Maastricht University

Group Project 2:

Project Report
Explainable AI: reasoning with natural language

created by

Inglot, Kamil - i6146717

He, Jingyi - i6242369

Tazwar, Mahir - i6251001

Abraham, Frederic - i6262598

Fernandez Ameijeiras, Ricardo - i6263028

Supervisors: Roos, Nico and Tintarev, Nava

Coordinator: Schoenmakers, Gijs

University of Maastricht

Data Science & Knowledge Engineering

Maastricht, June 23, 2021

Contents

1 Introduction **1**
- 1.1 Motivation . . . 1
- 1.2 Background Information . . . 1
  - 1.2.1 First-Order-Logic . . . 1
  - 1.2.2 Controlled Natural Language (CNL) . . . 2
- 1.3 Problem Statement and Research Questions . . . 3

2 Related Work **4**

3 Approach **7**
- 3.1 Semantic Tableau . . . 7
- 3.2 Tableau Solver Application . . . 8
  - 3.2.1 Tableau reasoner . . . 8
- 3.3 Implementation of Logical Rules . . . 11
- 3.4 Constraint Natural Language . . . 14
  - 3.4.1 Expressions . . . 14
  - 3.4.2 Sentence Parser . . . 16
- 3.5 Explainability . . . 17

4 Results **19**
- 4.1 Web Interface . . . 19
- 4.2 Examples . . . 20
  - 4.2.1 Propositional Logic . . . 20
  - 4.2.2 Syllogism . . . 22
  - 4.2.3 Predicate Logic . . . 24
- 4.3 Invalid Conclusion . . . 25
- 4.4 Limitations . . . 27

5 Discussion **30**
- 5.1 Addressing Research Questions . . . 30
- 5.2 Critical Analysis . . . 31
- 5.3 Future work . . . 32

6 Conclusion **33**

List of Figures **34**

Bibliography **35**

Abbreviations and terms

Abbreviations

CNL Controlled Natural Language

RTE Recognizing Textual Entailments

NLI Natural Language Inference

NLP Natural Language Processing

1 Introduction

1.1 Motivation

Logic reasoning plays an indispensable role in our daily life and work: it lies behind every critical-thinking, problem-solving, and decision-making process, assisting us to exploit and transform the readily available information into useful knowledge. Among the deluge of information, a fair amount is structured and expressed in a formal and standardized language, for instance, medical regulations, legal contracts, software specifications, and business rules. While it is common in practice for professionals and specialists to read through the information and extract arguments from it, manually examining all the information can be time-consuming, error-prone, and thus ineffective and expensive.

Naturally, we want to seek help and solutions from computers, to take advantage of AI to attain two goals: 1) correctness by forming a solid argument and deriving the right conclusions from the given argument; 2) explainability. Besides a valid conclusion, transparency and explainability are critical because, in practical applications, it is equally important for us to understand how the conclusion is made step by step, where mistakes occur when unreasonable results are produced, and what actions we are supposed to take to improve and perfect the reasoning process.

1.2 Background Information

1.2.1 First-Order-Logic

In the field of logic and reasoning, it is common to use predicate logic to derive arguments and examine whether a statement represented in predicate logic holds or not.

One of the methods that arise for reasoning is the semantic tableau method, originated by the Dutch logician Evert W. Beth. This technique attempts to interpret

the premises given and the negation of the conclusion in a way that every single one of them gets satisfied at the same time. If this fails, the conclusion must hold.

As earlier mentioned, the most extended way to develop this method is to utilize predicate logic, which eases the derivation of the logic. However, most of the information that we can find nowadays is expressed in natural language. This is not only for the regular life of a citizen, but also applies to the work space, for example, the law world. Even though this area employs a reformulated logic language, it is not clear enough for many lawyers to understand formal logic, which represents a struggle and burden for the overall population.

For this reason, the Netherlands has put a considerable effort into rewriting the Tax Law rules into the so-called *Regelspraak*, which constitutes a controlled natural language with a very restricted syntax. This has made the life of businesses and citizens much easier, since rules are represented in a natural language in such a way that these can be easily understood by people working with the tax law. Additionally, these rules are precise enough for a compiler to read and understand.

A natural language reasoner will be implemented, allowing the user to enter the desired clauses and hypotheses. After this step, the tree derived from the tableau method will be displayed. Closing the branches indicates a contradiction, which results in proving the formulae. For every intermediate step, the applied rule will be labeled and displayed, for instance a negation, or a disjunction. It will be one of the objectives of this project to extend the accepted natural language as much as possible, which will be done by the inclusion of tautologies, syllogisms or different rules such as De Morgan Law.

1.2.2 Controlled Natural Language (CNL)

Modern logics make use of a structured artificial language, for instance: **p is true, if p then q, therefore q is true.** The principle of valid inferences based on syntactic patterns has not changed. However, natural logics differ from modern logics by using a subset of natural language instead of an artificial formal language. As mentioned in the motivation, our aim is to reason with natural logic, which still focuses on identifying valid inferences based on syntactic patterns. Nevertheless, the natural logic reasoning is more complex than using simple arguments to prove or disprove conclusion.

First of all, we need to retrieve the underlying arguments from the natural sentences.

In order to do so, we can make use of a controlled natural language, which is a restricted form of natural language. The controlled natural language is intended to be the basis for formal and detailed natural language declarative description of a complex entity, such as structured language used in business or law.

There are several different standard approaches to provide a controlled natural language subset. We have yet to choose one for this project experiments.

We selected simple sentences at the beginning to experiment with the implemented semantic tableau, and gradually included more complex arguments, which we discuss more in 3.

1.3 Problem Statement and Research Questions

In this project, we want to investigate the possibility of reasoning with a controlled natural language, and our aim is to build a reasoning tool, in the form of a webpage interface, that can perform logical reasoning tasks phrased in a controlled natural language, and also provide explanations for the conclusions it derives.

Specifically, we attempt to address the following research questions:

- Which subset of natural language can be used to reason in semantic natural tableau method?
- How should the semantic tableau method be adapted to directly reason with natural language?
- To what extent must the natural language structure be specified and constrained for the semantic tableau method in order to reason with it?

2 Related Work

The research in reasoning with natural language dates back to 450 BC when Aristotle introduced syllogistic reasoning. A syllogism is a logical argument that applies deductive reasoning to arrive at a conclusion based on two or more propositions that are asserted or assumed to be true.[1] Exemplary, a reasoning pattern introduced by Aristotle is: all A are B, all B are C, therefore all A are C.

In this field of research the task of Natural Language Inference (NLI), which is also known as Recognizing Textual Entailments (RTE), emerged. "[It] is the task of defining the semantic relation between a premise text p and a conclusion text c. [The Premise] p can a) entail, b) contradict or c) be neutral to [the conclusion] c" (Kalouli et al. 2019). (Abzianidze 2015) considers the RTE task as a complex and fundamental NLP task.

A major step in the task to solve the RTE problem is the development of a natural logic. "Natural Logic attempts to do formal reasoning in natural language making use of syntactic structure and the semantic properties of lexical items and constructions" (Karttunen 2015). One semantic structure, for example, is the relationship between two nouns. A 'more specific' and 'more general' relation between fruit and apple can be defined. These relationships are called monotonicity relations which are not present only in nouns, and can be extended to every expression of any syntactic category (Karttunen 2015).

On top of the history of natural logic, (Muskens 2010) defined terms that are adequate both from a linguistic and logical point of view. These terms, so called Lambda Logical Forms (**LLFs**), are built up from variables and non-logical constants, with the help of application and lambda abstraction (Muskens 2010). Based on the LLF's, (Muskens 2010) defined a tableau system that works directly on these LLF's.

Muskens mentions that his tableau approach requires, contrary to the usual handful rules, many rules connected with special classes of expressions. (Abzianidze 2015) provides exemplary in figure 2.1 tableau rules for the the quantifiers ($\forall_F$

1. Reference: Syllogism.

$$\frac{X\ A\ B:\ [\,]:\ \mathbb{F}}{\begin{array}{c}A:\ [c]:\ \mathbb{T}\\ B:\ [c]:\ \mathbb{F}\\ \text{s.t. } X \in \{\texttt{all}, \texttt{every}\}\\ \text{and } c \text{ is a } \textit{fresh} \text{ term}\end{array}}\forall_F \qquad \frac{A\ B:\ [\vec{C}]:\ \mathbb{X}}{A:\ [B,\vec{C}]:\ \mathbb{X}}\text{PUSH} \qquad \frac{A:\ [B,\vec{C}]:\ \mathbb{X}}{A\ B:\ [\vec{C}]:\ \mathbb{X}}\text{PULL}$$

$$\frac{\begin{array}{c}A:\ [\vec{C}]:\ \mathbb{T}\\ B:\ [\vec{C}]:\ \mathbb{F}\end{array}}{\begin{array}{c}\times\\ \text{s.t. } A \leq B\end{array}}\leq\times \qquad \frac{X\ A\ B:\ [\,]:\ \mathbb{F}}{A:\ [d]:\ \mathbb{F} \qquad B:\ [d]:\ \mathbb{F}}\exists_F \quad \begin{array}{l}\text{s.t. } X \in \{\texttt{some}, \texttt{a}\} \text{ and } d \text{ is an } \textit{old} \text{ term}\end{array}$$

$$\frac{\texttt{not}\ A:\ [\vec{C}]:\ \mathbb{X}}{A:\ [\vec{C}]:\ \overline{\mathbb{X}}}\text{NOT}$$

Figure 2.1: Semantic tableau rules for utilising LLFs

and $\exists_F$), Boolean operators, formatting (*PUSH* and *PULL*) and inconsistency ($\leq \times$).

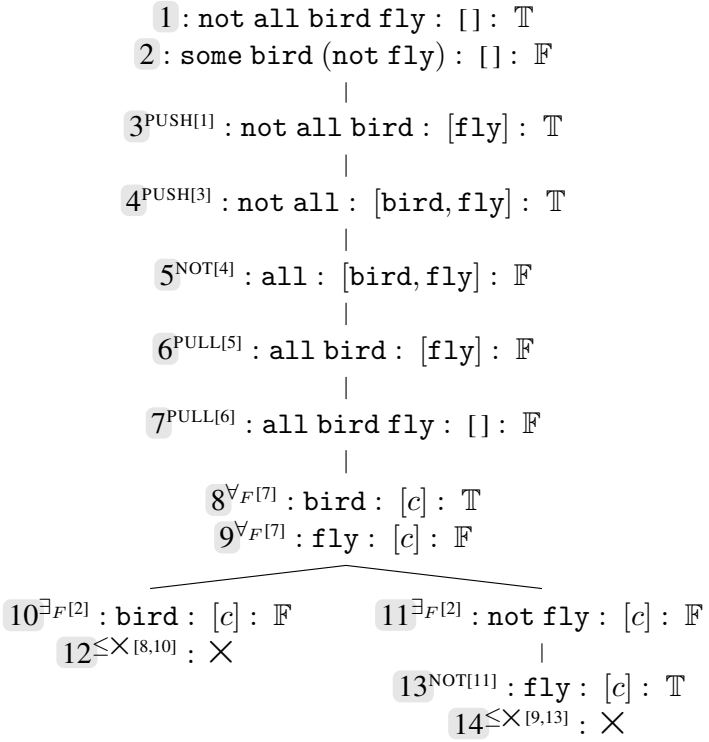


Figure 2.2: Semantic tableaux rules applied to the entailment:
not all birds fly → some bird does not fly

A tableau proof which utilizes (Muskens 2010) rules of figure 2.1 is exemplary given in figure 2.2. It derives the entailment "not all birds fly → some bird does not fly". The tableau method starts with assuming the premise being true and the conclusion being false. Then by repeatedly applying the rules to the arguments, it tries to close the branches by introducing an inconsistency.

Other interesting approaches regarding how to solve syllogisms include the use of *Term Functor Logic* (TFL) such as in (Castro-Manzano 2018), which is a plus-minus calculus dealing with syllogism by using terms rather than first order language elements, i.e quantifiers or individual variables. The solving method for the

aforementioned paper is still by the use of tableau in which they gather that a conclusion is following on a valid way from a set of premises if (1) the sum of the premises is algebraically equal to the conclusion and (2) the number of conclusion with particular quantity is the same as the number of premises with particular quantity.

3 Approach

3.1 Semantic Tableau

As mentioned, we implemented a semantic tableau method on natural language directly contrary to the usual approach mentioned in chapter 2. We start by taking a look at propositional logic and fit the constrained natural language to it. With the standard semantic tableau rules given in figure 3.1 we try to derive rules that work directly on the Controlled Natural Language (CNL)

$$\frac{\varphi \wedge \psi}{\varphi, \psi} \qquad \frac{\varphi \vee \psi}{\varphi \mid \psi} \qquad \frac{\varphi \rightarrow \psi}{\neg\varphi \mid \psi} \qquad \frac{\varphi \leftrightarrow \psi}{\varphi \rightarrow \psi, \psi \rightarrow \varphi}$$

$$\frac{\neg(\varphi \vee \psi)}{\neg\varphi, \neg\psi} \qquad \frac{\neg(\varphi \wedge \psi)}{\neg\varphi \mid \neg\psi} \qquad \frac{\neg(\varphi \rightarrow \psi)}{\varphi, \neg\psi} \qquad \frac{\neg(\varphi \leftrightarrow \psi)}{\neg(\varphi \rightarrow \psi) \mid \neg(\psi \rightarrow \varphi)}$$

$$\frac{\neg\neg\varphi}{\varphi} \qquad \frac{\varphi \barwedge \psi}{\neg\varphi \wedge \psi \mid \varphi \wedge \neg\psi} \qquad \frac{\neg(\varphi \barwedge \psi)}{\varphi \wedge \psi \mid \neg\varphi \wedge \neg\psi}$$

Figure 3.1: Semantic tableau rules for propositional logic

For example the rule containing the premise $\varphi \wedge \psi$ can be directly translated into the rule with the premise "s_1 **and** s_2" where s_1, s_2 are two sentences containing a enclosed sentiment. This approach can be extended to every rule with the use of the CNL.

The first step therefore is to implement a reasoner that uses the semantic tableau method using these derived rules. Our focus in this project is not on the theory for how to define logically and completely correct a semantic tableau method but on the implementation of a software that is able to reason with natural language. We used the programming language Python as it provides a variety of text manipulation and modification tools.

Our approach is to have a semantic tableau method that can at first reason with a simple propositional logic, as we extended our reasoner to include more complex

sentences. The next steps include being able to reason with all syllogisms in order to implement predicate logic or first order logic which introduces quantifiable variables.

3.2 Tableau Solver Application

This section addresses the main information about the application of our tableau solver to reason with the natural language, including the description of the rules implemented.

The applied semantic tableau can take sentences as input, the language must be constrained by some specific rules, so that the application can understand the essential parts such as: premise, negation, quantifiers, etc. Our application gives examples for each possible sentence structure, as well as a tool to examine whether given sentence can be applied. The tool also informs us on the sentence structure, by dissecting it and displaying each phrase, their components and its base information dependent on which semantic rules may be applied. Figure 3.2 demonstrates the applications workflow.

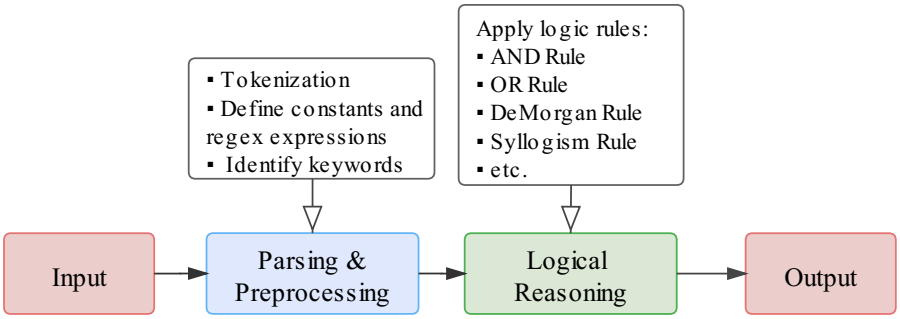


Figure 3.2: Application workflow

3.2.1 Tableau reasoner

The implemented tableau reasoner takes the correctly parsed sentences as input. Their arguments are considered to be true, additionally the last provided sentence is the argument to be proven as a true or false by all provided semantic arguments.

Algorithm 1 Natural Tableau Reasoner

```
1: procedure PROOF(sentences, toBeShown)
2:     clauses ← empty []
3:     for each sentence do
4:         parsedSentence ← SentenceParser(sentence)
5:         add parsedSentence to clauses
6:     parsedToBeShown ← SentenceParser(toBeShown)
7:     add negated parsedToBeShown to clauses
8:     result ← RecursiveProof(
           clauses
           applied rules ← []
           list of new objects ← []
       )
9:     return result
```

The entry point to the recursive solving algorithm is given in algorithm 1. The first step is to parse each given sentence into the created expression structure discussed in section 3.4.1.

Given the sentence containing the concluding argument we want to prove, we have to negate the meaning and also add it to the clause list. The following recursive proof requires the list of clauses in which it searches for contradictory information. Further it requires a list of already applied rules in order to not repeat the same rule on the already seen expression. Moreover, some syllogism and quantification rules specified in section 3.3 require the creation of a new object. In order to keep track of which object was created in each branch we pass a list of the created objects into each recursive call.

The main algorithm given in algorithm 2 constructs through its recursive calls a tree and uses its branches to prove or refute the main argument. It consists of two parts: the first is the search for contradictory information in the clauses of the current branch and the second branch is the search for an applicable rule.

The first part from line **2** to **6** shows the algorithm that goes over all pairs of clauses in the current branch and checks whether the two clauses are meaning the opposite of one another. In line **4** we check the correct type of the two expressions as we are only able to directly compare the base expression and the function expression explained in section 3.4.1.

The first check for opposite information informs us on whether the two expressions negate each other, therefore creating a contradiction. Following is the check for equality that goes over all the stored information in each respective expressions and

Algorithm 2 Recursive Tableau Proof

```
1:  procedure RecursiveProof(clauses, applied rules, list of new objects)
2:      for each reference clause in clauses do
3:          for each compare clause in clauses do
4:              Check for the correct types of the two clauses
5:              if reference clause is opossit of compare clause then
6:                  return True

7:      for each rule do
8:          for each clause in clauses do
9:              Check whether the rule has been used on the specific clause
10:             branches ← rule (
                    clause,
                    clauses,
                    list of new objects
                )
11:             if branches are not empty then
12:                 add (clause id, rule) to applied rules
13:             if just one branch then
14:                 Add the expressions of the branch to the clauses
15:                 return RecursiveProof(...)
16:             closes ← True
17:             for each branch in branches do
18:                 Add the expressions of the branch to the clauses
19:                 branch closes ← RecursiveProof(...)
20:                 if branch closes is False then
21:                     closes ← False
22:             return closes
23:     return False
```

compares them. If any word with its position in the expression is not equal to its counterpart in the compared expression the check returns false.

For a previously quantified expression it could be the case that a variable was replaced by a unification variable, which represents any kind of information. If we try to match a word with a unification variable we have to replace the free variable with the compared word. In the case of comparing two unification variables, a new object is introduced. Finally we do not only return whether the reference expression is the opposite of the compared expression we also return all replacements that were needed, in order to match the two expressions.

The second part from line **7** to **20** shows the generation of new information through the application of defined rules further specified in section 3.3. The algorithm goes over each rule and tries to apply it with each clause. In order to not generate

already present clauses we store the applied rule with the identification number of the used clause in the list of applied rules for the current branch and therefore check beforehand whether the rule, ID combination is present in the list. As it is possible to apply the same rule in two parallel branches, the applied rules list resets at every return to the state it was before the recursive call the same way the clauses list does.

We check whether the rule is applicable or not in the definition of each rule respectively. The applicable test checks for the correct expression type of the sentence, and whether the proper keywords are at their correct positions in the sentence. If that is not the case, an empty branch list is returned and we continue with the next clause or rule. Following, we differentiate between the amount of branches returned by the rule. When we only have one branch, we can immediately return a recursive call that adds the information to the current branch list and searches for tautology with the added information. Otherwise we go over each created branch and check whether the recursive call for each branch closes. If all the branches close, we may confirm the provided arguments contradict with the negated information from the beginning, which proves finally proves it. Any open branch refutes the concluding statement.

The search for opposite information runs in a $O(n^2)$ run-time where n is the amount of clauses in the current branch. The application of the rules run in $O(n \cdot m)$ run-time where m is the amount of rules in the application. The overall run-time depends on the depth h and branching factor b which results to $O(hb \cdot (n^2 + nm))$

3.3 Implementation of Logical Rules

The semantic tableau performs different actions depending on the rules applied to its arguments. Generally in order to have a modular system for the rules, they all have to implement the same rule interface.

We categorize the rules into the ones coming from propositional logic, the rules used in syllogisms and lastly the rules used in predicate logic.

Figure 3.3 shows the implemented rules coming from the propositional logic that can be further divided into the expressions connected through the **and** or **or** keyword and the compound expressions with premise and conclusion.

1. **And Rule** - The conjunction of two formulae, add to the leaf the chain of two

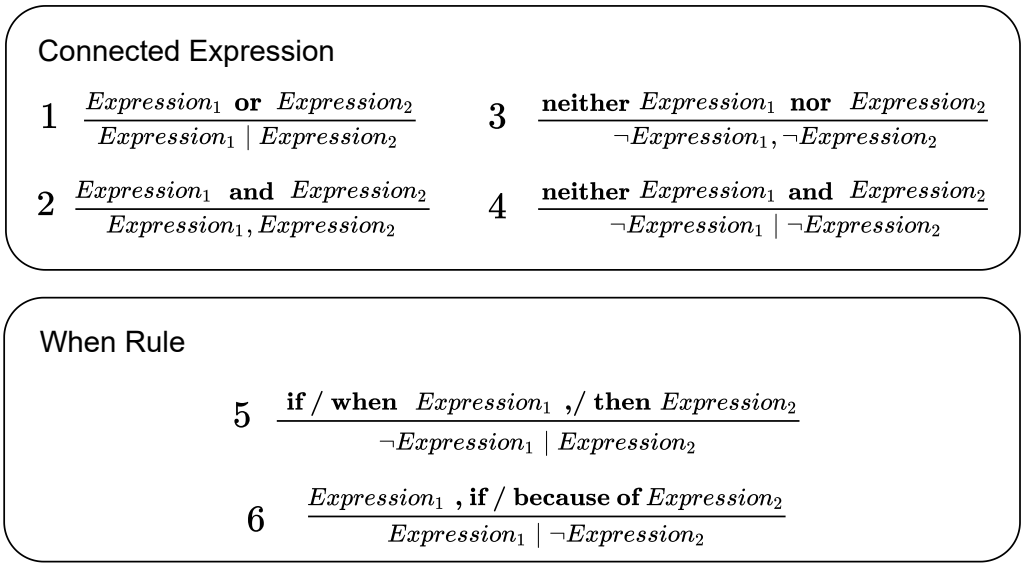


Figure 3.3: Rules coming from the propositional logic

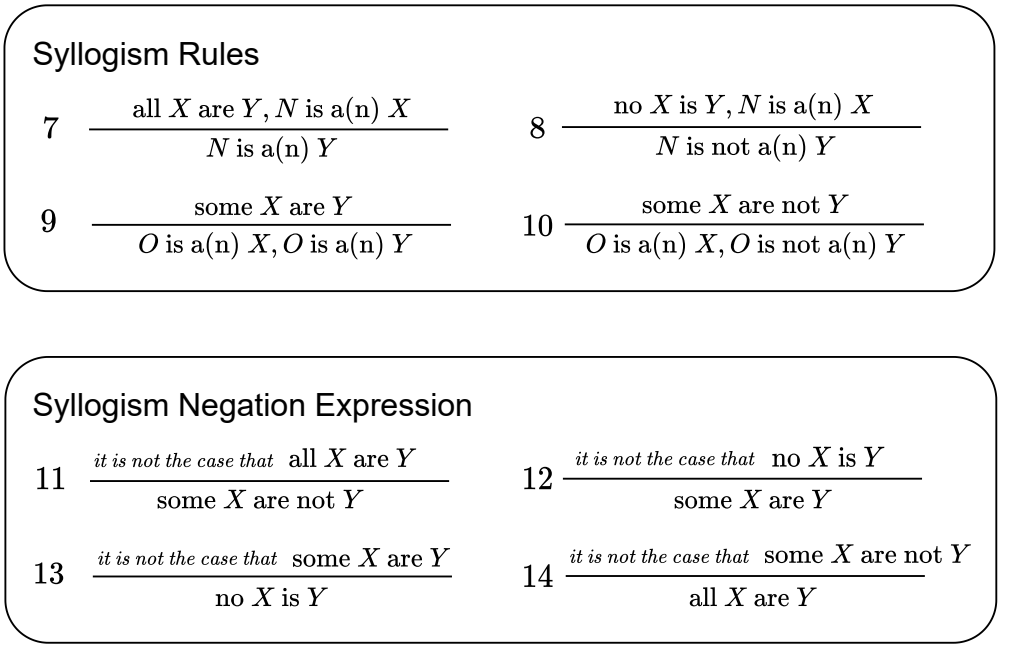


Figure 3.4: Rules used in syllogism

nodes containing both arguments ($A \wedge B$), both arguments must be true in the branch, if there is a counterargument, the branch closes.

2. **Or Rule** - The disjunctive formula ($A \vee B$), create two sibling children to the leaf of the branch, both new branches contain two separate arguments respectively.

3. **When Rule** - Semantic "When A then B" in logic is equal to $A \Rightarrow B$ which

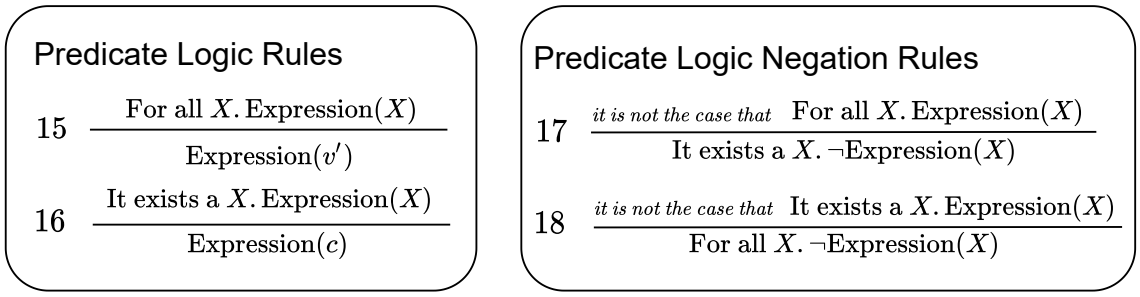


Figure 3.5: Replace rules from the predicate logic.

logically is equivalent to $\neg A \vee B$, similar to OR rule, creates two new branches as leaf nodes with $\neg A$, B, respectively.

4. **De Morgan's laws Rule** - Takes into account cases of $\neg(A \wedge B) \equiv (\neg A \vee \neg B)$ and $\neg(A \vee B) \equiv (\neg A \wedge \neg B)$ and transforms them to their logical equivalences.

Figure 3.4 illustrates the four **Syllogism rules** we implemented, moreover, we provided a separate extra rule to deal with quantified expressions. We focus on syllogisms that consist of two premises and one conclusion. The syllogism rules differ from the rest by producing new expressions from the provided information, for instance, given the sentence "all philosopher are person" one of the syllogism rules creates a new expression "there is a philosopher" to produce a new "philosopher" object and remember it for future mentions. Below we introduce the exact implemented rules for syllogism:

- **Syllogism Rule 1** - Uses the fact of "All X are Y, unit U is X, therefore U is Y". In other words the first rule creates a link between two facts and attaches the right unit to the object.
- **Syllogism Rule 2** - Takes into consideration sentences with "some", for instance "some X are Y" or "some X are not Y". The output from the given rule becomes brand new unit $U(0)$ to link objects X and Y. The main difference between rule 1 and 2 is the keyword "some", which considers both objects to be the same just in one case, whereas the first rule considers both objects to be the same for all cases.
- **Syllogism Rule 3** - The negation of rule 1. "No X is Y, unit U is X, therefore U is not Y", this holds for all cases.
- **Syllogism Rule 4** - Reverses the conclusion to provide invalid information, if it doesn't hold the branch can be closed to complete the proof verification.

After implementing the rules for syllogism the next steps are the rules for predicate logic. The set of rules is again separable into the general rules and the negation rules, which can be seen in 3.5.

The standard replacement rule [15] for *for all* quantified expressions replaces the quantified variable X with any ground term or in our case a piece of information. With *Expression*(X) being any kind of expression covered by the constrained natural language. As it is replaced with an arbitrary term it is also possible to replace it with a unification variable that is further replaced with the respective ground truth when we try to match the expression in search for the opposite meaning.

- **Quantification** - Used for predicate logic, takes into consideration sentences with quantifiers such as "for all X", and "exists such X as" - for instance "for all X, if X is a bird, then it can fly, X is an eagle, eagle is a bird, therefore it can fly" When "for all" is found, the rule keeps track of the current quantification, after decomposing the arguments it looks for any specific sentence that fits with the premise for all X in order to unify it.

3.4 Constraint Natural Language

The constraint of our language does not prohibit the use of any specific word but the general structure of possible sentences with the use of some fixed keywords identifying distinct expressions. We identified 6 different classes of expressions that either encapsulate information or connect different pieces of information to put them into relation to one another. The connecting expressions can also combine other encapsulating expressions to create more complex sentences as for example when two conditional expressions are connected through an "and". We developed an algorithm that recursively detects the encapsulated expressions and stores the relevant information in expression objects used in the rules of the semantic tableau reasoner.

3.4.1 Expressions

The two information holding expressions are either the base expression or function expression. In the following visualizations, a green block represents the location of a required keyword while a red block represents the possible location of a negation keyword.

The base expression represented left in figure 3.6 encapsulates information about the object. No specific keyword is used to detect this structure. Only in order to detect whether the expression is meant in a negative meaning, special negation keywords

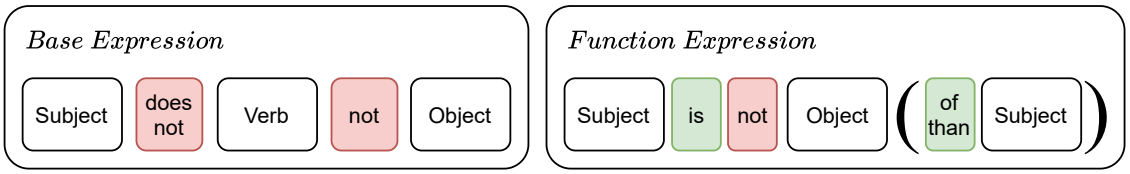


Figure 3.6: General structure of base expression and function expression.

such as: *do not*, *does not*, *not*, *never*, *dont*, *don't*, *doesnt* and *doesn't* are used. It usually describes an activity of the subject or captures a specific state that does not describe the subject as a specific object. We also extend the range of possible sentences with a list of filler words - for example *the*, *must* or *a* - that can be present at any location in the base expression and modify the information to be matched accordingly.

Contrary the function expression visualized on the right in figure 3.6 describes that the subject **is** something specific. When also using the second optional part *(of/then) subject* it describes the subject being something of another subject or puts the two subjects into relation. It is distinctly different to the base expression as the subject and object are separated by the keywords: *is the*, *is an*, *is a*, *is* and *are* while the subject itself is only allowed to be one specific word.

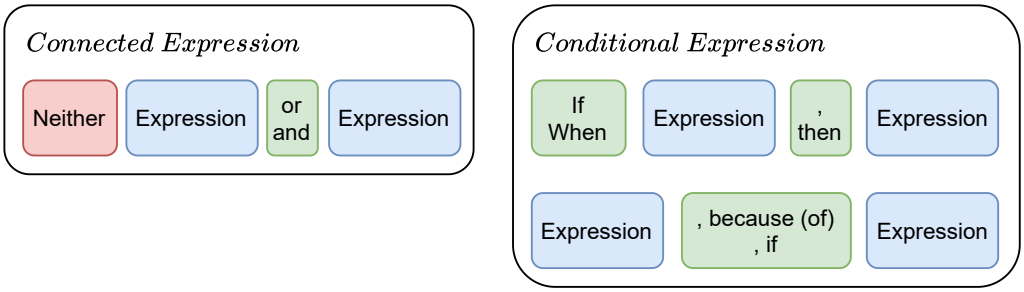


Figure 3.7: Structure of the connected expression and conditional expression.

In figure 3.7 the structure of the connected expression and both cases of the conditional expression are visualized. The *Expression* block marked in blue, doesn't only represent information holding expressions, but also represents any kind of recursive expression. One exception to this rule is when we try to put a conditional expression in between the keywords of another conditional expressions as this would require the detection of opening and closing keywords which would make the algorithm that detects the sentence structure more complicated.

The last expressions, that are necessary to make the step into predicate logic are the syllogism expressions and quantified expression given in figure 3.8. While the quantified expression can encapsulate other kind of expressions the syllogism expressions are as mentioned its own separated part of logic as they don't encapsulate any

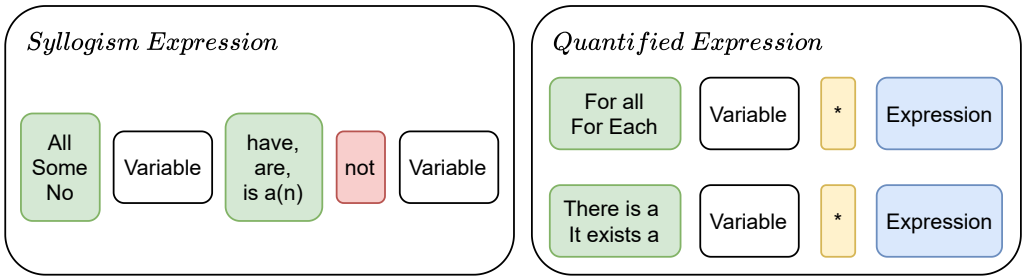


Figure 3.8: Structure of the syllogism expressions and quantification expression.

expressions and there are no specific rule in order to create other type of expressions besides function expressions. The quantification expression utilises an optional keyword that separates the quantification variable and the expression. If in the expression the variable is used in any type of base expression or function expression it will be replaced by the respective quantification rule mentioned in section 3.3. Otherwise the quantification will just be dropped in the replacement rule.

Generally for all the mentioned expressions we designated the phrase: *It is not the case that* as a general negation expression that you can put in front of every expression regardless of the internal negation. When detected, the parser removes the phrase, sets the negation flag accordingly and for the function expression reverses the used *it* keyword into the corresponding negative type. In case of a double negation the keyword is replaced with its corresponding positive type.

3.4.2 Sentence Parser

In order to detect the correct expressions in the sentences we define keywords for every kind of expression. Using these keywords we generate regular expressions that detects the existence of the required keywords for a specific expression in the sentence. We therefore create pairs consisting out of the sentence parser and corresponding regular expression. In order to be able to extend the language model our algorithm covers, we put the pairs into a list and iterate over them trying to match the regular expression. A found match results in the sentence parser making an attempt in parsing the sentence and extracting the required information. If any type of expected error occurs, a special kind of exception is raised to differentiate regular exceptions. If any kind of exception is caught in the matching part of the application, we continue with the next pair. As the base expression doesn't use any kind of keywords to match the type of expression we simply try to parse every sentence as a base expression.

The sentence parser for all connecting type of expressions uses the regular expression

to detect what keyword was used and in which location in the sentence they are written. Usually the keywords are stored in the expression and used in the respective rule. The part of the sentence between the keywords is recursively put into the algorithm that creates the expressions. This way we are able to capture complex and encapsulated expressions.

To compare each piece of information in the semantic tableau one major step is the tokenization of the input sentence, that are detected as expressions that hold information, into individual words. For each tokenized word we try to unify them as far as possible without destroying distinct different meaning. A simple example where a equal operation would fail is: *for all x, x plays football* and *I play football*. The different conjugations of **play** don't match and the tableau reasoner would fail. The same problem occurs in matching plural and singular cases of a noun. For example *all chairs are furniture* and *x is a chair* would result in no match between **chairs** and **chair**. The equalization method employs a special library that can detect the type and case of a specific word and puts it into a default state so the matching works with a simple equals operator.

The function expression, as a information holding sentence, falls also in to the category of expressions, where we are able to use regular expression matches to parse the sentence. Contrary, the approach to parsing the base expression comes down to checking whether we have exactly three tokens long expression as we only require the subject, verb and object. In order to accommodate not only for three token long sentences we add to the allowed length the amount of occurring filler words defined earlier in the expression section. 3.4.1 We also remove, if present, the used negation keyword or keywords. As follows, if the check for the correct length, taking the filler words into account, was successful, we split the sentence into the subject, verb and object section. We basically split the cleaned sentence at the tokens that are not defined in the filler word list. This approach enables us to capture a more natural way of writing sentences. It is easily extendable as to what filler words and negation tokens are used so we can modify what based expressions are allowed.

3.5 Explainability

Explainability is one major aspect of this research project. The minimum required output for the task is whether the conclusion holds or not. This would be simply a positive or negative return if all the generated branches are closed properly. To receive a more detailed explanation over the applied rules, we extend our semantic tableau implementation with a tree generator that creates nodes at each call of

the recursive function. We represent the reasoning process in the DOT language and visualize the parsed tree either in the web page or the local implementation using respective rendering engines for the DOT language. This provides a detailed explanations as to how the reasoning develops.

The explainability reflects in the following aspects:

- The language checker is capable of categorizing the input sentences into different types of logical expressions — basic information, compound expression consists of two or more sentences with basic information, uni-function expressions, multi-function expressions, syllogisms, or instances that are overly complicated, vague, or mistaken that our program cannot deal with. The parsing results are illustrated within a colorful box with every element of an expression clearly identified and highlighted, and thus it is easily understandable.
- The reasoner not only provides a conclusion (whether an input statement holds or not), it also prints out the corresponding semantic tableau, which shows which rule it applies at every node and whether a branch is closed or not. The tableau well explains steps leading to the conclusion the reasoner has made, therefore, the reasoning process is transparent and explainable.
- To increase the explainability of our implementation, we have designed and created a tooltip for the tableau tree. When a user's cursor hovers over a node, a text box will show up explaining what logical rule is applied at the node and how the rule works. And with the tooltip and explanatory information, one can read and understand the reasoning without difficulty.

4 Results

In this chapter, we first present the web interface we designed and introduce the functions and features it has, in section 4.1. Next, we give example results and explain how the reasoner perform each type of task in section 4.2 -4.3. Lastly, we discuss the limitations of our implementation and provide instances that our reasoner cannot handle in section 4.4.

4.1 Web Interface

As part of our incentive for explainability and usability we developed a web interface in the VUE framework. Figure 4.1 gives a general impression.

Figure 4.1: Web Interface

This includes the main part of the application, the interface to the reasoner and as mentioned in the explainability section the language checker. Each providing a variety of examples that show different aspects of how to use the reasoner and that give a feel for what constraint language we consider.

The main requirements are:

- An intuitive layout and simple interactions.
- An interactive interface, a user can add or delete expressions using the " + " or " − " button, and customize various problems for the reasoner to solve.

4.2 Examples

As we have described, we have implemented the rules coming from propositional logic, the rules used in syllogisms and the rules used in predicate logic. Within the constraints of these rules we have tested a variety of different examples in order to check the limit of our reasoner. Below some examples and their associated output from the reasoner are illustrated.

4.2.1 Propositional Logic

Example 1: Simple

INPUT

John plays football or chess
When it is raining, John does not play football.
It is raining.
Conclusion: John does not play football.

OUTPUT

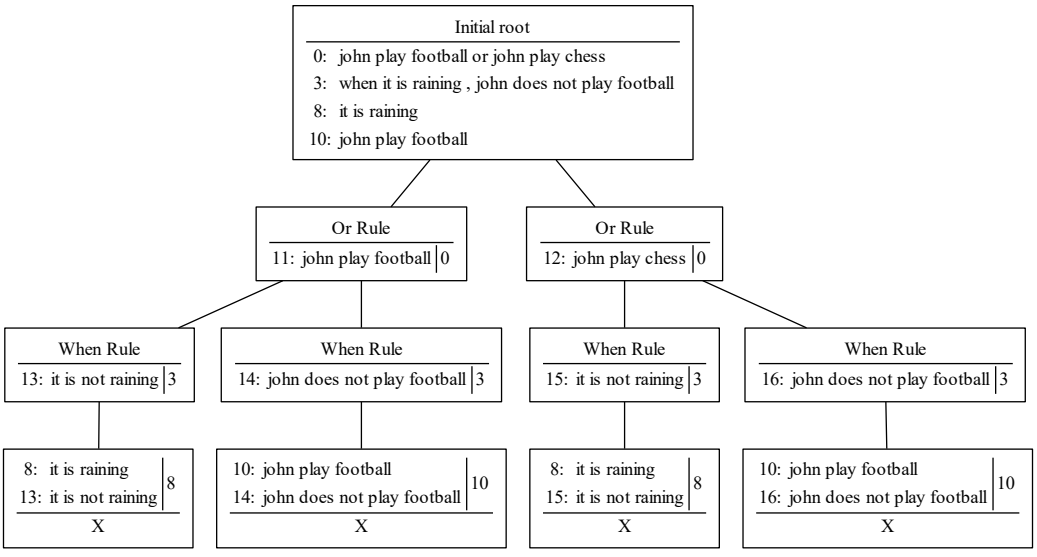


Figure 4.2: Output for the propositional logic example 1

Example 2: Negation

INPUT

When neither I visit him nor you visit him then he becomes sad
I do not visit him and you do not visit him
Conclusion: He becomes sad.

OUTPUT

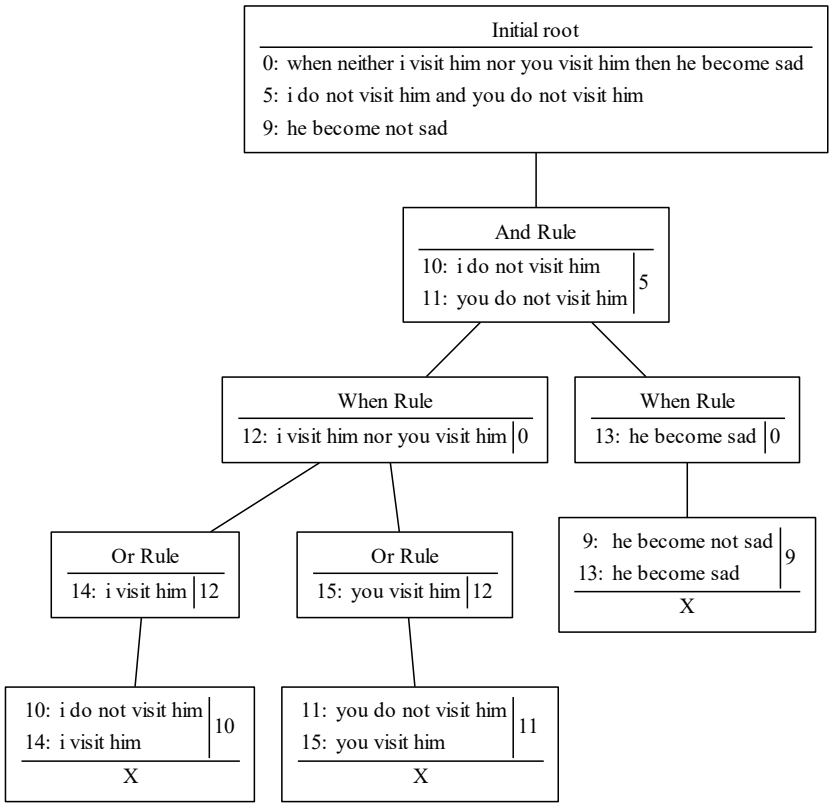


Figure 4.3: Output for the propositional logic example 2

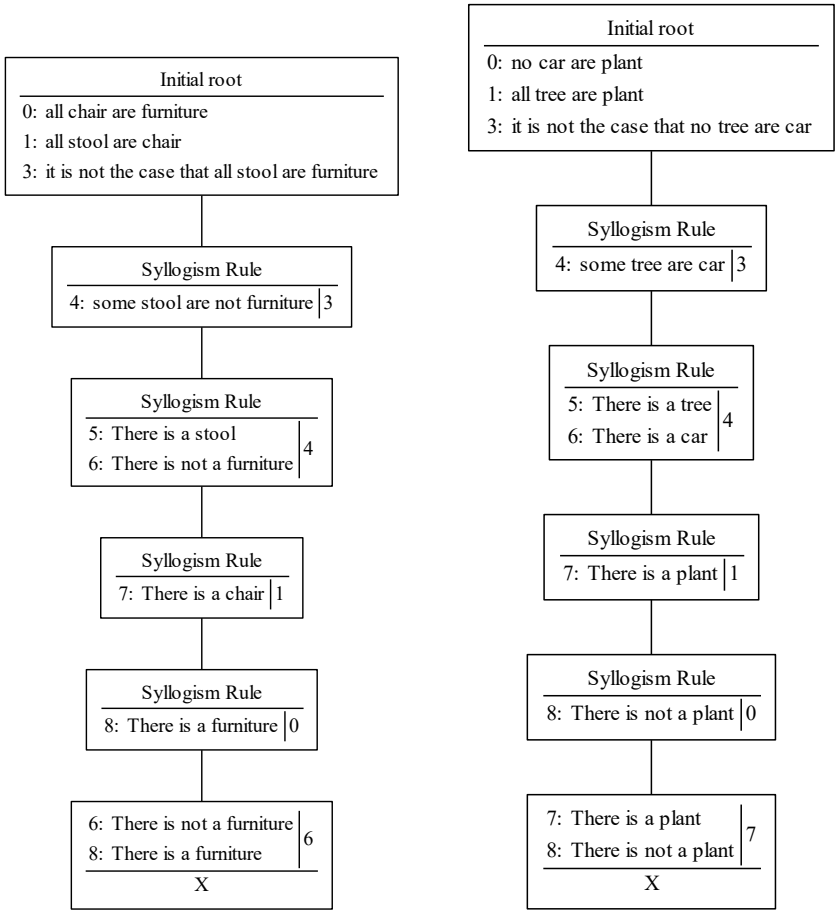
In these figures 4.2 and 4.3 it can be seen that the branches close for each example, effectively conveying that the conclusions are logically valid. In each branch, the rules (3.3) which are used are mentioned. For the first example, *Or* and *When* rules were used to reach the conclusion, whereas for the second example, *And*, *When* and *Or* rule were used.

4.2.2 Syllogism

There are 256 types of syllogism possible and among them 15 are universally valid, the others invalid (Hall 2010). We tested a data-set containing all the 256 types of possible syllogisms and our reasoner arrived at the correct conclusion in 100 percent of the cases. The reasoner's explanation for some different syllogism examples are illustrated here:

Example 1: AAA-1
INPUT
All chairs are furniture
All stools are chairs
Therefore, all stools are furniture

Example 2: EAE-2
INPUT
No cars are plants
All trees are plants
Therefore, no trees are cars



(a) AAA-1 (b) EAE-2

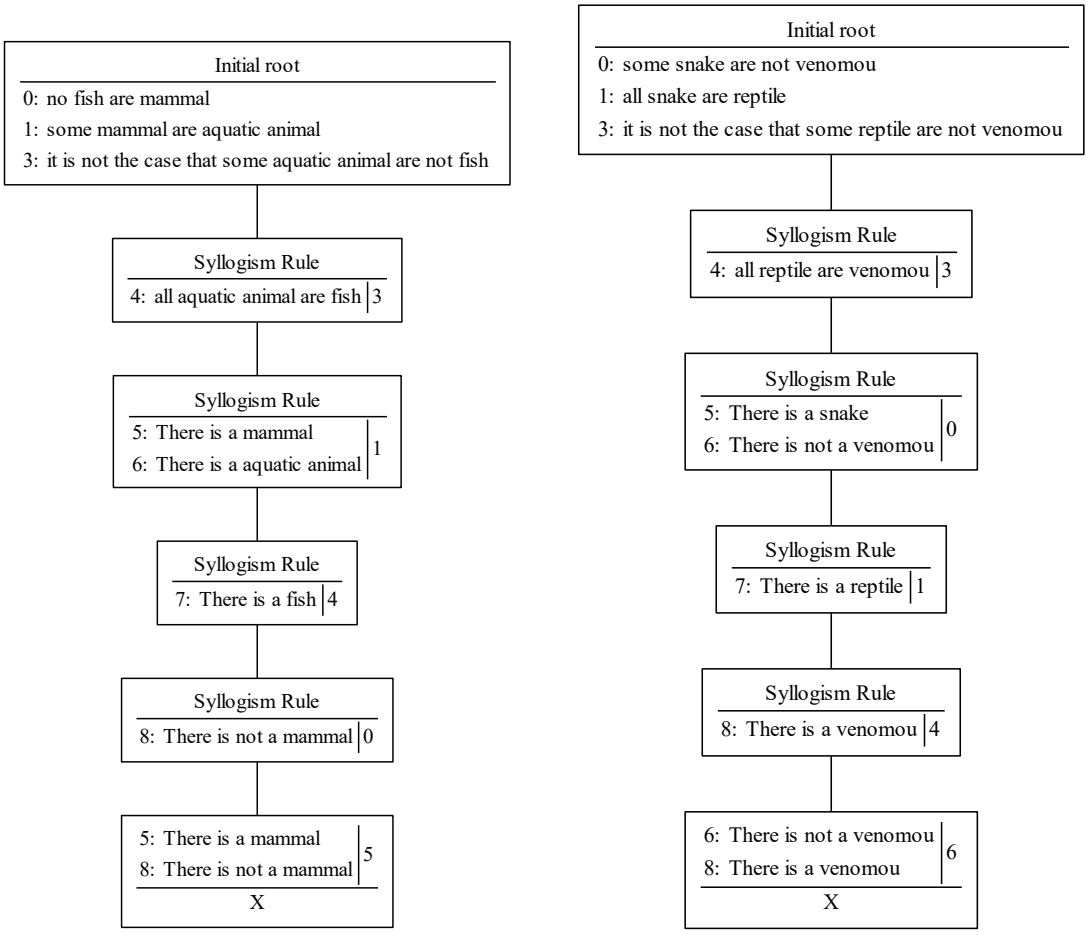
Figure 4.4: Output for the syllogism examples 1 and 2

Example 3: EIO-4
INPUT

No fishes are mammals
Some mammals are aquatic animals
Therefore, some aquatic animals are not fishes

Example 4: OAO-3
INPUT

Some snakes are not venomous
All snakes are reptiles
Therefore, some reptiles are not venomous



(a) EIO-4 (b) OAO-3

Figure 4.5: Output for the syllogism examples 3 and 4

In these figures (4.4 and 4.5) it can be seen that the branches close implying the conclusion is valid. The way this conclusion is reached using the syllogism rule (3.3) is also illustrated at each step.

4.2.3 Predicate Logic

Example 1
INPUT

For all x, for all y, if x is a cat and y is a mouse, then x is bigger than y
Tom is a cat and Jerry is a mouse
Conclusion: Tom is bigger than Jerry

OUTPUT

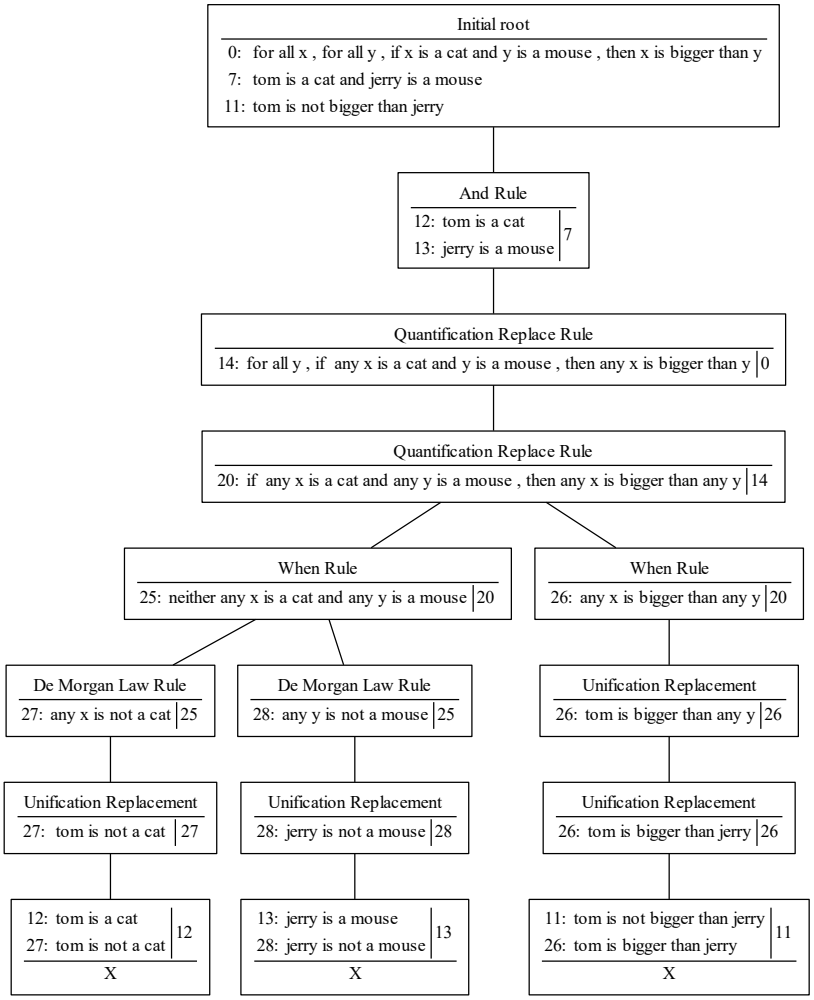


Figure 4.6: Output for the predicate logic example

In figure 4.6 the output of the example provided is illustrated. At each step the rules used are clearly stated and the branches close to convey that the conclusion is a valid one.

4.3 Invalid Conclusion

Here are some examples with invalid conclusion and the accompanying outputs by our reasoner, to show how our reasoner approaches invalid conclusions.

True negative: example 1
INPUT
For all Tom, Jerry is cool
Conclusion: Tom is cool
OUTPUT

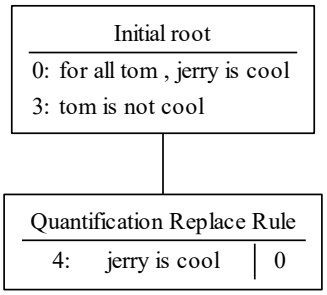


Figure 4.7: Output for the true negative example 1

True negative: example 2
INPUT
For all x, for all y, if x is a cat and y is a mouse, then x is bigger than y
Tom is a cat
Conclusion: Tom is bigger than mouse
OUTPUT

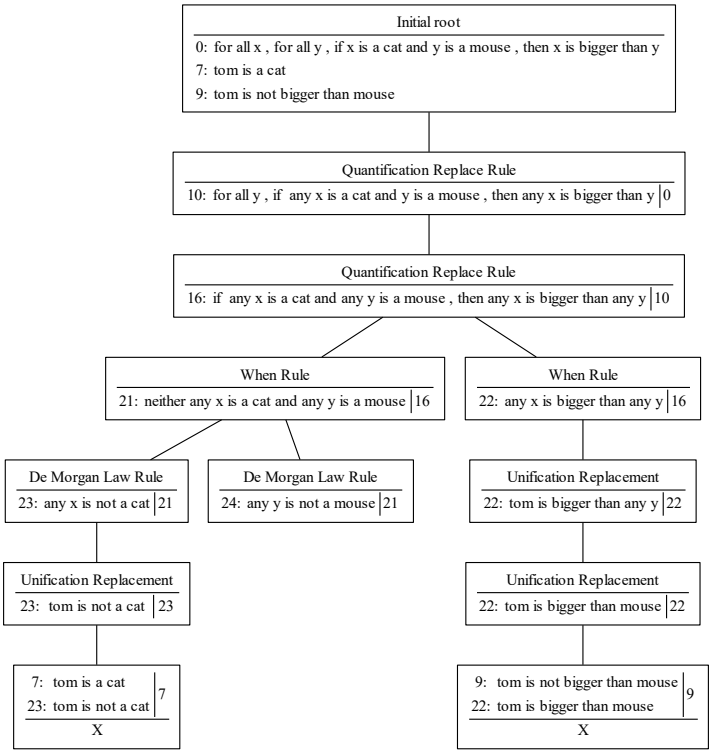


Figure 4.8: Output for the true negative example 2

True negative: example 2
INPUT
All chairs are furniture
All stools are chairs
Conclusion: Therefore, no stools are furniture
OUTPUT

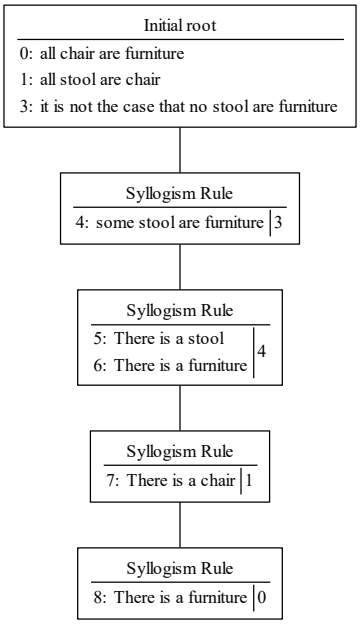


Figure 4.9: Output for the true negative example 3

As it can be seen in the outputs in figures 4.7, 4.8 and 4.9, The reasoner correctly reaches the decision that the conclusions are in fact invalid. The branches do not close in all the examples, because a rule has been broken.

4.4 Limitations

Within the constraints of these rules we have tested multiple examples and checked the limit of our reasoner. There are certain constraints to the natural language our reasoner can not handle:

- Our reasoner can not handle conjunctions (and/or) in the subject of the sentence. For example: *Tom and Jerry plays frisbee.* It has to be *Tom plays frisbee and Jerry plays frisbee.*
- Verb form has to be same across all the sentences in the premises and conclusion. For example *eat* in one sentence and *eaten* in another can not be handled by the reasoner.
- Our reasoner also can not handle verb in continuous form or participle form with are/were/has/had etc. associated with the verb. For example: *You have visited me/You are visiting me.*

- If/When has to be followed by "comma"/then for connecting the two clauses. For example: ***When*** *you visit me* ***then*** *I eat pizza.*

Within the scope of these constraints, we have tested a lot of sentences that our reasoner can semantically reason with and reach correct logical conclusion. To make it easier for the user to check the constraints we have also added a language checker component in the web interface that makes use of the expressions described before (3.4.1). The user can input a sentence in the language checker and the checker outputs if the sentence can be parsed by our reasoner or not. Below some examples and the associated result from this language checker is illustrated.

Example sentence 1
INPUT
If you visit me then I eat pizza
OUTPUT

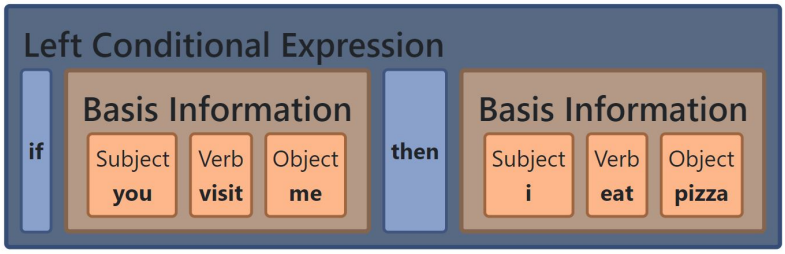


Figure 4.10: Output from the language checker for example 1

Example sentence 2
INPUT
When neither you visit him nor I visit him then he becomes sad
OUTPUT

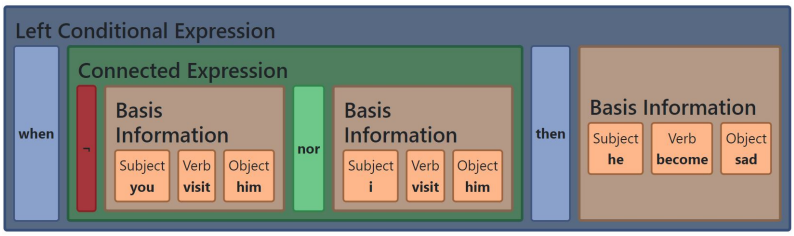


Figure 4.11: Output from the language checker for example 2

Example sentence 3
INPUT
For all Tom, Jerry is cool
OUTPUT

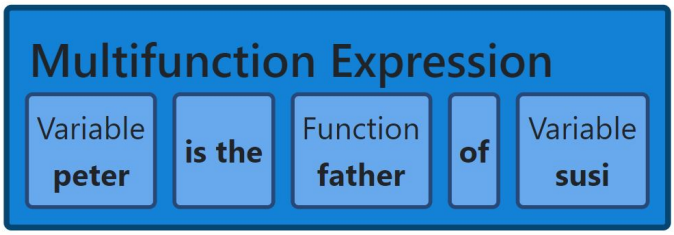


Figure 4.12: Output from the language checker for example 3

Figure 4.10 and 4.11 illustrated two conditional sentences, connected by conjunctions. According to the outputs these sentences are within the constraints and they are parsed correctly. Figure 4.12 shows the output of the checker that illustrates how a multi-function expression is parsed. The output shows how the parser simplified the negations.

5 Discussion

In this chapter, we first give our responses to the research questions in section 5.1, and make critical analysis in section 5.2. Then, we discuss directions for future Work in section 5.3.

5.1 Addressing Research Questions

As we mentioned in our introduction, we define three research questions at the beginning of our project, which we will try to answer now considering the explanations given throughout the previous sections:

- *Which subset of natural language can be used to reason in semantic natural tableau method?*
  With the implementation of the rules mentioned in the section 3, we have managed to handle multiple scenarios of our daily life language such as disjunctions, conjunctions or conditionals. This has also been expanded by the inclusion of syllogisms and different logic rules. Even though the clauses have to be expressed on a relatively constrained way, we can determine that the employed subset of natural language has been enough to reason in a semantic tableau method.

- *How should the semantic tableau method be adapted to directly reason with natural language?*
  By parsing the different elements within a sentence and analyzing them, we are able to prove recursively the set of rules and clauses that have been used in order to reason with the semantic tableau method. This has been largely explained in the section 3.2.

- *To what extent must the natural language structure be specified and constrained for the semantic tableau method in order to reason with it?* This will be dependant on the amount of rules and the subset of natural language that we want to incorporate: we could build a basic semantic tableau reasoner which

could handle very constrained set of language. However, it has been shown throughout the report that it can be enlarged and it could even get bigger if we incorporated additional rules and potentially NLP.

5.2 Critical Analysis

In this section we discuss the limitations of semantic tableau method. On one hand, we can observe that the use of semantic tableau method has the particular perk that it can get to work with a very limited and constrained language, making it an useful tool for straight-forward and individual sentences. On the other hand, extending the tableau method with further rules and covered sentences is a slow process as we have observed throughout the development of the project. Another remark against the use of the semantic tableau method is, that a specific domain with present sentences, needs to be in the exact constraint natural language as the one used by the reasoner. If that is not the case, depending on the size of the domain and present rules, it could be infeasible to adapt present statements to work with the given reasoner.

One possible solution to improve the range of sentences our reasoner can handle would be the use of neural networks, transformers or attention mechanisms. It would allow us to detect role, type and dependencies between different elements within a sentence which followingly could be used in the rules of the semantic tableau. We did not consider this approach as it would get us out of the original scope of our project to reason with constraint natural language.

Another limitation when reasoning with a semantic tableau method is that we are restricted to work with statements concluded in one sentence. Therefore the analysis of larger texts is complicated and not considered for the implementation of our model. This is one implication of the use of this particular method, so if we wanted to take into account this scenario, we would have to switch to another reasoning model.

The semantic tableau may be adapted towards some more specific domain, yet the language restrictions may bring some minor problems. For every separate domain the restricted language should be adapted towards its specific filler words and structures.

However, we have found the use of the semantic tableau method as useful and understandable way to reason with different input clauses and its use could be improved and enlarged by the use of some of the aforementioned techniques.

5.3 Future work

As our implementation at the current stage is limited, future studies can be done in several directions:

1. Improve the implementation by incorporating other logic rules to deal with more complex instances. So far, our implementation has covered the most fundamental logic rules and is capable of correctly solving simple arguments. Still, more rules can be considered and added to empower the reasoning tool, for example, "necessarily" or syllogisms with more premises and conclusions.

2. Broaden the range of controlled natural language. Our project has mainly looked into simple statements or propositions in everyday English, which is very restricted. Future studies can investigate methods to expand the subset of natural language or to design machine-oriented controlled natural languages, and to include more commonly used instances, both in life and practice.

3. Build reliable sample datasets. The lack of well-annotated testing datasets is also a problem worth investigating. During our implementation, we had to manually curate example data to evaluate the performance of our reasoner. With datasets of guaranteed size and quality, it would be easier to measure and assess how well the program carries out reasoning tasks, also, machine learning or other complicated techniques and models can be implemented.

4. Find a better way to evaluate our results. Given the nature of this project, it is complicated to assess the accuracy of the natural language reasoner. Therefore, finding a way to quantify the results could help visualizing how well is the project working and if there is room to improve on how is the sentence parse being carried out.

6 Conclusion

In this project, we study the problem of how to reason with a controlled natural language using the semantic tableau method.

We find that a natural language expression should be machine-friendly in order to reason with it using the semantic tableau method. Generally, it should satisfy the following prerequisites: 1) The sentences need to be structured in straightforward and clear syntax, with restricted grammar and ambiguity as little as possible. 2) The expressions must contain logical keywords such as **AND** or **OR**, or syllogism propositions like **ALL** or **SOME**, to represent the logical relations and to formulate an argument. 3) The language should not only resemble natural languages people read and write in everyday life but also simple enough for the computers to read and process.

In the implementation stage, we first build a language checker to parse through the input expression, identify the main constituents (subject, verb, object) of the expression, and check whether it is constrained enough for reasoning. Then, a reasoner is created with 4 propositional logical rules, 4 syllogism rules, the quantification rule, and other predicate logical rules applied. The reasoner can solve the input question, show the validity of the argument along with a semantic tableau explaining the reasoning process and every rule applied.

List of Figures

2.1 Semantic tableau rules for utilising LLFs 5
2.2 Semantic tableaux rules applied to the entailment: not all birds fly → some bird does not fly 5

3.1 Semantic tableau rules for propositional logic 7
3.2 Application workflow 8
3.3 Rules coming from the propositional logic 12
3.4 Rules used in syllogism 12
3.5 Replace rules from the predicate logic. 13
3.6 General structure of base expression and function expression. 15
3.7 Structure of the connected expression and conditional expression. . . 15
3.8 Structure of the syllogism expressions and quantification expression. . 16

4.1 Web Interface 19
4.2 Output for the propositional logic example 1 20
4.3 Output for the propositional logic example 2 21
4.4 Output for the syllogism examples 1 and 2 22
4.5 Output for the syllogism examples 3 and 4 23
4.6 Output for the predicate logic example 24
4.7 Output for the true negative example 1 25
4.8 Output for the true negative example 2 26
4.9 Output for the true negative example 3 27
4.10 Output from the language checker for example 1 28
4.11 Output from the language checker for example 2 28
4.12 Output from the language checker for example 3 29

Bibliography

Abzianidze, Lasha. 2015. "A Tableau Prover for Natural Logic and Language." In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing,* 2492–2502. Lisbon, Portugal: Association for Computational Linguistics, September. https://doi.org/10.18653/v1/D15-1296.

Castro-Manzano, J.-Martín. 2018. "A tableaux method for term logic." In *LANMR.*

Hall, Ronald L. 2010. *Logic: A Brief Introduction.* Chap. 6. XanEdu Publishing. ISBN: 978-1581526998.

Kalouli, Aikaterini-Lida, Annebeth Buis, Livy Real, Martha Palmer, and Valeria de Paiva. 2019. "Explaining Simple Natural Language Inference." In *Proceedings of the 13th Linguistic Annotation Workshop,* 132–143. Florence, Italy: Association for Computational Linguistics, August. https://doi.org/10.18653/v1/W19-4016.

Karttunen, Lauri. 2015. "From Natural Logic to Natural Reasoning." In *Computational Linguistics and Intelligent Text Processing,* edited by Alexander Gelbukh, 295–309. Cham: Springer International Publishing. ISBN: 978-3-319-18111-0.

Muskens, Reinhard. 2010. "An Analytic Tableau System for Natural Logic." In *Logic, Language and Meaning,* edited by Maria Aloni, Harald Bastiaanse, Tikitu de Jager, and Katrin Schulz, 104–113. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN: 978-3-642-14287-1.