.Label(self, textvariable=self.nameVar,bg='#A7D0FF',font="Verdana 20 ")
        self.nameLabel.place( relx = 0.25, rely=0.2)

        var = tk.StringVar()
        var.set("Domain Name:")

        var2 = tk.StringVar()
        var2.set(self.info.adf.name)

        var3 = tk.StringVar()
        var3.set('Set Questions for Base-Level Factors')

        var4 = tk.StringVar()
        var4.set('Name:')

        var7 = tk.StringVar()
        var7.set('Question:')

        label = tk.Label(self, textvariable=var,bg='#A7D0FF',font="Verdana 20")
        label.pack(side = 'left',anchor='nw')

        label2 = tk.Label(self, textvariable=var2,bg='#A7D0FF',font="Verdana 20")
        label2.pack(side = 'left',anchor='nw')

        label3 = tk.Label(self, textvariable=var3,bg='#A7D0FF',font="Verdana 20 underline")
        label3.place( relx = 0, rely=0.1 )

        name = tk.Label(self, textvariable=var4,bg='#A7D0FF',font="Verdana 20 ")
        name.place( relx = 0, rely=0.2 )

        question = tk.Label(self, textvariable=var7,bg='#A7D0FF',font="Verdana 20 ")
        question.place( relx = 0, rely=0.3)

        self.questionEntry = tk.Entry(self,font='Calibri 20')
        self.questionEntry.place(relx = 0.25,rely = 0.3,relheight=0.1,relwidth=0.5)

        next = ttk.Button(self, text = 'NEXT',style= 'my.TButton',command= lambda: self.next())
        next.place(relheight= 0.1 , relwidth = 0.35, relx = 0.32, rely=0.85 )

        addQuestionEntry = ttk.Button(self, text = 'ADD QUESTION',style= 'my.TButton',com-
mand= lambda: self.addQuestion())
        addQuestionEntry.place(relheight= 0.1 , relwidth = 0.35, relx = 0.32, rely=0.75 )
```

```
  def addQuestion(self):
      node = self.info.adf.nodes[self.name]

      if self.questionEntry != '':
          node.question = self.questionEntry.get()

          if self.info.editFlag:
              self.info.adf.questionOrder.append(node.name)

      self.questionEntry.delete(0,10000000)

  def next(self):

      if self.info.adf.questionAssignment() == None:

          if self.info.editFlag == True:
              self.controller.frameCreation(EditDomain, self.info)
              self.controller.show_frame(EditDomain)
              self.info.editFlag = False
          else:
              self.controller.frameCreation(QuestionOrder, self.info)
              self.controller.show_frame(QuestionOrder)

      else:
          name = self.info.adf.questionAssignment()
          self.nameVar.set(name)
          self.name = name

class QuestionOrder(Background):
  def __init__(self,parent,controller, info):

      Background.__init__(self,parent)

      self.info = info
      self.controller = controller

      var = tk.StringVar()
      var.set("Domain Name:")

      var2 = tk.StringVar()
      var2.set(self.info.adf.name)

      var3 = tk.StringVar()
      var3.set('Set Question Order')

      label = tk.Label(self, textvariable=var,bg='#A7D0FF',font="Verdana 20")
      label.pack(side = 'left',anchor='nw')
```

```
    label2 = tk.Label(self, textvariable=var2,bg='#A7D0FF',font="Verdana 20")
    label2.pack(side = 'left',anchor='nw')

    label3 = tk.Label(self, textvariable=var3,bg='#A7D0FF',font="Verdana 20 underline")
    label3.place( relx = 0, rely=0.1 )

    next = ttk.Button(self, text = 'DONE',style= 'my.TButton',command= lambda: self.done())
    next.place(relheight= 0.1 , relwidth = 0.3, relx = 0.65, rely=0.65 )

    up = ttk.Button(self, text = 'UP',style= 'my.TButton',command= lambda: self.up())
    up.place(relheight= 0.1 , relwidth = 0.3, relx = 0.65, rely=0.45 )

    down = ttk.Button(self, text = 'DOWN',style= 'my.TButton',command= lambda: self.down())
    down.place(relheight= 0.1 , relwidth = 0.3, relx = 0.65, rely=0.55 )

    self.nodeDict = {}

    self.questionList = []

    #CHANGE
    if self.info.adf.questionOrder == []:

        for node in self.info.adf.nodes.values():
            if node.question != None:
                self.questionList.append(node.question)
                self.nodeDict[node.question] = node

    else:

        for name in self.info.adf.questionOrder:
            node = self.info.adf.nodes[name]
            if node.question != None:
                self.questionList.append(node.question)
                self.nodeDict[node.question] = node

        for node in self.info.adf.nodes.values():
            if node.question != None and node.question not in self.questionList:
                self.questionList.append(node.question)
                self.nodeDict[node.question] = node

    n = tk.StringVar(value=self.questionList)
    self.list = tk.Listbox(self, height=20, width = 70, listvariable = n)
    self.list.place(relx = 0.05, rely=0.25)

def done(self):

    newOrder = []
```

```
        for i in self.questionList:

            x = self.nodeDict[i]
            newOrder.append(x.name)

        self.info.adf.questionOrder = newOrder

        self.controller.frameCreation(EditDomain, self.info)
        self.controller.show_frame(EditDomain)

    def up(self):

        try:
            index = self.list.curselection()[0]

            if index == 0:
                pass
            else:
                text=self.list.get(index)
                self.list.delete(index)

                self.list.insert(index-1, text)
                self.questionList.pop(index)
                self.questionList.insert(index-1, text)
                self.list.selection_set(index-1)

        except:
            pass

    def down(self):

        try:
            index = self.list.curselection()[0]


            text=self.list.get(index)
            self.list.delete(index)

            self.list.insert(index+1, text)
            self.questionList.pop(index)
            self.questionList.insert(index+1, text)
            self.list.selection_set(index+1)

        except:
            pass

class SaveScreen(Background):
```

```
  def __init__(self,parent,controller, info):

      Background.__init__(self,parent)

      self.info = info
      self.controller = controller

      name = self.info.adf.questionAssignment()

      self.name = name

      var = tk.StringVar()
      var.set("Domain Name:")

      var2 = tk.StringVar()
      var2.set(self.info.adf.name)

      var3 = tk.StringVar()
      var3.set('Save ADF as .xlsx file')

      var4 = tk.StringVar()
      var4.set('Filename:')

      help = tk.StringVar()
      help.set('?')

      helpLabel = tk.Label(self, textvariable=help,bg='#A7D0FF',fg='#FF001F',relief=RIDGE,font="Ver-
dana 20 bold")
      helpLabel.place(relx = 0.8,rely = 0.2)

      box = Balloon(self)

      box.bind_widget(helpLabel,balloonmsg="Please ensure your filename has no spaces")

      label = tk.Label(self, textvariable=var,bg='#A7D0FF',font="Verdana 20")
      label.pack(side = 'left',anchor='nw')

      label2 = tk.Label(self, textvariable=var2,bg='#A7D0FF',font="Verdana 20")
      label2.pack(side = 'left',anchor='nw')

      label3 = tk.Label(self, textvariable=var3,bg='#A7D0FF',font="Verdana 20 underline")
      label3.place( relx = 0, rely=0.1 )

      name = tk.Label(self, textvariable=var4,bg='#A7D0FF',font="Verdana 20 ")
      name.place( relx = 0, rely=0.2 )

      self.fileName = tk.Entry(self,font='Calibri 20')
      self.fileName.place(relx = 0.25,rely = 0.2,relheight=0.1,relwidth=0.5)
```

```
        save = ttk.Button(self, text = 'SAVE',style= 'my.TButton',command= lambda: self.save())
        save.place(relheight= 0.1 , relwidth = 0.35, relx = 0.32, rely=0.75 )
        menu = ttk.Button(self, text = 'MENU',style= 'my.TButton',command= lambda: self.menu())
        menu.place(relheight= 0.1 , relwidth = 0.35, relx = 0.32, rely=0.85 )

    def save(self):

        if self.fileName.get() != '':
            filename = self.fileName.get()
            self.info.adf.saveNew(filename)

            varDone = tk.StringVar()
            varDone.set('DONE')

            self.done = tk.Label(self, textvariable=varDone,bg='#A7D0FF',font="Verdana 20")
            self.done.place( relx = 0.42, rely=0.4 )

        else:
            pass

    def menu(self):

        self.controller.show_frame(MainMenu)

class Report(Background):

    def __init__(self,parent,controller, info):

        Background.__init__(self,parent)

        self.controller = controller
        self.info = info

        self.graph = self.info.adf.visualiseNetwork(self.info.adf.case)

        var = tk.StringVar()
        var.set("Domain Name:")

        var2 = tk.StringVar()
        var2.set(self.info.adf.name)

        self.var3 = tk.StringVar()
        self.var3.set("Report for {}:".format(self.info.caseName))

        label = tk.Label(self, textvariable=var,bg='#A7D0FF',font="Verdana 20 underline")
        label.pack(side = 'left',anchor='nw')
```

```
label2 = tk.Label(self, textvariable=var2,bg='#A7D0FF',font="Verdana 20")
label2.pack(side = 'left',anchor='nw')

label3 = tk.Label(self, textvariable=self.var3,bg='#A7D0FF',font="Verdana 20")
label3.place( relx = 0.05, rely=0.1)

nodeNames = set()
for i in self.graph.get_node_list():
    nodeNames.add(i.get_name())

#make a set so it only counts unique nodes
numNodes = len(nodeNames)


var4 = tk.StringVar()
var4.set("{} nodes in this domain".format(numNodes))

label4 = tk.Label(self, textvariable=var4,bg='#A7D0FF',font="Verdana 16")
label4.place( relx = 0.05, rely=0.2)

numAccepted = len(self.info.adf.case)

var5 = tk.StringVar()
var5.set("{} nodes accepted in this case".format(numAccepted))

label5 = tk.Label(self, textvariable=var5,bg='#A7D0FF',font="Verdana 16")
label5.place( relx = 0.05, rely=0.3)

var6 = tk.StringVar()
var6.set("Factors in this case: ")

label6 = tk.Label(self, textvariable=var6,bg='#A7D0FF',font="Verdana 16")
label6.place( relx = 0.05, rely=0.4)

n = tk.StringVar()
self.combo = ttk.Combobox(self, width = 50, textvariable = n, state='readonly')

self.combo['values'] = self.info.case
self.combo.place(relx = 0.45, rely=0.41)

outcome = self.info.adf.statements[-1]
var7 = tk.StringVar()
var7.set("Outcome: {}".format(outcome))

label7 = tk.Label(self, textvariable=var7,bg='#A7D0FF',font="Verdana 16")
label7.place( relx = 0.05, rely=0.5)
```

```
        visualise = ttk.Button(self, text = 'VISUALISE',style= 'my.TButton',command= lambda: self.visual-
ise())
        visualise.place(relheight= 0.1 , relwidth = 0.2, relx = 0.42, rely=0.75 )

        back = ttk.Button(self, text = 'BACK',style= 'my.TButton',command= lambda: self.back())
        back.place(relheight= 0.1 , relwidth = 0.2, relx = 0.42, rely=0.85 )

        self.fileName = tk.Entry(self,font='Calibri 16')
        self.fileName.place(relx = 0.3,rely = 0.65,relheight=0.07,relwidth=0.5)

        var4 = tk.StringVar()
        var4.set('Filename:')

        filename = tk.Label(self, textvariable=var4,bg='#A7D0FF',font="Verdana 16 ")
        filename.place( relx = 0.05, rely=0.65 )

    def visualise(self):

        if self.fileName.get() != '':

            filename = self.fileName.get()

            self.graph.write_png('{}.png'.format(filename))

            os.startfile('{}.png'.format(filename))

    def back(self):
        self.controller.show_frame(Outcome)

app = UI()

app.mainloop()
```

H.4 WildAnimals.py

This file initialises the Wild Animals domain. adf() adds the node data to the ADF. cases() contains the base-level factors for each case in the test dataset. expectedOutcome() contains the expected outcome data for the test dataset.

```
from MainClasses import *

def adf():
    """
    the adf for the domain
    """
    adf = ADF('WildAnimals')
```

```
    #non-leaf factors
    adf.addNodes('Decide',['Ownership or ( RightToPursue and IllegalAct and not NoBlame )','RightToPursue and IllegalAct'],['find for the plaintiff, find against the defendant','do not find for the plaintiff, the defendant did not act illegally, do not find against the defendant','do not find for the plaintiff, find for the defendant'])
    adf.addNodes('RightToPursue',['OwnsLand or ( ( HotPursuit and PMotive ) or ( PMotive and ( not DMotive ) ) )'],['plaintiff had a right to pursue the quarry','plaintiff had no right to pursue the quarry'])
    adf.addNodes('Ownership',['( OwnsLand and Resident ) or Convention or Capture'],['the plaintiff owned the quarry','the plaintiff did not own the quarry'])
    adf.addNodes('IllegalAct',['Trespass or Assault'],['an illegal act was committed','no illegal act was committed'])
    adf.addNodes('Trespass',['LegalOwner and AntiSocial'],['defendant committed trespass','defendant committed no trespass'])
    adf.addNodes('AntiSocial',['( Nuisance or Impolite ) and ( not DMotive )'],['defendant committed an antisocial act','defendant committed no antisocial acts'])
    adf.addNodes('PMotive',['PLiving or ( ( PSport or PGain ) and ( not DLiving ) )'],['plaintiff has good motive','plantiff has no good motive'])
    adf.addNodes('DMotive',['not Malice and ( DLiving or DSport or DGain )'],['defendant has good motive','defendant has no good motive'])
    adf.addNodes('Capture',['not NotCaught'],['the plaintiff had captured the quarry','the plaintiff had not captured the quarry'])
    adf.addNodes('OwnsLand',['LegalOwner'],['plaintiff owned the land','plaintiff did not own the land'])

    #questions
    adf.addNodes('NoBlame',question='Was the defendant blameless in the interference of the plaintiff\'s pursuit?')
    adf.addNodes('Resident',question='Did the quarry reside on the land?')
    adf.addNodes('Convention',question='Is the possession of the quarry governed by convention?')
    adf.addNodes('Assault',question='Did an assault prevent the plaintiff from retaining possession of the quarry?')
    adf.addNodes('LegalOwner',question='Was the plaintiff the legal owner of the land?')
    adf.addNodes('Nuisance',question='Did the defendant\'s interference with the plaintiff\'s pursuit amount to a nuisance?')
    adf.addNodes('Impolite',question='Was the interference of the defendant in the plaintiff\'s pursuits impolite?')
    adf.addNodes('PLiving',question='Was the plaintiff pursuing the quarry for their livelihood?')
    adf.addNodes('PSport',question='Was the plaintiff pursuing the quarry for sport?')
    adf.addNodes('PGain',question='Did the plaintiff seek to personally gain from the quarry?')
    adf.addNodes('DLiving',question='Was the defendant pursuing the quarry for their livelihood?')
    adf.addNodes('Malice',question='Was the defendant malicious in their motive?')
    adf.addNodes('DSport',question='Was the defendant pursuing the quarry for sport?')
    adf.addNodes('DGain',question='Did the defendant seek to personally gain from the quarry?')
    adf.addNodes('NotCaught',question='Was the quarry not caught by the plaintiff?')
    adf.addNodes('HotPursuit',question='Was the plaintiff in hot pursuit of the quarry?')

    adf.questionOrder = ['PSport','PGain','PLiving','DSport','DGain','DLiving','Malice','HotPursuit','NotCaught','LegalOwner','Impolite','Nuisance','Assault','Resident','Convention','NoBlame']
```

```
    return adf

def cases():
    """
    test cases
    """

    keeble = ['NotCaught','LegalOwner','Malice','Nuisance','DSport','PLiving']
    pierson = ['NotCaught','HotPursuit','Impolite','PSport','Vermin']
    young = ['NotCaught','HotPursuit','Impolite','PLiving','DLiving']
    ghen = ['NotCaught','Convention','NoBlame','PLiving','DLiving']
    popov = ['NotCaught','HotPursuit','Assault','NoBlame','PGain','DGain']

    cases = {'Keeble v Hickeringill':keeble,'Pierson v Post':pierson,'Young v Hitch-
ens':young,'Ghen v Rich':ghen,'Popov v Hayashi':popov}

    return cases

def expectedOutcomeCases():
    """
    first factor is the outcome - the other factors are those from the prolog program of the domain
    """
    keeble = ['find for the plaintiff, find against the defendant','IllegalAct','Trespass','AntiSocial','RightTo-
Pursue','OwnsLand','PMotive','NotCaught','LegalOwner','Malice','Nuisance','DSport','PLiving']
    pierson = ['do not find for the plaintiff, find for the defendant','AntiSocial','RightToPursue','PMo-
tive','NotCaught','HotPursuit','Impolite','PSport','Vermin']
    young = ['do not find for the plaintiff, find for the defendant','RightToPursue','DMotive','PMo-
tive','NotCaught','HotPursuit','Impolite','PLiving','DLiving']
    ghen = ['find for the plaintiff, find against the defendant','DMotive','PMotive','Owner-
ship','NotCaught','Convention','NoBlame','PLiving','DLiving']
    popov = ['do not find for the plaintiff, the defendant did not act illegally, do not find against the defend-
ant','IllegalAct','RightToPursue','DMotive','PMotive','NotCaught','HotPursuit','Assault','No-
Blame','PGain','DGain']

    cases = {'Keeble v Hickeringill':keeble,'Pierson v Post':pierson,'Young v Hitch-
ens':young,'Ghen v Rich':ghen,'Popov v Hayashi':popov}

    return cases
```

H.5 TradeSecrets.py

This file initialises the Trade Secrets domain. adf() adds the node data to the ADF. cases() contains the base-level factors for each case in the test dataset. expectedOutcome() contains the expected outcome data for the test dataset.

```
from MainClasses import *

def adf():

    """
    creates the ADF for the domain
    """

    adf = ADF('TradeSecrets')

    #non-leaf factors
    adf.addNodes('Decide',['( not DefendantOwnershipRights ) and ( TradeSecretMisappropriation and ( ImproperMeans or ConfidentialRelationship ) )'],['a trade secret was misappropriated, find for plaintiff','no trade secret was misappropriated, find for defendant'])
    adf.addNodes('TradeSecretMisappropriation',['EffortstoMaintainSecrecy and InfoValuable and not InfoKnownOrAvailiable'],['information was a trade secret','information was not a trade secret'])
    adf.addNodes('EffortstoMaintainSecrecy',['SecurityMeasures or MaintainSecrecyDefendant or MaintainSecrecyOutsiders or not NoSecurityMeasures'],['efforts were taken to maintain secrecy','no efforts were taken to maintain secrecy'])
    adf.addNodes('InfoValuable',['UniqueProduct or CompetitiveAdvantage or not InfoKnownOrAvailiable'],['the information was valuable','the information was not valuable'])
    adf.addNodes('InfoKnownOrAvailiable',['InfoKnown or InfoAvailiableElsewhere'],['the information was known or availiable','the information was neither known nor availiable'])
    adf.addNodes('InfoKnown',['InfoKnownToCompetitors and not UniqueProduct'],['information is known','information is not known'])
    adf.addNodes('InfoAvailiableElsewhere',['( InfoReverseEngineerable or InfoObtainableElsewhere ) and not ( InfoReverseEngineerable and ( RestrictedMaterialsUsed or IdenticalProducts ) )'],['the information was availiable elsewhere','the information was not availiable elsewhere'])
    adf.addNodes('ImproperMeans',['QuestionableMeans and not LegitimatelyObtainable'],['improper means were used','improper means were not used'])
    adf.addNodes('QuestionableMeans',['( not InfoReverseEngineered and not InfoIndependentlyGenerated ) and ( InvasiveTechniques or Deception or RestrictedMaterialsUsed or BribeEmployee )'],['questionable means were used','questionable means were not used'])
    adf.addNodes('InfoUsed',['BroughtTools or IdenticalProducts or CompetitiveAdvantage or not InfoIndependentlyGenerated'],['the information was used','the information was not used'])
    adf.addNodes('ConfidentialRelationship',['NoticeOfConfidentiality or ConfidentialityAgreement'],['there was a confidential relationship','there was no confidential relationship'])
    adf.addNodes('NoticeOfConfidentiality',['WaiverOfConfidentiality or KnewInfoConfidential or RestrictedMaterialsUsed or NoncompetitionAgreement or AgreedNotToDisclose'],['defendant was on notice of confidentiality','defendant was not on notice of confidentiality'])
    adf.addNodes('LegitimatelyObtainable',['InfoKnownOrAvailiable and not QuestionableMeans'],['the information was legitimately obtained','the information was not legitimately obtained'])
    adf.addNodes('ConfidentialityAgreement',['AgreedNotToDisclose and not WaiverOfConfidentiality'],['there was a confidentiality agreement','there was no confidentiality agreement'])
    adf.addNodes('MaintainSecrecyDefendant',['AgreedNotToDisclose'],['efforts made vis a vis defendant','no efforts made vis a vis defendant'])
    adf.addNodes('MaintainSecrecyOutsiders',['OutsiderDisclosuresRestricted'],['efforts made vis a vis outsiders','no efforts made vis a vis outsiders'])
```

```
    adf.addNodes('DefendantOwnershipRights',['EmployeeSoleDeveloper'],['defend-
ant is owner of secret','defendant is not owner of secret'])

    #leaf factors and questions
    adf.addNodes('BribeEmployee',question='Did the defendant offer the plaintiff\'s current or former em-
ployee an incentive to work for the defendant?')
    adf.addNodes('EmployeeSoleDeveloper',question='Was the defendant the sole developer of the prod-
uct whilst employed by the plaintiff?')
    adf.addNodes('AgreedNotToDisclose',question='Had the defendant entered into a non-disclosure agree-
ment with the plaintiff?')
    adf.addNodes('SecurityMeasures',question='Did the plaintiff take measures to ensure the secu-
rity of its information?')
    adf.addNodes('BroughtTools',question='Did an employee of the plaintiff give product development in-
formation to the defendant?')
    adf.addNodes('CompetitiveAdvantage',question='Did the plaintiff\'s product information allow the de-
fendant to save time or expense?')
    adf.addNodes('OutsiderDisclosuresRestricted',question='Was the plaintiff\'s disclosure to outsiders sub-
ject to confidential restrictions?')
    adf.addNodes('NoncompetitionAgreement',question='Had the plantiff and the defendant en-
tered into a noncompetition agreement?')
    adf.addNodes('RestrictedMaterialsUsed',question='Did the defendant use materials that were sub-
ject to confidentiality restrictions?')
    adf.addNodes('UniqueProduct',question='Is the product of the plaintiff unique?')
    adf.addNodes('InfoReverseEngineerable',question='Could the plaintiff\'s product infor-
mation have been learned by reverse engineering?')
    adf.addNodes('InfoIndependentlyGenerated',question='Did the defendant develop its product through in-
dependent research?')
    adf.addNodes('IdenticalProducts',question='Was the defendant\'s product identical to the plaintiff\'s?')
    adf.addNodes('NoSecurityMeasures',question='Did the plaintiff not adopt any secu-
rity measures to maintain the secrecy of their information?')
    adf.addNodes('InfoKnownToCompetitors',question='Was the plaintiff\'s information known to competi-
tors?')
    adf.addNodes('KnewInfoConfidential',question='Did the defendant know the plaintiff\'s infor-
mation was confidential?')
    adf.addNodes('InvasiveTechniques',question='Did the defendant use invasive techniques to gain ac-
cess to the plaintiff\'s information? ')
    adf.addNodes('WaiverOfConfidentiality',question='Had the plaintiff entered into an agree-
ment that waived confidentiality?')
    adf.addNodes('InfoObtainableElsewhere',question='Could the information have been obtained from pub-
licly availiable sources?')
    adf.addNodes('InfoReverseEngineered',question='Did the defendant discover the plaintiff\'s product in-
formation through reverse engineering?' )
    adf.addNodes('Deception',question='Did the defendant obtain the plaintiff\'s information through decep-
tion?')

    adf.questionOrder = ['BribeEmployee','EmployeeSoleDeveloper','AgreedNotToDisclose','Securi-
tyMeasures','BroughtTools','CompetitiveAdvantage','OutsiderDisclosuresRestricted','Noncompetition-
```

```
Agreement','RestrictedMaterialsUsed','UniqueProduct','InfoReverseEngineerable','InfoReverseEngineered','InfoIndependentlyGenerated','IdenticalProducts','NoSecurityMeasures','InfoKnownToCompetitors','KnewInfoConfidential','InvasiveTechniques','WaiverOfConfidentiality','InfoObtainableElsewhere','Deception']

    return adf

def cases():
    """
    test cases
    """
    arco = ['SecretsDisclosedOutsiders', 'InfoReverseEngineerable','InfoKnownToCompetitors'] #pass
    boeing = ['AgreedNotToDisclose','SecurityMeasures','OutsiderDisclosuresRestricted','RestrictedMaterialsUsed','KnewInfoConfidential','DisclosureInNegotations','SecretsDisclosedOutsiders'] #pass
    bryce = ['AgreedNotToDisclose','SecurityMeasures','IdenticalProducts','KnewInfoConfidential','DisclosureInNegotations'] #pass
    collegeWatercolour = ['UniqueProduct','Deception','DisclosureInNegotations'] #pass
    denTalEz = ['AgreedNotToDisclose','SecurityMeasures','KnewInfoConfidential','Deception','DisclosureInNegotations'] #pass
    ecolgix = ['KnewInfoConfidential','DisclosureInNegotations','NoSecurityMeasures','WaiverOfConfidentiality'] #pass
    emery = ['IdenticalProducts','KnewInfoConfidential','SecretsDisclosedOutsiders'] #pass
    ferranti = ['BribeEmployee','InfoIndependentlyGenerated','NoSecurityMeasures','InfoKnownToCompetitors','DisclosureInPublicForum'] #pass
    robinson = ['IdenticalProducts','Deception','DisclosureInNegotations','SecretsDisclosedOutsiders','NoSecurityMeasures'] #pass
    sandlin = ['DisclosureInNegotations','SecretsDisclosedOutsiders','InfoReverseEngineerable','NoSecurityMeasures','DisclosureInPublicForum'] #pass
    sheets = ['IdenticalProducts','NoSecurityMeasures','DisclosureInPublicForum'] #pass
    spaceAero = ['CompetitiveAdvantage','UniqueProduct','IdenticalProducts','DisclosureInNegotations','NoSecurityMeasures'] #pass
    televation = ['SecurityMeasures','OutsiderDisclosuresRestricted','UniqueProduct','IdenticalProducts','KnewInfoConfidential','SecretsDisclosedOutsiders','InfoReverseEngineerable'] #pass
    yokana = ['BroughtTools','SecretsDisclosedOutsiders','InfoReverseEngineerable','DisclosureInPublicForum'] #pass
    cm1 = ['AgreedNotToDisclose','SecurityMeasures','InfoKnownToCompetitors','InfoIndependentlyGenerated','InfoReverseEngineerable','SecretsDisclosedOutsiders','DisclosureInPublicForum'] #pass
    digitalDevelopment = ['SecurityMeasures','CompetitiveAdvantage','UniqueProduct','IdenticalProducts','KnewInfoConfidential','DisclosureInNegotations'] #pass
    fmc = ['AgreedNotToDisclose','SecurityMeasures','BroughtTools','OutsiderDisclosuresRestricted','SecretsDisclosedOutsiders','VerticalKnowledge'] #pass
    forrest = ['SecurityMeasures','UniqueProduct','KnewInfoConfidential','DisclosureInNegotations'] #pass
    goldberg = ['KnewInfoConfidential','DisclosureInNegotations','SecretsDisclosedOutsiders','DisclosureInPublicForum'] #pass
    kg = ['SecurityMeasures','RestrictedMaterialsUsed','UniqueProduct','IdenticalProducts','KnewInfoConfidential','InfoReverseEngineerable','InfoReverseEngineered'] #pass
    laser = ['SecurityMeasures','OutsiderDisclosuresRestricted','KnewInfoConfidential','DisclosureInNegotations','SecretsDisclosedOutsiders'] #pass
```

```
    lewis = ['CompetitiveAdvantage','KnewInfoConfidential','DisclosureInNegotations'] #pass
    mbl = ['AgreedNotToDisclose','SecurityMeasures','NoncompetitionAgreement','Agreement-
NotSpecific','SecretsDisclosedOutsiders','InfoKnownToCompetitors'] #pass
    mason = ['SecurityMeasures','UniqueProduct','KnewInfoConfidential','DisclosureInNegotations','In-
foReverseEngineerable'] #pass
    mineralDeposits = ['IdenticalProducts','RestrictedMaterialsUsed','DisclosureInNegotations','In-
foReverseEngineerable','InfoReverseEngineered'] #pass
    nationalInstruments = ['IdenticalProducts','KnewInfoConfidential','DisclosureInNegotations'] #pass
    nationalRejectors = ['BroughtTools','UniqueProduct','IdenticalProducts','SecretsDisclosedOutsiders','In-
foReverseEngineerable','NoSecurityMeasures','DisclosureInPublicForum'] #pass
    reinforced = ['AgreedNotToDisclose','SecurityMeasures','CompetitiveAdvantage','UniquePro-
duct','KnewInfoConfidential','DisclosureInNegotations'] #pass
    scientology = ['AgreedNotToDisclose','SecurityMeasures','OutsiderDisclosuresRestricted','SecretsDis-
closedOutsiders','VerticalKnowledge','InfoKnownToCompetitors'] #pass
    technicon = ['SecurityMeasures','OutsiderDisclosuresRestricted','RestrictedMaterialsUsed','KnewIn-
foConfidential','SecretsDisclosedOutsiders','InfoReverseEngineerable','InfoReverseEngineered'] #pass
    trandes = ['AgreedNotToDisclose','SecurityMeasures','OutsiderDisclosuresRestricted','DisclosureInNe-
gotations','SecretsDisclosedOutsiders'] #pass
    valcoCincinnati = ['SecurityMeasures','OutsiderDisclosuresRestricted','UniqueProduct','KnewInfoConfi-
dential','DisclosureInNegotations','SecretsDisclosedOutsiders'] #pass

    cases = {'Arco Industries Corp v Chemcast Corp':arco, 'The Boeing Company v Sierracin Corp':boe-
ing,'M. Bryce & Associates Inc v Gladstone':bryce, 'College Watercolour Group Inc v William H. New-
bauer':collegeWatercolour,'Den-Tal-Ez Inc v Siemens Capital Corp':denTalEz,'Ecol-
gix Inc v Fantsteel Inc':ecolgix,'A.H. Emery Co v Marcon Products Corp':emery,'Ferranti Elec-
tric Inc v Harwood':ferranti,'Commonwealth v Robinson':robinson,'Sandlin v John-
son':sandlin,'Sheets v Yamaha Motors Corp':sheets,'Space Aero Products Corp v R.E. Darling Corp':space-
Aero,'Television Telecommunications Systems Inc v Saindon':television,'Midland-
Ross Corp v Yokana':yokana,'CMI Corp v Jakob':cm1,'Digital Development Corp v Interna-
tional Memory Systems':digitalDevelopment,'FMC Corp v Taiwan Tainan Giant Ind Co Ltd':fmc,'For-
est Laboratories Inc v Formulations Inc':forrest,'Goldberg v Medtronic':goldberg,'K & G Oil Tool & Ser-
vices Co v G & G':kg,'Laser Industries Ltd v Eder Instrument Co':laser,'Computer Print Systems v Lew-
is':lewis,'MBL (USA) Corp v Diekman':mbl,'Mason v Jack Daniel Distillery':mason,'Mineral Depos-
its Ltd v Zigan':mineralDeposits,'National Instruments Labs Inc v Hycel Inc':nationalInstruments,'Na-
tional Rejectors Inc v Trieman':nationalRejectors,'Reinforced':reinforced,'Scientology':scientology, 'Tech-
nicon Data Systems Corp v Curtis':technicon,'Trandes Corp v Guy F. Atkinson Co':trandes,'Valco Cincin-
nati Inc v N & D Machining Service Inc':valcoCincinnati}

    return cases

def expectedOutcomeCases():
    """
    first factor is the outcome, the other factors are the same as in the prolog program
    """
    arco = ['no trade secret was misappropriated, find for defendant','LegitimatelyObtainable','Effort-
stoMaintainSecrecy','InfoKnownOrAvailiable','InfoKnown','InfoAvailiableElsewhere','InfoUsed', 'Se-
cretsDisclosedOutsiders', 'InfoReverseEngineerable','InfoKnownToCompetitors']
```

```
boeing = ['a trade secret was misappropriated, find for plaintiff','TradeSecretMisappropriation','EffortstoMaintainSecrecy','InfoValuable','ImproperMeans','QuestionableMeans','InfoUsed','ConfidentialRelationship','NoticeOfConfidentiality','ConfidentialityAgreement','MaintainSecrecyDefendant','MaintainSecrecyOutsiders', 'AgreedNotToDisclose','SecurityMeasures','OutsiderDisclosuresRestricted','RestrictedMaterialsUsed','KnewInfoConfidential','DisclosureInNegotations','SecretsDisclosedOutsiders']
bryce = ['a trade secret was misappropriated, find for plaintiff','TradeSecretMisappropriation','EffortstoMaintainSecrecy','InfoValuable','InfoUsed','ConfidentialRelationship','NoticeOfConfidentiality','ConfidentialityAgreement','MaintainSecrecyDefendant', 'AgreedNotToDisclose','SecurityMeasures','IdenticalProducts','KnewInfoConfidential','DisclosureInNegotations']
collegeWatercolour = ['a trade secret was misappropriated, find for plaintiff','TradeSecretMisappropriation','EffortstoMaintainSecrecy','InfoValuable','ImproperMeans','QuestionableMeans','InfoUsed', 'UniqueProduct','Deception','DisclosureInNegotations']
denTalEz = ['a trade secret was misappropriated, find for plaintiff','TradeSecretMisappropriation','EffortstoMaintainSecrecy','InfoValuable','ImproperMeans','QuestionableMeans','InfoUsed','ConfidentialRelationship','NoticeOfConfidentiality','ConfidentialityAgreement','MaintainSecrecyDefendant', 'AgreedNotToDisclose','SecurityMeasures','KnewInfoConfidential','Deception','DisclosureInNegotations']
ecolgix = ['no trade secret was misappropriated, find for defendant','InfoValuable','InfoUsed','ConfidentialRelationship','NoticeOfConfidentiality', 'KnewInfoConfidential','DisclosureInNegotations','NoSecurityMeasures','WaiverOfConfidentiality']
emery = ['a trade secret was misappropriated, find for plaintiff','TradeSecretMisappropriation','EffortstoMaintainSecrecy','InfoValuable','InfoUsed','ConfidentialRelationship','NoticeOfConfidentiality', 'IdenticalProducts','KnewInfoConfidential','SecretsDisclosedOutsiders']
ferranti = ['no trade secret was misappropriated, find for defendant','LegitimatelyObtainable','InfoKnownOrAvailiable','InfoKnown', 'BribeEmployee','InfoIndependentlyGenerated','NoSecurityMeasures','InfoKnownToCompetitors','DisclosureInPublicForum']
robinson = ['no trade secret was misappropriated, find for defendant','InfoValuable','ImproperMeans','QuestionableMeans','InfoUsed','IdenticalProducts','Deception','DisclosureInNegotations','SecretsDisclosedOutsiders','NoSecurityMeasures']
sandlin = ['no trade secret was misappropriated, find for defendant','LegitimatelyObtainable','InfoKnownOrAvailiable','InfoAvailiableElsewhere','InfoUsed','DisclosureInNegotations','SecretsDisclosedOutsiders','InfoReverseEngineerable','NoSecurityMeasures','DisclosureInPublicForum']
sheets = ['no trade secret was misappropriated, find for defendant','InfoValuable','InfoUsed', 'IdenticalProducts','NoSecurityMeasures','DisclosureInPublicForum']
spaceAero = ['no trade secret was misappropriated, find for defendant','InfoValuable','InfoUsed', 'CompetitiveAdvantage','UniqueProduct','IdenticalProducts','DisclosureInNegotations','NoSecurityMeasures']
televation = ['a trade secret was misappropriated, find for plaintiff','TradeSecretMisappropriation','EffortstoMaintainSecrecy','InfoValuable','InfoUsed','ConfidentialRelationship','NoticeOfConfidentiality','MaintainSecrecyOutsiders', 'SecurityMeasures','OutsiderDisclosuresRestricted','UniqueProduct','IdenticalProducts','KnewInfoConfidential','SecretsDisclosedOutsiders','InfoReverseEngineerable']
yokana = ['no trade secret was misappropriated, find for defendant','LegitimatelyObtainable','EffortstoMaintainSecrecy','InfoKnownOrAvailiable','InfoAvailiableElsewhere','InfoUsed', 'BroughtTools','SecretsDisclosedOutsiders','InfoReverseEngineerable','DisclosureInPublicForum']
cm1 = ['no trade secret was misappropriated, find for defendant','LegitimatelyObtainable','EffortstoMaintainSecrecy','InfoKnownOrAvailiable','InfoKnown','InfoAvailiableElsewhere','ConfidentialRelationship','NoticeOfConfidentiality','ConfidentialityAgreement','MaintainSecrecyDefendant', 'AgreedNotToDisclose','SecurityMeasures','InfoKnownToCompetitors','InfoIndependentlyGenerated','InfoReverseEngineerable','SecretsDisclosedOutsiders','DisclosureInPublicForum']
```

```
digitalDevelopment = ['a trade secret was misappropriated, find for plaintiff','TradeSecretMisappropriation','EffortstoMaintainSecrecy','InfoValuable','InfoUsed','ConfidentialRelationship','NoticeOfConfidentiality', 'SecurityMeasures','CompetitiveAdvantage','UniqueProduct','IdenticalProducts','KnewInfoConfidential','DisclosureInNegotations']
fmc = ['a trade secret was misappropriated, find for plaintiff','TradeSecretMisappropriation','EffortstoMaintainSecrecy','InfoValuable','InfoUsed','ConfidentialRelationship','NoticeOfConfidentiality','ConfidentialityAgreement','MaintainSecrecyDefendant','MaintainSecrecyOutsiders','AgreedNotToDisclose','SecurityMeasures','BroughtTools','OutsiderDisclosuresRestricted','SecretsDisclosedOutsiders','VerticalKnowledge']
forrest = ['a trade secret was misappropriated, find for plaintiff','TradeSecretMisappropriation','EffortstoMaintainSecrecy','InfoValuable','InfoUsed','ConfidentialRelationship','NoticeOfConfidentiality', 'SecurityMeasures','UniqueProduct','KnewInfoConfidential','DisclosureInNegotations']
goldberg = ['a trade secret was misappropriated, find for plaintiff','TradeSecretMisappropriation','EffortstoMaintainSecrecy','InfoValuable','InfoUsed','ConfidentialRelationship','NoticeOfConfidentiality', 'KnewInfoConfidential','DisclosureInNegotations','SecretsDisclosedOutsiders','DisclosureInPublicForum']
kg = ['a trade secret was misappropriated, find for plaintiff','TradeSecretMisappropriation','EffortstoMaintainSecrecy','InfoValuable','InfoUsed','ConfidentialRelationship','NoticeOfConfidentiality', 'SecurityMeasures','RestrictedMaterialsUsed','UniqueProduct','IdenticalProducts','KnewInfoConfidential','InfoReverseEngineerable','InfoReverseEngineered']
laser = ['a trade secret was misappropriated, find for plaintiff','TradeSecretMisappropriation','EffortstoMaintainSecrecy','InfoValuable','InfoUsed','ConfidentialRelationship','NoticeOfConfidentiality','MaintainSecrecyOutsiders', 'SecurityMeasures','OutsiderDisclosuresRestricted','KnewInfoConfidential','DisclosureInNegotations','SecretsDisclosedOutsiders']
lewis = ['a trade secret was misappropriated, find for plaintiff','TradeSecretMisappropriation','EffortstoMaintainSecrecy','InfoValuable','InfoUsed','ConfidentialRelationship','NoticeOfConfidentiality', 'CompetitiveAdvantage','KnewInfoConfidential','DisclosureInNegotations']
mbl = ['no trade secret was misappropriated, find for defendant','LegitimatelyObtainable','EffortstoMaintainSecrecy','InfoKnownOrAvailiable','InfoKnown','InfoUsed','ConfidentialRelationship','NoticeOfConfidentiality','ConfidentialityAgreement','MaintainSecrecyDefendant', 'AgreedNotToDisclose','SecurityMeasures','NoncompetitionAgreement','AgreementNotSpecific','SecretsDisclosedOutsiders','InfoKnownToCompetitors']
mason = ['no trade secret was misappropriated, find for defendant','LegitimatelyObtainable','EffortstoMaintainSecrecy','InfoValuable','InfoKnownOrAvailiable','InfoAvailiableElsewhere','InfoUsed','ConfidentialRelationship','NoticeOfConfidentiality', 'SecurityMeasures','UniqueProduct','KnewInfoConfidential','DisclosureInNegotations','InfoReverseEngineerable']
mineralDeposits = ['a trade secret was misappropriated, find for plaintiff','TradeSecretMisappropriation','EffortstoMaintainSecrecy','InfoValuable','InfoUsed','ConfidentialRelationship','NoticeOfConfidentiality', 'IdenticalProducts','RestrictedMaterialsUsed','DisclosureInNegotations','InfoReverseEngineerable','InfoReverseEngineered']
nationalInstruments = ['a trade secret was misappropriated, find for plaintiff','TradeSecretMisappropriation','EffortstoMaintainSecrecy','InfoValuable','InfoUsed','ConfidentialRelationship','NoticeOfConfidentiality', 'IdenticalProducts','KnewInfoConfidential','DisclosureInNegotations']
nationalRejectors = ['no trade secret was misappropriated, find for defendant','InfoValuable','InfoUsed', 'BroughtTools','UniqueProduct','IdenticalProducts','SecretsDisclosedOutsiders','InfoReverseEngineerable','NoSecurityMeasures','DisclosureInPublicForum']
```

```
    reinforced = ['a trade secret was misappropriated, find for plaintiff','TradeSecretMisappropriation','EffortstoMaintainSecrecy','InfoValuable','InfoUsed','ConfidentialRelationship','NoticeOfConfidentiality','ConfidentialityAgreement','MaintainSecrecyDefendant', 'AgreedNotToDisclose','SecurityMeasures','CompetitiveAdvantage','UniqueProduct','KnewInfoConfidential','DisclosureInNegotations']
    scientology = ['no trade secret was misappropriated, find for defendant','LegitimatelyObtainable','EffortstoMaintainSecrecy','InfoKnownOrAvailiable','InfoKnown','InfoUsed','ConfidentialRelationship','NoticeOfConfidentiality','ConfidentialityAgreement','MaintainSecrecyDefendant','MaintainSecrecyOutsiders', 'AgreedNotToDisclose','SecurityMeasures','OutsiderDisclosuresRestricted','SecretsDisclosedOutsiders','VerticalKnowledge','InfoKnownToCompetitors']
    technicon = ['a trade secret was misappropriated, find for plaintiff','TradeSecretMisappropriation','EffortstoMaintainSecrecy','InfoValuable','InfoUsed','ConfidentialRelationship','NoticeOfConfidentiality','MaintainSecrecyOutsiders', 'SecurityMeasures','OutsiderDisclosuresRestricted','RestrictedMaterialsUsed','KnewInfoConfidential','SecretsDisclosedOutsiders','InfoReverseEngineerable','InfoReverseEngineered']
    trandes = ['a trade secret was misappropriated, find for plaintiff','TradeSecretMisappropriation','EffortstoMaintainSecrecy','InfoValuable','InfoUsed','ConfidentialRelationship','NoticeOfConfidentiality','ConfidentialityAgreement','MaintainSecrecyDefendant','MaintainSecrecyOutsiders', 'AgreedNotToDisclose','SecurityMeasures','OutsiderDisclosuresRestricted','DisclosureInNegotations','SecretsDisclosedOutsiders']
    valcoCincinnati = ['a trade secret was misappropriated, find for plaintiff','TradeSecretMisappropriation','EffortstoMaintainSecrecy','InfoValuable','InfoUsed','ConfidentialRelationship','NoticeOfConfidentiality','MaintainSecrecyOutsiders', 'SecurityMeasures','OutsiderDisclosuresRestricted','UniqueProduct','KnewInfoConfidential','DisclosureInNegotations','SecretsDisclosedOutsiders']

    cases = {'Arco Industries Corp v Chemcast Corp':arco, 'The Boeing Company v Sierracin Corp':boeing,'M. Bryce & Associates Inc v Gladstone':bryce, 'College Watercolour Group Inc v William H. Newbauer':collegeWatercolour,'Den-Tal-Ez Inc v Siemens Capital Corp':denTalEz,'Ecolgix Inc v Fantsteel Inc':ecolgix,'A.H. Emery Co v Marcon Products Corp':emery,'Ferranti Electric Inc v Harwood':ferranti,'Commonwealth v Robinson':robinson,'Sandlin v Johnson':sandlin,'Sheets v Yamaha Motors Corp':sheets,'Space Aero Products Corp v R.E. Darling Corp':spaceAero,'Televation Telecommunications Systems Inc v Saindon':televation,'Midland-
Ross Corp v Yokana':yokana,'CMI Corp v Jakob':cm1,'Digital Development Corp v International Memory Systems':digitalDevelopment,'FMC Corp v Taiwan Tainan Giant Ind Co Ltd':fmc,'Forest Laboratories Inc v Formulations Inc':forrest,'Goldberg v Medtronic':goldberg,'K & G Oil Tool & Services Co v G & G':kg,'Laser Industries Ltd v Eder Instrument Co':laser,'Computer Print Systems v Lewis':lewis,'MBL (USA) Corp v Diekman':mbl,'Mason v Jack Daniel Distillery':mason,'Mineral Deposits Ltd v Zigan':mineralDeposits,'National Instruments Labs Inc v Hycel Inc':nationalInstruments,'National Rejectors Inc v Trieman':nationalRejectors,'Reinforced':reinforced,'Scientology':scientology, 'Technicon Data Systems Corp v Curtis':technicon,'Trandes Corp v Guy F. Atkinson Co':trandes,'Valco Cincinnati Inc v N & D Machining Service Inc':valcoCincinnati}

    return cases
```

H.6 FourthAmendment.py

This file initialises the Fourth Amendment domain. adf() adds the node data to the ADF. cases() contains the base-level factors for each case in the test dataset. expectedOutcome() contains the expected outcome data for the test dataset.

```
from MainClasses import *

def adf():

  """
  creates the ADF for the domain
  """
  adf = ADF('FourthAmendment')

  #non leaf
  adf.addNodes('Decide',['Privacy','Exigency'],['warantless search violates the fourth amendment','warant-
less search did not violate the fourth amendment','wrong decision'])
  adf.addNodes('Privacy',['ExpectationOfPrivacyInUse and ( not SubjectToInspectionRegula-
tion or not VisibilityOfItem )','not PublicParking and not_authorised','OnlyVehicleContainer and not Visi-
bilityOfItem'],['high expectation of privacy not justified under automobile exception','high expecta-
tion of privacy obtained warrant was issued by neutral and detached magustrate and not autorized','pri-
vacy is not justified under automobile exception','reduced expectation of privacy'])
  adf.addNodes('Exigency',['( Mobile and ExigencyWhenApproached and ProbableCauseToSearchVehi-
cle )', 'reject EaseWarrant'],['justified under automobile exception cite carroll v us','reduce expecta-
tion of exigency','reduce expectation of exigency'])
  adf.addNodes('ExpectationOfPrivacyInUse',['( Accomodation and Residence ) or PrivateContentsCar-
riage'],['there is a high expectation of privacy in use','default low expectation of privacy in use'])
  adf.addNodes('Residence',['ConnectedServices'],['connected to one or more main living services','de-
fault not connected to one or more main living services'])
  adf.addNodes('PrivateContentsCarriage',['ProtectionType and GoodsCarried','reject GoodsCarried'],['pri-
vate contents','private contents but not protected','default contents are not considered private'])
  adf.addNodes('Accomodation',['AccomodationSpaces or RoomsFunction'],['the place was used for acco-
modation','default the place was used for accomodation'])
  adf.addNodes('SubjectToInspectionRegulation',['reject Licence and RestrictedArea','Licence'],['sub-
ject to regular inspection but the search was allocated at restricted area','subject to regular inspec-
tion','it is not subject to regular inspection'])
  adf.addNodes('VisibilityOfItem',['OnPublicView or CanBeSeen'],['item is visible to public','de-
fault it is not visible to public'])
  adf.addNodes('ExigencyWhenApproached',['UrgentStatus or ( CapableToMove and ( PublicPark-
ing or PublicLocation ) )'],['there was exigency when approached','there was no exigency when ap-
proached'])
  adf.addNodes('Mobile',['Automobile or Vessel or Towable or LargeContainer or MovableContain-
er'],['it is a mobile','default it is not a mobile'])
  adf.addNodes('EaseWarrant',['not RiskofLosingEvidence or ( AvailiabilityofMagistrate and Authori-
tyOfAvailiableMagistrate ) '],['it is easy to obtain a warrant','it is not easy to obain a warrant'])
  adf.addNodes('ProbableCauseToSearchVehicle',['LegalSearchScope and UrgentReason-
ToSearch and AuthorizedOriginOfProbableCause','reject UrgentReasonToSearch and AuthorizedOrigi-
```

```
nOfProbableCause'],['there is a probable cause to search the vehicle','there is a proba-
ble cause to search the vehicle but the search scope was illegal','default there is no proba-
ble cause to search the vehicle'])
  adf.addNodes('AuthorizedOriginOfProbableCause',['Information or Observation or Procedure','not Infor-
mation'],['there was an authorized origin of probable cause','default origin of probable cause is not author-
ized or not clarified',' '])
  adf.addNodes('UrgentReasonToSearch',['PublicSafety or Crime'],['the main reason to search was ur-
gent','default the main reason to immediate search is not clarified'])
  adf.addNodes('LegalSearchScope',['WholeVehicle','reject OnlyVehicleContainer and Search-
Place'],['the search scope is legal','the search scope is illegal','default the serach scope is illegal'])

  #blf
  adf.addMulti('Automobile',['car','mobile_home'],['it is a vehicle cite carroll v us','it is a mobile home ve-
hicle cite carney v california','default it is not an automobile cite carroll v us'],'If the vehi-
cle that was searched was an automobile select the correct type:')
  adf.addMulti('Vessel',['not Automobile and vessel','not Automobile and sailboat','not Automo-
bile and rowboat'],['motorboat is a vessel cite carroll v us','sailboat is a vessel cite carroll v us','row-
boat is a vessel cite carroll v us','default it is not a vessel cite carroll v us'],'If the vehi-
cle that was searched was a vessel select the correct type:')
  adf.addMulti('Towable',['trailer','wagon','cart'],['trailer is towable cite carroll v us','wagon is towa-
ble cite carroll v us','cart is towable cite carroll v us','default it is not towable cite carroll v us'],'If the vehi-
cle that was searched was towable select the correct type:')
  adf.addMulti('LargeContainer',['not Vessel and foot_locker','not Vessel and goods_container'],['foot-
locker is a large container cite us v chadwick','large goods container','default no large contain-
ers'],'If a large container was searched select the correct type')
  adf.addMulti('MovableContainer',['pouch','paper_bag','briefcase','suitcase'],['pouch is a movable contain-
er','paper bag is a movable container','brief case is a movable container','suitcase is a movable contain-
er','default it is not a movable container'],'If a movable container was searched select the correct type')
  adf.addMulti('AuthorityOfAvailiableMagistrate',['authorised','reject not_authorised'],['neutral and de-
tached authorized magistrate are availiabe cite johnson v us','warrant issued by unauthorized magis-
trate cite johnson v us','default authorized magistrate are not availiable'],'Is an authorised magistrate availi-
able?')
  adf.addNodes('RiskofLosingEvidence',['ExigencyWhenApproached','re-
ject near_court'],['there is risk to lose evidence','there was no risk to lose evidence','de-
fault there was no risk to lose evidence'])
  adf.addNodes('near_court',question='Was there a risk of losing evidence?')
  adf.addMulti('AvailiabilityofMagistrate',['working_time','reject overnight'],['magistrate availiabe dur-
ing working hours','magistrate are not availiable overnight','default magistrate are not availia-
ble'],'What is the availiability, if any, of the magistrate?')
  adf.addMulti('Licence',['vehicle','motorhome','Automobile'],['has a vehicle licence','has a special mo-
torhome licence','default all automobiles are registered','no automobile'],'What type of vehicle li-
cence, if any, was held?')
  adf.addMulti('RestrictedArea',['airport','home','police_station'],['airport is a restricted area','pri-
vate home is a restricted area','police station is a restricted area','default not restricted ar-
ea'],'Did the search take place in a restricted area?')
  adf.addNodes('OnPublicView',['OnSeat'],['items were on the seat it is on public view','de-
fault item is not on public view or details are not provided'])
  adf.addNodes('OnSeat',question="Where the items on the seat in public view?")
```

```
  adf.addMulti('CanBeSeen',['not OnPublicView and public_view','not On-
PublicView and on_floor'],['items can be seen by public','items were no floor it can be seen by public','de-
fault can not be seen by public or details are not provid-
ed'],'Were the items on the floor or in a place that could be seen by the public?')
  adf.addMulti('CannotBeSeen',['not CanBeSeen and opaque_container','not CanBeSeen and glove-
box','not CanBeSeen and boot'],['items were in an opaque container it can not be seen by pub-
lic','items were inside the glove box it can not be seen by public','items were in-
side the boot it can be not seen by public','de-
fault it is not clear that items can not be seen'],'Were the items not viewable to the public in one of the fol-
lowing?')
  adf.addMulti('UrgentStatus',['Mobile and moving','reject Mobile and stationary','reject Mo-
bile and parked','reject Mobile and crashed'],['there was an urgent status when vehicle is moving cite car-
roll v us','there was no urgent status automobile found stationary','there was no urgent status automo-
bile was parked','there was no urgent status automobile was crashed','default there is no urgent sta-
tus'],'Was the vehicle any of the following:')
  adf.addMulti('CapableToMove',['Mobile and not UrgentStatus and ( driver_in or occupied or cur-
tains_open or motive_force )', 'Mobile and not moving and not ( driver_in or occupied or cur-
tains_open or motive_force )'],['the vehicle is capable to move','default the vehicle is capa-
ble to move',' '],'Was the vehicle capable to move for any of the following reasons?')
  adf.addMulti('PublicParking',['( UrgentStatus or CapableToMove ) and ( parked_on_highway or ( park-
ing_lot and not dwelling ) )','reject ( UrgentStatus or CapableToMove ) and ( dwelling or ( ( own-
land or work or rented_land ) and not parking_lot ) )','reject ( UrgentStatus or CapableToMove ) and Ur-
gentStatus'],['the vehicle was parked in public parking','the vehicle was parked in private parking','de-
fault vehicle was not parked','default vehicle parking type is not specified'],'Where was the vehi-
cle parked, if anywhere?')
  adf.addMulti('PublicLocation',['( UrgentStatus or CapableToMove ) and ( highway or downtown )','re-
ject ( UrgentStatus or CapableToMove ) and ( dwelling or urban_residential or suburban or ru-
ral )'],['the vehicle was in public location','the vehicle was in private location','default vehicle loca-
tion is not specified'],'In what location was the vehicle?')
  adf.addMulti('PermittedDuration',['short_stay','overnight','long_stay'],['the vehi-
cle was parked for a short time','the vehicle was parked for one overnight','the vehi-
cle was parked for over one night long period','default the vehicle was parked for unknown peri-
od'],'For how long was the vehicle parked?')
  adf.addMulti('Information',['public_informant','agent_officer'],['received information from public inform-
ant','received information from agent officer','default the original probable cause is not by information re-
ceived'],'Did the original probable cause come from information received from:')
  adf.addMulti('Observation',['not Information and the_public','not Information and an_agent_of-
ficer'],['observed from public observer','observed from agent officer','default the original proba-
ble cause is not by observation'],'Was the original probable cause observed from:')
  adf.addMulti('Procedure',['not Observation and incident_to_arrest','not Observation and multi-
ple_parking','not Observation and inspection_regulation'],['search incident to arrest cite harris v us and pre-
seton v us','multiple parking procedure','inspection procedure cite harris v us and preston v us','de-
fault the original probable cause is not a procedure or procedure is not clarified'],'Was the original proba-
ble cause a procedure such as:')
  adf.addMulti('PublicSafety',['weapon and illegal_substance','weapon','illegal_sub-
stance','not weapon or not illegal_substance'],['main reason to search was to protect the public','main rea-
```

```
son to search was to protect the public cite harris v us and preston v us','main reason to search was to pro-
tect the public cite carol v us','default main reason to search was to protect the public','main rea-
son to search was not to protect the public'],'Was the main reason to search due to:')
  adf.addMulti('Crime',['smuggling or dealing or murder','robbery'],['main rea-
son to search was due to a crime','main reason to search was not due to a crime','default main rea-
son to search was not due to a crime'],'Had any of these crimes preempted the search?')
  adf.addNodes('WholeVehicle',['all_parts'],['all vehicle parts have been searched cite car-
rol v us and us v ross','default it is not clear if all vehicle parts have been searched'])
  adf.addNodes('all_parts',question='Have all the vehicle parts been searched?')
  adf.addMulti('OnlyVehicleContainer',['not WholeVehicle and ( car_trunk and glove_compart-
ment )','not WholeVehicle and car_trunk','not WholeVehicle and glove_compartment'],['only vehicle con-
tainers have been searched','only vehicle containers have been searched cite us v chadwick and arkan-
sas v sanders','only vehicle containers have been searched','default it is not clear which part of vehi-
cle is searched'],'Were any of the following parts of the vehicle searched?')
  adf.addMulti('SearchPlace',['police_station_location and automobile_location','reject police_station_loca-
tion','reject garage','automobile_location'],['the vehicle was searched twice at the same automobile loca-
tion and at police station','the vehicle was searched at police station','the vehicle was searched at a gar-
age','the vehicle was searched at the same automobile location','default the vehicle searching loca-
tion is not clarified'],'Where was the vehicle searched?')
  adf.addMulti('GoodsCarried',['personal_effects or papers or commercial_items','reject weapons or ille-
gal_substance','money'],['private goods','illegal goods','private goods','default goods carried are un-
known'],'Which, if any, of these goods were being carried?')
  adf.addMulti('ProtectionType',['reject open','reject closed','locked','double_locked'],['not protect-
ed','just closed but not protected','locked and protected','double locked and protected','default protec-
tion level can not be determined'],'What was the protection level of the goods?')
  adf.addMulti('ConnectedServices',['gas and water','electricity and water','gas','electrici-
ty','water'],['gas and water services were connected','electricity and water services were connected','gas ser-
vice was connected','electricity service was connected','water service was connected','default none of liv-
ing main services are specified'],'Were any of the following services connected?')
  adf.addMulti('AccomodationSpaces',['cab and suitable_accomodation_space','reject cab','suitable_acco-
modation_space'],['consists of a cab and suitable accomodation space','consists of a cab only','con-
sists of suitable accomodation space','default vehicle accomodation spaces are not clarified'],'Does the ve-
hicle accomodation consist of:')
  adf.addMulti('RoomsFunction',['not AccomodationSpaces and ( bedroom or bathroom or kitchen or liv-
ing_room )'],['essential room for accomodation','default there are no rooms or rooms function is not speci-
fied'],'Are there any rooms in the accomodation space?')

  #set to stop the program prompting it be set as a question but excluded from the question order
  adf.addNodes('not_authorised',question='<>')

  adf.questionOrder = ['Automobile','Vessel','Towable','LargeContainer','MovableContainer','Authori-
tyOfAvailiableMagistrate','near_court','AvailiabilityofMagistrate','Licence','RestrictedArea','On-
Seat','CanBeSeen','CannotBeSeen','UrgentStatus','CapableToMove','PublicParking','PublicLocation','Per-
mittedDuration','Information','Observation','Procedure','PublicSafety','Crime','all_parts','OnlyVehicleCon-
tainer','SearchPlace','GoodsCarried','ProtectionType','ConnectedServices','Accomoda-
tionSpaces','RoomsFunction']

  return adf
```

```
def cases():
  """
  test cases
  """
  cvus = ['car','moving','public_informant','illegal_substance','all_parts','automobile_location','illegal_substance'] #pass
  cvm = ['car','ft015w','moving','highway','inspection_regulation','robbery','all_parts'] #pass
  cvnh = ['car','not_authorised','parked','dwelling','dwelling','inspection_regulation','murder','all_parts'] #can't tell
  cvd = ['car','public_view','boot','crashed','parked_on_highway','dwelling','inspection_regulation','murder','all_parts','garage'] #pass
  sdvo = ['car','paper_bag','glovebox','parked','parking_lot','multiple_parking','illegal_substance','all_parts','automobile_location','illegal_substance'] #pass
  usvc = ['car','foot_locker','police_station','boot','parked','parking_lot','public_informant','illegal_substance','car_trunk','police_station_location','illegal_substance','double_locked'] #pass
  avs = ['car','goods_container','suitcase','airport','boot','moving','agent_officer','illegal_substance','car_trunk','illegal_substance','closed'] #pass
  usvr = ['car','paper_bag','police_station','parked','parking_lot','public_informant','illegal_substance','all_parts','car_trunk','automobile_location','police_station_location','illegal_substance','money','closed'] #pass
  cvc = ['mobile_home','paper_bag','near_court','motorhome','police_station','parked','driver_in','parking_lot','downtown','public_informant','the_public','illegal_substance','all_parts','police_station_location','automobile_location','illegal_substance','closed','cab','suitable_accomodation_space','bedroom','kitchen'] #pass
  cva = ['car','paper_bag','police_station','boot','moving','highway','public_informant','illegal_substance','car_trunk','automobile_location','illegal_substance','closed'] #pass

  cases = {'Carroll v United States':cvus,'Chambers v Maroneys':cvm,'Cady v Dombrowski':cvd,'South Dakota v Opperman':sdvo,'United States v Chadwick':usvc,'Arkansas v Sanders':avs,'United States v Ross':usvr,'California v Carney':cvc,'California v Acevedo': cva}#,'Coolidge v New Hampshire':cvnh}

  return cases

def expectedOutcomeCases():
  """
  first factor is the outcome - the other factors are those from the prolog program of the domain

  """

  cvus = ['warantless search did not violate the fourth amendment','car','moving','public_informant','illegal_substance','all_parts','automobile_location','illegal_substance','Exigency','RiskofLosingEvidence','SubjectToInspectionRegulation','Licence','ProbableCauseToSearchVehicle','LegalSearchScope','SearchPlace','WholeVehicle','UrgentReasonToSearch','PublicSafety','AuthorizedOriginOfProbableCause','Information','ExigencyWhenApproached','UrgentStatus','Mobile','Automobile'] #pass
```

```
  cvm = ['warantless search did not violate the fourth amendment','car','ft015w','moving','highway','inspection_regulation','robbery','all_parts','RiskofLosingEvidence','SubjectToInspectionRegulation','Licence','ProbableCauseToSearchVehicle','LegalSearchScope','WholeVehicle','UrgentReasonToSearch','Crime','PublicSafety','AuthorizedOriginOfProbableCause','Procedure','ExigencyWhenApproached','PublicLocation','UrgentStatus','Mobile','Automobile','Exigency'] #pass
  cvd = ['warantless search did not violate the fourth amendment','car','public_view','boot','crashed','parked_on_highway','dwelling','inspection_regulation','murder','all_parts','garage','RiskofLosingEvidence','VisibilityOfItem','CanBeSeen','SubjectToInspectionRegulation','Licence','ProbableCauseToSearchVehicle','LegalSearchScope','WholeVehicle','UrgentReasonToSearch','Crime','PublicSafety','AuthorizedOriginOfProbableCause','Procedure','ExigencyWhenApproached','PublicParking','CapableToMove','Mobile','Automobile','Exigency'] #pass
  sdvo = ['warantless search did not violate the fourth amendment','car','paper_bag','glovebox','parked','parking_lot','multiple_parking','illegal_substance','all_parts','automobile_location','illegal_substance','RiskofLosingEvidence','CannotBeSeen','SubjectToInspectionRegulation','Licence','ProbableCauseToSearchVehicle','LegalSearchScope','SearchPlace','WholeVehicle','UrgentReasonToSearch','PublicSafety','AuthorizedOriginOfProbableCause','Procedure','ExigencyWhenApproached','PublicParking','CapableToMove','Mobile','MovableContainer','Automobile','Exigency'] #pass
  usvc = ['warantless search violates the fourth amendment','car','foot_locker','police_station','boot','parked','parking_lot','public_informant','illegal_substance','car_trunk','police_station_location','illegal_substance','double_locked','RiskofLosingEvidence','ProtectionType','CannotBeSeen','RestrictedArea','Licence','OnlyVehicleContainer','UrgentReasonToSearch','PublicSafety','AuthorizedOriginOfProbableCause','Information','ExigencyWhenApproached','PublicParking','CapableToMove','Mobile','LargeContainer','Automobile','Privacy'] #pass
  avs = ['warantless search violates the fourth amendment','car','goods_container','suitcase','airport','boot','moving','agent_officer','illegal_substance','car_trunk','illegal_substance','closed','RiskofLosingEvidence','CannotBeSeen','RestrictedArea','Licence','OnlyVehicleContainer','UrgentReasonToSearch','PublicSafety','AuthorizedOriginOfProbableCause','Information','ExigencyWhenApproached','UrgentStatus','Mobile','MovableContainer','LargeContainer','Automobile','Privacy'] #pass
  usvr = ['warantless search did not violate the fourth amendment','car','paper_bag','police_station','parked','parking_lot','public_informant','illegal_substance','all_parts','car_trunk','automobile_location','police_station_location','illegal_substance','money','closed','RiskofLosingEvidence','RestrictedArea','Licence','ProbableCauseToSearchVehicle','LegalSearchScope','SearchPlace','WholeVehicle','UrgentReasonToSearch','PublicSafety','AuthorizedOriginOfProbableCause','Information','ExigencyWhenApproached','PublicParking','CapableToMove','Mobile','MovableContainer','Automobile','Exigency'] #pass
  cvc = ['warantless search did not violate the fourth amendment','mobile_home','paper_bag','near_court','motorhome','police_station','parked','driver_in','parking_lot','downtown','public_informant','the_public','illegal_substance','all_parts','police_station_location','automobile_location','illegal_substance','closed','cab','suitable_accomodation_space','bedroom','kitchen','RiskofLosingEvidence','Accomodation','AccomodationSpaces','RestrictedArea','Licence','ProbableCauseToSearchVehicle','LegalSearchScope','SearchPlace','WholeVehicle','UrgentReasonToSearch','PublicSafety','AuthorizedOriginOfProbableCause','Information','ExigencyWhenApproached','PublicParking','PublicLocation','CapableToMove','Mobile','MovableContainer','Automobile','Exigency'] #pass
  cva = ['warantless search violates the fourth amendment','car','paper_bag','police_station','boot','moving','highway','public_informant','illegal_substance','car_trunk','automobile_location','illegal_substance','closed','RiskofLosingEvidence','CannotBeSeen','RestrictedArea','Licence','SearchPlace','OnlyVehi-
```

```
cleContainer','UrgentReasonToSearch','PublicSafety','AuthorizedOriginOfProbableCause','Infor-
mation','ExigencyWhenApproached','PublicLocation','UrgentStatus','Mobile','MovableContainer','Automo-
bile','Privacy'] #pass

  #prolog for this case is not fully functional so cannot obtain an accurate expected list
  #cvnh = ['warantless search violates the fourth amendment','car','not_authorised','parked','dwell-
ing','dwelling','inspection_regulation','murder','all_parts']

  cases = {'Carroll v United States':cvus,'Chambers v Maroneys':cvm,'Cady v Dombrowski':cvd,'South Da-
kota v Opperman':sdvo,'United States v Chadwick':usvc,'Arkansas v Sand-
ers':avs,'United States v Ross':usvr,'California v Carney':cvc,'California v Acevedo': cva}#,'Coo-
lidge v New Hampshire':cvnh}

  return cases
```

H.7 test_data.py

Running this file initialises the unit tests as described in section 5.2.

```
import unittest
from MainClasses import *
import WildAnimals
import TradeSecrets
import FourthAmendment

#excluded due to data privacy concerns
#import NIHL


class Tests(unittest.TestCase):

   def test_wild_animals(self):

      adf = WildAnimals.adf()
      expOutcome = WildAnimals.expectedOutcomeCases()
      cases = WildAnimals.cases()

      self.query_adf(adf,expOutcome,cases)

      adf = WildAnimals.adf()
      expOutcome = WildAnimals.expectedOutcomeCases()
      cases = WildAnimals.cases()

      self.save_import(adf,"TestWildAnimals",cases,expOutcome)
```

```
def test_trade_secrets(self):

    adf = TradeSecrets.adf()
    expOutcome = TradeSecrets.expectedOutcomeCases()
    cases = TradeSecrets.cases()

    self.query_adf(adf,expOutcome,cases)

    adf = TradeSecrets.adf()
    expOutcome = TradeSecrets.expectedOutcomeCases()
    cases = TradeSecrets.cases()

    self.save_import(adf,"TestTradeSecrets",cases,expOutcome)

def test_fourth_amendment(self):

    adf = FourthAmendment.adf()
    expOutcome = FourthAmendment.expectedOutcomeCases()
    cases = FourthAmendment.cases()

    self.query_adf(adf,expOutcome,cases)

    adf = FourthAmendment.adf()
    expOutcome = FourthAmendment.expectedOutcomeCases()
    cases = FourthAmendment.cases()

    self.save_import(adf,"TestFourthAmendment",cases,expOutcome)

# def test_NIHL(self):

#     adf = NIHL.adf()
#     expOutcome = NIHL.expectedOutcomeCases()
#     cases = NIHL.cases()

#     self.query_adf(adf,expOutcome,cases)

#     adf = NIHL.adf()
#     expOutcome = NIHL.expectedOutcomeCases()
#     cases = NIHL.cases()

#     self.save_import(adf,"TestNIHL",cases,expOutcome)

def save_import(self,adf,filename,cases,expOutcome):

    #keeps track of the old adf for comparison
    adf_old = adf

    adf.saveNew('{}'.format(filename))
```

```
        file = "{}.xlsx".format(filename)
        adf_new = importADF(file,filename)

        #test nodes have the same name after import and that they are all present
        self.assertEqual(adf_old.nodes.keys(),adf_new.nodes.keys())

        #test the question order is the same in each
        self.assertEqual(adf_old.questionOrder, adf_new.questionOrder)

        #tests that the expected outcomes after the save/import are correct
        self.query_adf(adf_new,expOutcome,cases)

    def query_adf(self,adf,expectedOutcome,cases):

        for key,value in cases.items():

            #queries the case
            adf.evaluateTree(value)

            #removes the decide node from the outcome as not in the prolog
            try:
                adf.case.remove('Decide')
            except:
                pass

            #finds the expected outcome in the list of cases
            outcome = expectedOutcome[key]

            #removes the first item in the list which is a string of the expected outcome decision
            outcomeStatement = outcome.pop(0)

            #tests the outcomes are the same
            self.assertEqual(outcomeStatement,adf.statements[-1])

            #tests the individual fators are the same - converts each list into a set since order does not mat-
ter only equality
            self.assertEqual(set(adf.case),set(expectedOutcome[key]))

if __name__== '__main__':

    unittest.main()
```