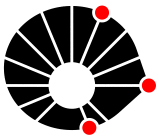

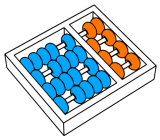

An open-world first-order logic reasoner with justification

E. Y. Sakabe *R. Gudwin* *E. L. Colombini*

Relatório Técnico - IC-PFG-21-11
Projeto Final de Graduação
2021 - Julho

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

An open-world first-order logic reasoner with justification

Eduardo Yuji Sakabe* Ricardo Ribeiro Gudwin† Esther Luna Colombini‡

Abstract

This paper explores the key concepts and methods towards the development of an open-world first-order logic reasoner with step-by-step justification. It covers the fundaments of classical logic to theorem proving algorithms and key methods for developing a reasoner. Lastly, it presents the reasoner's results through logic exercises.

1 Introduction

1.1 The Project

This article is part of an ongoing project in the fields of logic, linguistics, and semantics. The goal of the project is to build an agent capable of logical reasoning based on controlled English statements and queries, having a problem-solution approach understandable to readers non versed on logic, capable of interpreting sets of sentences and inferring new knowledge based on queries. Since its logic response requires the user's understanding, it needs to achieve beyond discerning whether a statement is true or false. It has to achieve a chain of entailments based on its presumed true statements leading to its answer. In this way, it attempts to imitate a human proving an argument.

This type of agent has a broad range of applications. It can reason about policies, describing in which conditions an action is permitted or forbidden [1], help humans to

*Instituto de Computação - UNICAMP e166810@dac.unicamp.br
†Faculdade de Engenharia Elétrica e de Computação - UNICAMP gudwin@unicamp.br
‡Instituto de Computação - UNICAMP esther@ic.unicamp.br

reflect about their morality and ethics giving logically grounded advice [2], perform formal verification of computer systems as a quality insurance mechanism [3], reason through ontologies finding inconsistencies and discovering new information [4] and give explanatory information over product reviews on online marketplaces [5].

For interpreting English and being able to perform reasoning, the first step is the translation of English sentences into a proper internal representation. This translation cannot be achieved in a single step. The Interpretation Model we propose uses four layers of structures for translating from natural English to Logic Expressions, with the proper syntax for logical reasoning. First, the Natural Language Parser decomposes the sentences by their syntax into a Parsed Grammar structure. Then, the Parsed Grammar is converted to a Discourse Representation Structure (DRS) [6], a semantic structure. Lastly, DRSs are converted to first-order logic syntax, a representation that is finally proper for being processed by the Logic Reasoner.

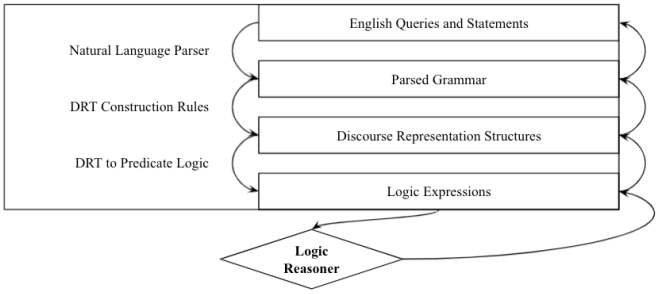

Figure 1: Interpretation Model

The process requires that the user first inserts input statements in controlled natural English, with the further translation of them into first-order logic syntax, and then their insertion into the knowledge base. Then, in the sequence, the user inputs a query, the agent translates it into first-order logic and finally gives a step-by-step solution, based on the knowledge base's premises.

This article will focus only on the Logic Reasoner module, which receives expressions in first-order logic syntax and answers queries with a step-by-step solution in first-order logic

syntax.

First-order logic was chosen since Hans Kamp's Discourse Representation Theory does not have a standard reasoning algorithm but has a great affinity with predicate calculus [6].

Also, there are extensive and well-established complete and sound algorithms for first-order logic theorem proving (e.g. Tableaux, Resolution).

Lastly, first-order logic seems a promising approach for a base system capable of adding features extending its semantics (e.g. temporal logic, higher-order logic, generalized quantifiers) [7]. It is worth mentioning that Discourse Representation Theory's ontological commitments are broader than classical logic's, thus for holding their complete semantics the logic system should be extended.

1.2 Classical Logic

Before pointing out the specific requirements for the project's reasoner a brief revision in classical logic is needed for the reader's further comprehension.

Classical logic is a powerful tool for analyzing information. Being able to infer the validity of new statements and answer questions starting from the premises in a formal manner similar to mathematics. Thus, translating information to this formal system gives the capability to perform reasoning about the provided context.

This subsection provides a brief introduction to classical logic, which includes propositional logic and first-order logic. Also, it's worth mentioning that this is not a complete explanation on the topic, neither on its syntax or semantics. For example, propositional logic has more connectives and rules of inference than shown in its subsection, and functions are not mentioned in the first-order logic's subsection although it composes its first-order logic's syntax. Those decisions were made to make this introduction succinct to include only the classical logic elements that are used in the Logic Reasoner.

1.2.1 Propositional Logic

Propositional logic is concerned with representing facts of the world and analyzing statements' validity [8]. Syntactically, it's composed by atomic sentences represented by letters (e.g. A, B, C) and logical connectives (e.g. $\vee, \wedge, \Rightarrow, \neg$) that can be used with atomic (and complex) sentences to form new complex sentences. All sentences assume true or false values, representing the sentences' validity.

Name	**Symbol**	**Usage**
negation	$\neg$	$\neg P$
conjunction	$\wedge$	$P \wedge Q$
disjunction	$\vee$	$P \vee Q$
implication	$\Rightarrow$	$P \Rightarrow Q$

Table 1: Logical Connectives

Atomic sentences are used to represent simple facts. For example C can represent *it's cloudy* and R can represent *it will rain*. Complex sentences add more meaning to atomic sentences as $\neg C$ that means *it's not cloudy* and $C \Rightarrow R$ means *if it's cloudy then it will rain*. The truth-value of a complex sentence directly depends on its connective and the sentences that compound it.

For instance, assuming that P is true, it is possible to infer that $\neg P$ is false and also that $P \vee Q$ is true. A complete truth-table for the logical connections mentioned is shown in Table 2:

P	Q	$\neg P$	$P \vee Q$	$P \wedge Q$	$P \Rightarrow Q$
False	False	True	False	False	True
False	True	True	True	False	True
True	False	False	True	False	False
True	True	False	True	True	True

Table 2: Truth-table

With the truth-table, it is possible to derive inference rules, i.e., patterns that are tautologies: propositions that are always true.

For instance the modus ponens inference rule: $P \Rightarrow Q, P \models Q$

It states that if $P \Rightarrow Q$ is true and P is true the conclusion Q is also true. Note on the truth-table that Q is necessarily true in the context where $P \Rightarrow Q$ and P are both true. A table of rules of reference with more examples is shown in Table 3:

Name	**Inference Rule**
Modus Ponens	$P \Rightarrow Q, P \models Q$
Modus Tollens	$P \Rightarrow Q, \neg Q \models \neg P$
Hypothetical Syllogism	$P \Rightarrow Q, Q \Rightarrow R \models P \Rightarrow R$
Conjunction Introduction	$P, Q \models P \wedge Q$
Conjunctive Syllogism	$P \wedge Q \models P, Q$
Disjunction Introduction	$P \models P \vee Q$
Disjunctive Syllogism	$P \vee Q, \neg Q \models P$
Material Implication	$P \Rightarrow Q \Leftrightarrow \neg P \vee Q$
De Morgan's Negation Conjunction	$\neg(P \wedge Q) \models \neg P \vee \neg Q$
De Morgan's Negation Disjunction	$\neg(P \vee Q) \models \neg P \wedge \neg Q$

Table 3: Propositional Logic Inference Rules modified from [9]

Inference rules enables the systematic proof of statements. The pattern-matching of sentences using inference rules derives new sentences and discovers sentences' truth values.

For example, consider the knowledge base of sentences (1) and (2) that are assumed as having true values and the query (Q):

1	$A \vee B$
2	$\neg B$
Q	A?

Since (2): $\neg B$ is true, consequently (3): B is false. Using the disjunctive syllogism inference rule to pattern-match (1) and (3) determines (4): A is true, therefore, the query is solved.

The example below provides a review of propositional logic, presenting how, starting from the translation of facts from the world, we are able to answer questions, using logical inference.

1 *If it's sunny, then Adam will go to the beach.*

2 *If it's rainy, then Adam will go to the library.*

3 *It's sunny.*

Q *Will Adam go to the beach?*

The first step is to convert the sentences to the syntax of propositional logic. Each simple fact is translated into an atomic sentence.

it's sunny $= S$

it's rainy $= R$

Adam will go to the beach $= B$

Adam will go to the library $= L$

Then, the causality relations, also known as if-then statements, are translated to use the implication operator $\Rightarrow$:

If it's sunny, then Adam will go to the beach $= S \Rightarrow B$

If it's rainy, then Adam will go to the library $= R \Rightarrow L$

The same statements can now be seen in propositional logic representation:

1 $S \Rightarrow B$

2 $R \Rightarrow L$

3 S

Q B?

This translated representation is now suitable to be used together with the inference rules. We can apply modus ponens in (1) and (3) to infer that (4): B is true. Thus, the query's answer is: *true*.

1.2.2 First-order Logic

While Propositional Logic's ontological commitments are only concerned with facts, First-order Logic's ontological commitments are concerned with representing facts, objects, and relations [8].

Syntactically, first-order logic's basic components are either terms representing objects of the world (e.g. $Chair$, $John$, A, Y_1), or predicates, representing relations between objects (e.g. $Likes(A, B)$, $Eats(A, C)$).

Atomic Sentences in first-order logic can be either predicates or terms. Analogous to propositional logic, atomic sentences form complex sentences with the with the use of connectives.

Using the first-order logic syntax, it is possible to represent natural language sentences like *Adam eats bread* as $Eats(Adam, Bread)$ and *if Adam is a man, then Adam is mortal* as $Man(Adam) \Rightarrow Mortal(Adam)$.

First-order logic sentences also have true or false values to indicate the sentences' validity.

Another interesting property of first-order logic is its capability to describe collections of objects using the $\forall$ and $\exists$ quantifiers, together with variables (e.g. x, y, z). The $\forall$ quantifier can be interpreted as *for all* or *every* and the quantifier $\exists$ can be understood as *exists*. Variables can be used in predicates as constants do, but they inherit the associated quantifiers' meaning expressing collections instead of specific objects. In that manner, it is possible to describe, for example, *for all things if something is a reptile, then this thing is an animal* as $\forall x Reptile(x) \Rightarrow Animal(x)$, that is semantically equivalent to *every reptile is an animal*. And it is also possible to represent *there exists something precious* as $\exists x Precious(x)$.

Furthermore, nested quantifiers can represent *Everything is part of something* as $\forall x \exists y Part(x, y)$.

A term with no variables is considered a ground term (e.g. $Precious(Gem)$).

With the addition of quantifiers and variables, first-order logic extends new inference rules as seen in Table 4. The inference rules in propositional logic are still valid for first-

order logic. Therefore, the sentence $Reptile(Turtle) \Rightarrow Animal(Turtle)$, $Reptile(Turtle)$ still derives the sentence Animal(Turtle) by modus ponens.

Name	**Inference Rule**
Universal Instantiation	$\forall x P(x) \models P(C)$
Universal Generalization	$P(C)$ for an arbitrary c $\models \forall x P(x)$
Existential Instantiation	$\exists x P(x) \models P(C)$ for some element C
Existential Generalization	$P(C)$ for some element c $\models \exists x P(x)$

Table 4: First-order Logic Inference Rules from [9]

In order to exemplify first-order logic inference, take the knowledge base and the query below:

1	$\forall x Reptile(x) \Rightarrow Animal(x)$
2	$Reptile(Turtle)$
Q	$Animal(Turtle)$?

Universal Instantiation can substitute the constant $Turtle$ in the ground term (2) for the variable x in (1), introducing the new valid sentence (3): $Reptile(Turtle) \Rightarrow Animal(Turtle)$. And then, modus ponens is applied between (2) and (3) to produce (4): $Animal(Turtle)$ answering (Q) as being true.

Note that the strategy shown is to create ground terms using universal instantiation and then perform inference rules from propositional logic. This procedure is called propositionalization and will be mentioned further in this article.

Another interesting aspect of first-order logic semantics is the ability to not only answer whether the query is true or false but which objects hold for a true value query. As shown in the example below:

1 $Reptile(Turtle)$

2 $Reptile(Lizard)$

3 $\forall x Reptile(x) \Rightarrow Animal(x)$

Q $\exists y Animal(y)$

Universal instantiation is used between (1) and (3), (2) and (3), deriving respectively:

4 $Reptile(Turtle) \Rightarrow Animal(Turtle)$

5 $Reptile(Lizard) \Rightarrow Animal(Lizard)$

Modus ponens is applied between (1) and (4), and (2) and (5):

6 $Animal(Turtle)$

7 $Animal(Lizard)$

The answer to the query is true and those are two objects (constants) that can be substitutes to the variable. This substitution takes the notation of: $\{y/Turtle\}$, $\{y/Lizard\}$. Therefore, first-order logic achieves the answer for whom or which the query stands as true.

1.3 The Logic Reasoner's Requirements

The main objective of the Logic Reasoner is to build an algorithm capable of giving step-by-step solutions like the ones demonstrated in the previous subsection.

This subsection describes in more detail the reasoner's requirements and the prioritization of the algorithm design process.

1.3.1 Open-world Assumption

In closed-world assumption, if the query does not derives from the knowledge base's statements, its validity is considered to be false, since this assumption expects the agent has complete knowledge of the world. Therefore, if the query's value does not entail from its

premises it is concluded to be false.

In open-world assumption, if the query does not entail from the premises, its value is unknown, as the example below shows:

$$\begin{array}{ll} 1 & \forall x\ TaxEvasion(x) \Rightarrow Criminal(x) \\ \hline Q & Criminal(John)? \end{array}$$

In closed-world assumption, if there are no inference rules to entail the query, the query's result is considered to be false. In an open-world assumption, the query's result is considered to be unknown.

The open-world option was chosen for the reasoner project since we assume that the agent has incomplete world information. Considering the same constraint for the last example, it becomes clear that it is semantically incorrect to answer the query as false since it is not possible to confirm that *John is not a criminal* without evidence to support this statement. The information that $TaxEvasion(John)$ could be present in the world, but not in the agent's knowledge base. Therefore, the query's answer should be unknown.

1.3.2 Removed Inference Rules

Some inference rules are not intuitive for humans to understand or do not grasp the semantic properties intended for the project. Therefore, these rules were removed from the inference process.

De Morgan Rule:

$$\neg(A \vee B) \ \Leftrightarrow \neg A \wedge \neg B$$

$$\neg(A \wedge B) \ \Leftrightarrow \neg A \vee \neg B$$

The De Morgan rule was removed because it did not seem intuitive for the non-versed logic reader.

Material Implication:

$$A \Rightarrow B \Leftrightarrow \neg A \vee B$$

This rule has a similar problem to the last one. Material implication did not seem to be understandable to the ordinary reader.

Vacuous Truth:

$$\neg A \models A \Rightarrow B$$

This rule is semantically not desirable since the implication connective represents if-then statements, meaning the causality relation. It's incorrect to say that if a fact is false, any causality relation is true, taking the fact as a premise.

For example, consider the premise:

It's not cloudy $= \neg C$

It's not accurate to arbitrarily assume that:

if it is cloudy, then it is sunny $= C \Rightarrow S$

1.3.3 Prioritisation

Considering that the main objective of the project, as a whole, is to achieve an intelligent agent capable of explaining, step by step, logic solutions to users, mimicking human arguments, the algorithm development process prioritized the soundness of the chain of entailments over completeness, since humans don't have complete logic reasoning capabilities.

2 Related Algorithms

Before describing the Logic Reasoner's algorithm it's important to mention related algorithms and explain why they did not commit to the requirements for this project.

2.1 Resolution Method [10] [8]

The basis of the Resolution Method is the proof by contradiction. To demonstrate that $KB \models A$, the statement $KB \wedge \neg A$ must not be satisfiable. First, the sentence $KB \wedge \neg A$ is converted to CNF (conjunctive normal form). CNF is defined as a conjunction with one or more clauses. Each clause is composed by a disjunction of positive or negative literals.

The example below is in CNF:

$(A \vee \neg B \vee C) \wedge (B \vee \neg C) \wedge (\neg B)$

The resolution algorithm continues by creating new clauses, joining two clauses that have a common complimentary literal (e.g. D and $\neg D$) and eliminating the complimentary literals from the new clause. This process continues until one of the two possibilities happens:

- There are no clauses to be added, therefore $KB \not\models A$ and the query's result is false.
- An empty clause is generated, therefore $KB \models A$ and the query's result is true.

The empty clause only happens when there are two clauses that only contain the complimentary literals like $(\neg A)$ and (A). Those two clauses cannot coexist as both being true, resulting in a contradiction.

Below, there is an example of the resolution algorithm with the KB already in CNF:

1	$A \vee \neg B \vee C$
2	$B \vee \neg C$
3	$\neg B$
Q	C?

To check if the C query is true, its negation, (4) $\neg C$ is added.

From (1) and (2) there comes the complimentary literal B, generating (5): $A \vee C \vee \neg C$.

Also, (2) and (3) resolve B, resulting in (6): $\neg C$.

Note that (6) and (4) are complementary, resulting in an empty clause, a contradiction, and then the C query is true.

The resolution method does not commit to the project requirements, because it relies on a closed-world assumption. To show $KB \models A$, it tries to demonstrate that $KB \wedge \neg A$ is not satisfiable. In an open-world assumption, it's possible that the value of A is not reachable by the premises. Therefore, $KB \wedge \neg A$ would not lead to a contradiction, meaning the query is false. But by the project's standards, the answer should be unknown.

Another reason for not using this method is the requirement to transform sentences to CNF. CNF conversion uses material implication to eliminate the implication connective, which was previously mentioned as an inference rule not understandable to the common reader.

2.2 Tableau Method [11]

Like the resolution method, the tableau also relies on proof by contradiction, therefore to $KB \models A$, the formula $KB \wedge \neg A$ must be not satisfiable, using material implication to eliminate the implication connective, together with De Morgan for negative conjunctions and negative disjunctions. Therefore, the tableau method did not commit to the Logic Reasoner's requirements.

But, this method is worth mentioning, since it has some key ideas used in our Logic Reasoner algorithm, and the method by itself is very interesting, because it is visually intuitive.

The tableau method enables the decomposition of complex sentences into simpler ones. First, the sentence $KB \wedge \neg A$ is inserted as the root of a tree. Then, each connective is decomposed expanding new nodes or branches until all nodes are positive or negative literals.

- Conjunctions are eliminated by adding the conjuncts in the same branch.
- Disjunctions open a new branch and add each disjunct as the leaf of a branch.

- Negation uses De Morgan to convert the negated disjunction into conjunction and vice-versa, and also double negation is eliminated adding the resultant sentence and a new node in the same branch.

- Implication is eliminated using material implication.

The complementary literals in the tree, belonging to the same branch, form a closure, a contradiction. Since there is a contradiction in the branch, further expansions are not necessary.

The decomposition process continues until there are only literals or closed branches. If all branches are closed, the statement $KB \wedge \neg A$ is false, therefore $KB \models A$.

If there are opened branches, the statement $KB \wedge \neg A$ has not been proven false, therefore $KB \not\models A$.

The tableau method uses inference rules to decompose the sentences, resulting in simpler sentences that are also valid. This same idea is presented in the preprocessing phase of our Logic Reasoner.

The example below demonstrates the tableau method process:

1	$A \vee B$
2	$\neg A \wedge B$
Q	B?

As the tableau method attempts a proof by contradiction, the negated query is added in the tableau with the knowledge base sentences (Figure 2).

The first sentence is decomposed, as a disjunction, opening a branch, and each disjunct is inserted in one of the branches (Figure 3).

Note that in the right branch, the complementary B and $\neg B$ form a closure (Figure 4). Since it's closed, further expansions in this branch are not necessary.

Next, the conjunction of the second sentence derives two nodes on the same branch

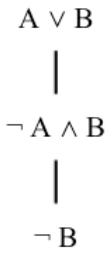

Figure 2: Initial Tableau

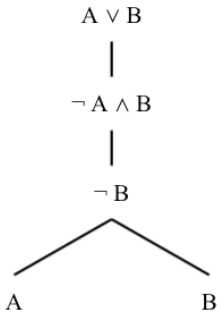

Figure 3: Disjunction Decomposition

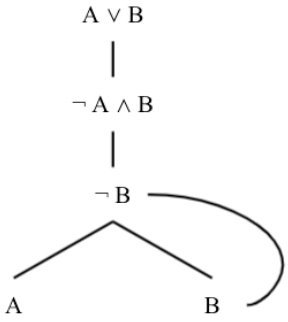

Figure 4: Closure on the right branch

(Figure 5).

The complementary A and $\neg A$ close the left branch (Figure 6). Both branches are closed, which means a contradiction, resulting that the query's answer is true.

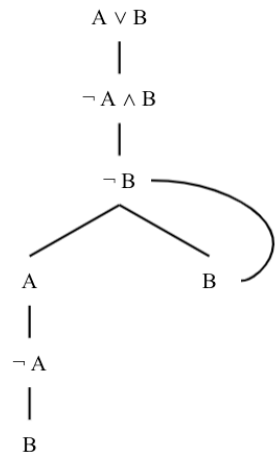

Figure 5: Conjunction Decomposition

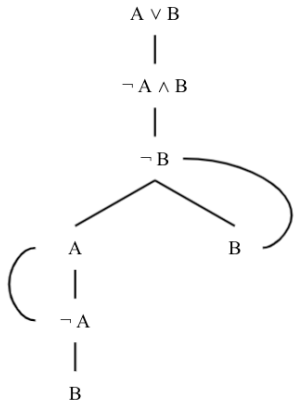

Figure 6: Closure on the left branch

2.3 RACE [12]

The Attempto project [13] has similar goals to the project discussed in this article. Attempto Controlled English (ACE) texts are also translated to Discourse Representation Structures that can be translated to other languages such as first-order logic, OWL 2, and others.

RACE is the first-order logic reasoner used in the Attempto project, using a modification of SATCHMO [14], a first-order logic theorem prover in Prolog. RACE has the interesting feature of providing proof justification. While SATCHMO just answers if the query succeeds

or fails, RACE gives the minimal subsets of the axioms that entail the query's result for both success and failure. Also, RACE claims to work under an open-world assumption. But there are two reasons for RACE not reaching the Logic Reasoner's requirements.

First, the proof justification provided by RACE is the minimal subset of axioms, which is different from a step-by-step solution of the problem. Take the example below:

1 $(S \vee P) \Rightarrow R$

2 $\neg R$

Q $\neg S$?

The RACE reasoner would answer: the following minimal subset of the axioms answer the query.

1 $(S \vee P) \Rightarrow R$

2 $\neg R$

Note that this justification does not explains how it got to its conclusion. It only mentions the axioms used for the proof. Following our Logic Reasoner's justification requirements, the answer should be similar to the justification below:

The query $\neg S$ is true, because:

(2) is true, then (3) R is false.

(1) is true and (3) is false, then (4) $S \vee P$ is false.

(4) is false, then (5) S is false.

(5) is false, then (6) $\neg S$ is true.

The second requirement issue is that RACE do not answers queries as false as seen in the execution of Figure 7:

Note that the example clearly states $\neg Pet(John)$ is an axiom. Therefore, $Pet(John)$? should be false.

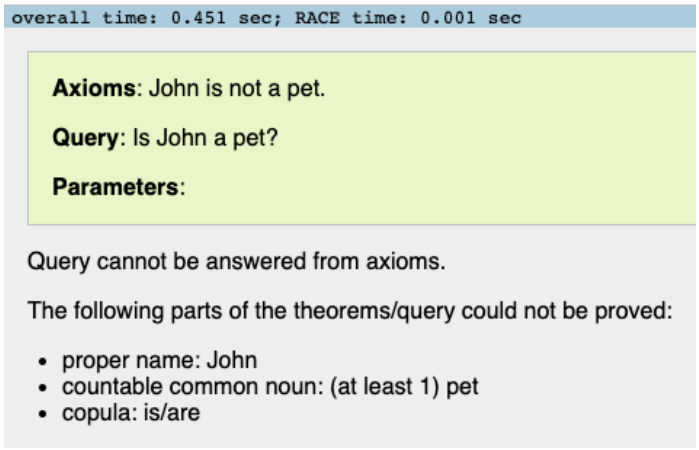

Figure 7: RACE Query answering execution

Queries in open-world assumption should have three possible states: true, false, or unknown. RACE treats false queries as unknown and this is undesirable when it is possible to prove the query is false.

The careful reader will notice that the counterexamples presented in the last subsections are in propositional logic, and not in first-order logic, but since those techniques work under propositionalization, the issues will persist for first-order logic .

3 The Logic Reasoner

3.1 Inherited Concepts

Before describing the Logic Reasoner, it is important to point some key concepts and algorithms that take a crucial part in the algorithm design.

3.1.1 Backward Chaining

The backward chaining algorithm deals with inferences starting from the query [8]. In fact, it is a form of goal-directed reasoning, useful for answering specific questions. It is a recursive depth-first search approach, which searches for a goal in the knowledge base, such that this goal can be the query itself or a sentence entailing the goal. The example below

demonstrates the backward chaining approach:

1	$\forall x C(x) \Rightarrow F(x)$
2	$\forall x P(x) \Rightarrow B(x)$
3	$\forall x F(x) \Rightarrow G(x)$
4	$\forall x B(x) \Rightarrow Y(x)$
5	$C(Nick)$
Q	$G(Nick)$?

Starting from query (Q) as a goal, the algorithm searches which sentences of the knowledge base answer's it, directly or indirectly. Sentence (3) is a valid sentence for this, because if the antecedent (G-1): $F(Nick)$? is true, then (Q) is also true by modus ponens and universal instantiation. Then, (G-1) becomes the new goal to be answered.

The same search method results in the (G-2): $C(Nick)$? goal, that proves (G-1), using modus ponens in (1).

Finally, sentence (5) proves (G-2), and consequently, query (Q) is proved recursively as true.

Backward chaining is used in our Logic Reasoner for answering the query or its expansions. For this, if the goal has a constant, it looks in the knowledge base for an equivalent sentence, but if the goal has a variable. it looks for a valid substitution. This substitution will be further explained in the Unification Algorithm subsection. Backward chaining also decomposes connectives in the query, creating new goals. For example, supposing an $A \wedge B$ query, backward chaining breaks the query in two goals, A and B, which together prove the query.

3.1.2 Forward Chaining

The forward-chaining algorithm deals with inferences starting from the knowledge base's sentences [8]. This approach applies inference rules onto the knowledge base's sentences,

adding the resultant sentences to the knowledge base. Therefore, it is a form of data-directed reasoning and a breadth-first search approach. The example below demonstrates:

1	$\forall x C(x) \Rightarrow F(x)$
2	$\forall x P(x) \Rightarrow B(x)$
3	$\forall x F(x) \Rightarrow G(x)$
4	$\forall x B(x) \Rightarrow Y(x)$
5	$C(Nick)$
Q-1	$G(Nick)$?

Starting from (1), it is possible to apply the inference rule hypothetical syllogism with (3) resulting in (6): $\forall x C(x) \Rightarrow G(x)$.

The same inference rule is applied between (2) and (4) to produce (7): $\forall x P(x) \Rightarrow Y(x)$.

And finally, universal instantiation and modus ponens between (6) and (5) produce (8): $G(Nick)$. Therefore, the query's answer is true.

Note that forward chaining produce sentences not related to the query's proof as in (7).

Forward chaining is used in our Logic Reasoner to produce simpler sentences in the knowledge base as done by the expansions in the Tableau Method. Backward chaining can perform in a simpler manner, only searching for equivalent sentences in the knowledge base or valid substitutions.

3.1.3 Skolem Normal Form

Skolemization is a method used for eliminating the existential quantifier, converting the sentence to Skolem Normal Form, which is similar to Existential Instantiation.

Skolemization: $\forall x \exists y L(x, y) \models \forall x L(x, F(x))$

Existential Instantiation: $\forall x \exists y L(x, C) \models \forall x L(x, C)$

The semantic difference is that the term C in existential instantiation is a constant, and thus it is related to a specific object. This is not necessarily the case, since the existential quantifier does not map to a particular entity, but a collection. Skolemization solves this issue with a Skolem function, as exemplified by the function F, which maps to a possible object, not a specific one [8].

The method used in our Logic Reasoner is similar to the existential instantiation, but it keeps the skolemization semantics.

Method used: $\forall x \exists y L(x, C) \models \forall x L(x, Sko_1)$

It removes the existential quantifier and flags the variable as being skolemized. Therefore, Sko_n does not map to a specific constant when first-order logic inference is applied.

3.1.4 Unification Algorithm [15]

The unification procedure returns, if possible, a substitution that makes two sentences identical [8]:

$Unify(S_1, S_2) = \theta$ where $Subst(S_1, \theta) = Subst(S_2, \theta)$

For example, the execution of the function $Unify(P(x, y), P(A, B))$ where x and y in the first argument are variables, and A and B in the second argument are constants, returns the substitution: $\theta = \{x/A, y/B\}$

Note that using θ to substitute the variables from the arguments results in identical sentences:

$Subst(P(x, y), \theta) = P(A, B)$

$Subst(P(A, B), \theta) = P(A, B)$

When there is no possible substitution, for example $Unify(P(A, y), P(B, C))$, the unification returns "fail".

The unification algorithm is useful for dealing with quantified sentences. Supppose the

universally quantified sentence $\forall x P(x) \Rightarrow Q(x)$. The unification of its antecedent with the ground term $P(A)$ would result in $\theta = \{x/A\}$. This substitution could be used to universally instantiate the quantified sentence, producing $P(A) \Rightarrow Q(A)$. Also, when the query is existentially quantified, for example $\exists x P(x)$, if the query unifies with a sentence in the knowledge base as $P(A)$, the substitution $\{x/A\}$ is found, being a valid answer to the query.

The two examples mentioned above are exactly the case where the unification algorithm is used in our Logic Reasoner.

3.2 Logic Reasoner Algorithm Description

Our Logic Reasoner algorithm has two phases: preprocessing and answering.

The preprocessing phase is composed by 4 stages:

1. The elimination of existential quantifiers in the knowledge base by skolemization.
2. Forward Chaining reasoning over sentences that are both universally quantified. For example: $\forall x P(x) \Rightarrow Q(x), \forall y Q(y) \Rightarrow R(y) \models \forall z P(z) \Rightarrow R(z)$.
3. The use of universal instantiation to the knowledge base's sentences containing predicate symbols, present in the query.
4. Forward Chaining through ground terms, analogous to propositional logic reasoning.

The answering phase uses backward chaining to decompose the query's connectives recursively, searching for the goal in the knowledge base or, if it is an existentially quantified query, searching for a substitution using unification.

The data structure for the logic sentences is an abstract syntax tree (AST), where every node is either a connective, a predicate, a variable, or a constant. The nodes have a justification attribute. Every time an inference rule is applied, the justification attribute from the asserted sentence is appended together with the justifications from the sentences

triggering the inference rule. Therefore, when the query has a value different from unknown, its justification attribute is the sequence of entailments starting from the premises.

Currently, part 2 and 3 are still under development. Thus, the program does not support sentences with the universal quantifier in the knowledge base. The development process works under Test-driven development (TDD). There are over 100 tests for propositional logic and 200 for first-order logic passing.

4 Results

4.1 Propositional Logic

Test 1: A simple modus ponens example

1	*If it's sunny, then Adam will go to the beach.*
2	*If it's rainy, then Adam will go to the library.*
3	*It's sunny.*
Q	*Will Adam go to the beach?*

First, the example is converted to propositional logic.

1	$S \Rightarrow B$
2	$R \Rightarrow L$
3	S
Q	B?

The original results from the algorithm are in prefix notation as shown below:

```
Query(B) is true BECAUSE:
IMP(S B) is true and S is true THEN B is true.
```

But since this article has classical logic notation, the results will be converted:

The query B is true BECAUSE:

$S \Rightarrow B$ is true and S is true THEN B is true.

Test 2: Autonomous Car

An autonomous vehicle obeys the following rules to navigate through the city:

If there is a person in front of the car, then brake.

If the traffic light is yellow, there is a policeman and the ground is not slippery then brake.

If there is a police car, then there is a policeman.

If it is snowing, then the ground is slippery.

If the ground is slippery, then the ground is not dry.

If the traffic light is red, then brake.

If it is winter, then it is snowing.

Its sensors are currently receiving the information below:

The traffic light is yellow.

The traffic light is not red.

It is not snowing.

The ground is dry.

There is a police car.

There is not a person in front of the car.

Will the car brake?

First, the problem is converted to propositional logic syntax:

1	$PersonInFrontOfCar \Rightarrow Brake$
2	$(YellowLight \wedge Policeman \wedge \neg Slippery) \Rightarrow Brake$
3	$Policecar \Rightarrow Policeman$
4	$Snow \Rightarrow Slippery$
5	$Slippery \Rightarrow \neg Dry$
6	$RedLight \Rightarrow Brake$
7	$Winter \Rightarrow Snow$
8	$YellowLight$
9	$\neg RedLight$
10	$\neg Snow$
11	Dry
12	$Policecar$
13	$\neg PersonInFrontOfCar$
Q	$Brake?$

The response below is a more complex, using multiple inferences to get to the query's answer. Note that each line of the justification uses sentences from the knowledge base or information proved before to generate new information.

The query $Brake$ is true BECAUSE:

1. $Policecar \Rightarrow Policeman$ is true and $Policecar$ is true THEN $Policeman$ is true.

2. $YellowLight$ is true and $Policeman$ is true THEN $YellowLight \wedge Policeman$ is true.

3. Dry is true THEN $\neg Dry$ is false.

4. $Slippery \Rightarrow \neg Dry$ is true and $\neg Dry$ is false THEN $Slippery$ is false.

5. $Slippery$ is false THEN $\neg Slippery$ is true.

6. $YellowLight \wedge Policeman$ is true and $\neg Slippery$ is true THEN $YellowLight \wedge Policeman \wedge \neg Slippery$ is true.

7. $(YellowLight \wedge Policeman \wedge \neg Slippery) \Rightarrow Brake$ is true and $YellowLight \wedge Policeman \wedge \neg Slippery$ is true THEN $Brake$ is true.

4.2 First-Order Logic

Test 1: Is Fritz green?

Suppose that the goal is to conclude the color of a pet named Fritz and that the rule base contains the following four rules:

If X croaks and X eats flies, then X is a frog.
If X chirps and X sings, then X is a canary.
If X is a frog, then X is green
If X is a canary, then X is yellow

It is also known that Fritz croaks and eats flies.

Fritz croaks.
Fritz eats flies.

First, the problem is converted to first-order logic syntax:

1	$\forall x((Croaks(x) \wedge EatFlies(x)) \Rightarrow Frog(x))$
2	$\forall x((Chirps(x) \wedge Sings(x)) \Rightarrow Canary(x))$
3	$\forall x(Frog(x) \Rightarrow Green(x))$
4	$\forall x(Canary(x) \Rightarrow Yellow(x))$
5	$Croaks(Fritz)$
6	$EatFlies(Fritz)$
Q	$Green(Fritz)$?

Then, our algorithm answers:

The query $Green(Fritz)$ is true BECAUSE:

1. $\forall x((Croaks(x) \wedge EatFlies(x)) \Rightarrow Frog(x))$ is true and $\forall x(Frog(x) \Rightarrow Green(x))$ is true THEN $\forall x((Croaks(x) \wedge EatFlies(x)) \Rightarrow Green(x))$ is true.

2. $\forall x((Croaks(x) \wedge EatFlies(x)) \Rightarrow Green(x))$ is true and $\{x : Fritz\}$ THEN $(Croaks(Fritz) \wedge EatFlies(Fritz)) \Rightarrow Green(Fritz)$ is true.

3. $Croaks(Fritz)$ is true and $EatFlies(Fritz)$ is true THEN $Croaks(Fritz) \wedge EatFlies(Fritz)$ is true.

4. $(Croaks(Fritz) \wedge EatFlies(Fritz)) \Rightarrow Green(Fritz)$ is true and $Croaks(Fritz) \wedge EatFlies(Fritz)$ is true THEN $Green(Fritz)$ is true.

Test 2: Deans and Professors

Consider the information below:

Lucy is a professor.

All professors are people.

Fuchs is a dean.

Deans are professors.

All professors consider the dean a friend or don't know him.

Everyone is a friend of someone.

People only criticize people that are not their friends.

Lucy criticized Fuchs.

The question is: *Does Lucy know Fuchs?*

First, the information and the question are converted to first-order logic syntax:

1	$prof(Lucy)$
2	$\forall x(prof(x) \Rightarrow person(x))$
3	$dean(Fuchs)$
4	$\forall x(dean(x) \Rightarrow prof(x))$
5	$\forall x, y(prof(x) \wedge dean(y) \Rightarrow isFriendOf(y, x) \vee \neg knows(x, y))$
6	$\forall x(\exists y(isFriendOf(y, x)))$
7	$\forall x, y(person(x) \wedge person(y) \wedge criticize(x, y) \Rightarrow \neg isFriendOf(y, x))$
8	$criticize(Lucy, Fuchs)$
Q	$knowns(Lucy, Fuchs)$?

Then, our reasoner answers:

The query $knowns(Lucy, Fuchs)$ is false BECAUSE:

1. $\forall x, y(prof(x) \wedge dean(y) \Rightarrow isFriendOf(y, x) \vee \neg knows(x, y))$ is true and $\{x : Lucy, y : Fuchs\}$ THEN $prof(Lucy) \wedge dean(Fuchs) \Rightarrow isFriendOf(Fuchs, Lucy) \vee \neg knows(Lucy, Fuchs)$
2. $prof(Lucy)$ is true and $dean(Fuchs)$ is true THEN $prof(Lucy) \wedge dean(Fuchs)$ is true.
3. $prof(Lucy) \wedge dean(Fuchs) \Rightarrow isFriendOf(Fuchs, Lucy) \vee \neg knows(Lucy, Fuchs)$

is true and $prof(Lucy) \wedge dean(Fuchs)$ is true THEN $isFriendOf(Fuchs, Lucy) \vee \neg knows(Lucy, Fuchs)$ is true.

4. $\forall x, y(person(x) \wedge person(y) \wedge criticize(x, y) \Rightarrow \neg isFriendOf(y, x)$ is true and $\{x : Lucy, y : Fuchs\}$ THEN $person(Lucy) \wedge person(Fuchs) \wedge criticize(Lucy, Fuchs) \Rightarrow \neg isFriendOf(Fuchs, Lucy)$ is true.

5. $\forall x(prof(x) \Rightarrow person(x))$ is true and $\{x : Lucy\}$ THEN $prof(Lucy) \Rightarrow person(Lucy)$ is true.

6. $prof(Lucy) \Rightarrow person(Lucy)$ is true and $prof(Lucy)$ is true THEN $person(Lucy)$ is true.

7. $\forall x(dean(x) \Rightarrow prof(x))$ and $\forall x(prof(x) \Rightarrow person(x))$ is true THEN $\forall x(dean(x) \Rightarrow person(x))$ is true.

8. $\forall x(dean(x) \Rightarrow person(x))$ is true and $\{x : Fuchs\}$ THEN $dean(Fuchs) \Rightarrow person(Fuchs)$ is true.

9. $dean(Fuchs) \Rightarrow person(Fuchs)$ is true and $dean(Fuchs)$ is true THEN $person(Fuchs)$ is true.

10. $person(Lucy)$ is true and $person(Fuchs)$ is true and $criticize(Lucy, Fuchs)$ is true THEN $person(Lucy) \wedge person(Fuchs) \wedge criticize(Lucy, Fuchs)$ is true.

11. $person(Lucy) \wedge person(Fuchs) \wedge criticize(Lucy, Fuchs) \Rightarrow \neg isFriendOf(Fuchs, Lucy)$ is true and $person(Lucy) \wedge person(Fuchs) \wedge criticize(Lucy, Fuchs)$ is true THEN $\neg isFriendOf(Fuchs, Lucy)$ is true.

12. $\neg isFriendOf(Fuchs, Lucy)$ is true THEN $isFriendOf(Fuchs, Lucy)$ is false.

13. $isFriendOf(Fuchs, Lucy) \vee \neg knows(Lucy, Fuchs)$ is true and $isFriendOf(Fuchs, Lucy)$ is false THEN $\neg knows(Lucy, Fuchs)$ is true.

14. $\neg knows(Lucy, Fuchs)$ is true THEN $knows(Lucy, Fuchs)$ is false.

4.3 Further Work

The Logic Reasoner is not fully implemented yet. It still has missing inference rules, as:

- Constructive Dillema: $A \Rightarrow B, C \Rightarrow D, A \vee D \models B \vee D$
- Implication's Conjunction Decomposition: $A \Rightarrow (B \wedge C) \models A \Rightarrow B, A \Rightarrow C$
- Implication's Disjunction Decomposition: $(A \vee B) \Rightarrow C \models A \Rightarrow C, B \Rightarrow C$

Although, it can process the existential quantifier in the knowledge base and query, but the universal quantifier is not implemented yet. Therefore, it only performs simple *For which objects is this sentence true?* queries.

After finishing the implementation of stages 2 and 3, the next step will be extending generalized quantifiers [16] to its semantics. Generalized quantifiers ontological commitments reaches more quantifier expressions present in natural language as *four chairs, only one, at least two, all but three, more than half, no...except, usually, never, each one, each other*.

References

[1] Joseph Y. Halpern and Vicky Weissman. Using first-order logic to reason about policies.

[2] Ghanem Mahmoud. Oxford uehiro prize in practical ethics: Should we take moral advice from our computers?

[3] Vladimir Pavlov, Alexander Schukin, and Tanzilia Cherkasova. Exploring automated reasoning in first-order logic: Tools, techniques and application areas. In Pavel Klinov and Dmitry Mouromtsev, editors, *Knowledge Engineering and the Semantic Web*, Communications in Computer and Information Science, pages 102–116. Springer.

[4] Ian Horrocks and Peter F. Patel-Schneider. Knowledge representation and reasoning on the semantic web: Owl. pages 365–398. Publisher: Citeseer.

[5] Qi Zhang, Jin Qian, Huan Chen, Jihua Kang, and Xuanjing Huang. Discourse level explanatory relation extraction from product reviews using first-order logic. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 946–957. Association for Computational Linguistics.

[6] Hans Kamp and Uwe Reyle. *From discourse to logic: Introduction to modeltheoretic semantics of natural language, formal logic and discourse representation theory*, volume 42. Springer Science & Business Media.

[7] Melvin Fitting. *First-Order Logic and Automated Theorem Proving (2nd Ed.)*. Springer-Verlag, Berlin, Heidelberg, 1996.

[8] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, USA, 3rd edition, 2009.

[9] Kenneth H. Rosen. *Discrete Mathematics and Its Applications*. McGraw-Hill Higher Education, 5th edition, 2002.

[10] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *J. ACM*, 7(3):201–215, July 1960.

[11] Handbook of tableau methods.

[12] Norbert E. Fuchs. First-order reasoning for attempto controlled english. In Michael Rosner and Norbert E. Fuchs, editors, *Controlled Natural Language*, pages 73–94. Springer Berlin Heidelberg.

[13] Norbert E. Fuchs, Kaarel Kaljurand, and Tobias Kuhn. Attempto controlled english for knowledge representation. In Cristina Baroglio, Piero A. Bonatti, Jan Małuszyński, Massimo Marchiori, Axel Polleres, and Sebastian Schaffert, editors, *Reasoning Web: 4th International Summer School 2008, Venice, Italy, September 7-11, 2008, Tutorial Lectures*, Lecture Notes in Computer Science, pages 104–124. Springer.

[14] Rainer Manthey and François Bry. SATCHMO: A theorem prover implemented in prolog. In Ewing Lusk and Ross Overbeek, editors, *9th International Conference on Automated Deduction*, pages 415–434. Springer Berlin Heidelberg.

[15] J. A. Robinson. A machine-oriented logic based on the resolution principle. *J. ACM*, 12(1):23–41, January 1965.

[16] Jon Barwise and Robin Cooper. Generalized quantifiers and natural language. In Jack Kulas, James H. Fetzer, and Terry L. Rankin, editors, *Philosophy, Language, and Artificial Intelligence: Resources for Processing Natural Language*, pages 241–301. Springer Netherlands.