March 2019

A Flexible, Natural Deduction, Automated Reasoner for Quick Deployment of Non-Classical Logic

Trisha Mukhopadhyay

University of South Florida, trishacse1001@gmail.com

Follow this and additional works at: https://digitalcommons.usf.edu/etd

Part of the Computer Engineering Commons

Scholar Commons Citation

This Thesis is brought to you for free and open access by the USF Graduate Theses and Dissertations at Digital Commons @ University of South Florida. It has been accepted for inclusion in USF Tampa Graduate Theses and Dissertations by an authorized administrator of Digital Commons @ University of South Florida. For more information, please contact digitalcommons@usf.edu.
A Flexible, Natural Deduction, Automated Reasoner for Quick Deployment of Non-Classical Logic

by

Trisha Mukhopadhyay

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Engineering
Department of Computer Science and Engineering
College of Engineering
University of South Florida

Major Professor: John Licato, Ph.D.
Lawrence Hall, Ph.D.
Eric Winsberg, Ph.D.

Date of Approval: March 11, 2019

Keywords: Automated Theorem Prover, First-Order Logic, Higher Order Logic, Codelets, Workspace, Speed Up Theorem

Copyright © 2019, Trisha Mukhopadhyay
DEDICATION

Dedicated to my late grandmother.
ACKNOWLEDGMENTS

I would like to express my heartiest gratitude towards my advisor, Dr. John Licato for his constant guidance and support throughout the research project. It was only his constant help, ideas, and encouragement that led to the work presented here. I would also like to thank the Advancing Machine and Human Reasoning Lab for helping me with its resources. I would like to thank the committee members Dr. Lawrence Hall and Dr. Eric Winsberg for their comments and feedback on the thesis.

I am thankful to Bradly Dragun for his contributions to MATR Parser, codelets and helping me on the speed-up theorem. I am thankful to Matthew Luker for his contributions to codelets. I am thankful to Elijah Malaby and Sean Jones for their contributions to MATR GUI, helping me in testing MATR and resolving the bugs. I want to thank Ashton Rodriquez for his contributions to MATR REST.

Last, but not least, I am grateful to my parents and my best friend Sauradeep for their support and encouragement which brought me this far.
TABLE OF CONTENTS

LIST OF TABLES ........................................................................................................ iii

LIST OF FIGURES ...................................................................................................... iv

ABSTRACT ................................................................................................................. vi

CHAPTER 1: INTRODUCTION/BACKGROUND .......................................................... 1

CHAPTER 2: FUNCTIONAL DETAILS ...................................................................... 5
  2.1 Workspace ........................................................................................................ 7
  2.2 Box .................................................................................................................. 7
  2.3 Nodes ............................................................................................................. 8
  2.4 Axioms ........................................................................................................... 11
  2.5 Goals ............................................................................................................. 11

CHAPTER 3: TECHNICAL DETAILS ........................................................................ 16
  3.1 Core ............................................................................................................... 17
    3.1.1 Parser ..................................................................................................... 17
    3.1.2 Codelets Recommender ......................................................................... 18
    3.1.3 Recommendation Resolver ..................................................................... 19
      3.1.3.1 Add Node ....................................................................................... 20
      3.1.3.2 Update Node .................................................................................. 20
      3.1.3.3 Delete Node .................................................................................... 20
      3.1.3.4 Add Box ......................................................................................... 20
      3.1.3.5 Update Box .................................................................................... 21
      3.1.3.6 Delete Box ...................................................................................... 21
      3.1.3.7 Check if Goals Achieved ................................................................. 21
      3.1.3.8 Checked Nodes ............................................................................. 21
    3.1.4 Communication with Codelets ................................................................. 21
    3.1.5 Connection with GUI .............................................................................. 23
  3.2 Codelets ......................................................................................................... 23
  3.3 REST Webservice .......................................................................................... 25
  3.4 GUI ................................................................................................................. 25
    3.4.1 Add Axiom ............................................................................................ 26
    3.4.2 Add Node .............................................................................................. 26
    3.4.3 Add Goal ............................................................................................... 26
    3.4.4 Run Prover ............................................................................................. 26
LIST OF TABLES

Table 1 Context Menu ................................................................................................................. 27
Table 2 Hotkeys with Functionalities in MATR GUI ................................................................. 27
LIST OF FIGURES

Figure 2.1 MATR Core and Codelets Interaction................................................................. 6
Figure 2.2 Fitch Style Versus MATR .................................................................................. 7
Figure 2.3 Super Box and Sub-Box in MATR Workspace ................................................. 9
Figure 2.4 Different Nodes in MATR Workspace................................................................. 10
Figure 2.5 Sample Proof – Goal is Not Checked Because the Top-Right Node is not Checked . 11
Figure 2.6 Goal is Achieved and Checked.......................................................................... 12
Figure 2.7 Sample Proof Where Box is Considered a Premise .......................................... 13
Figure 2.8 Step by Step Forward and Backward Reasoning in a Single Proof ...................... 14
Figure 3.1 MATR Block Diagram......................................................................................... 17
Figure 3.2 BNF Grammar Used by MATR FOL Parser......................................................... 18
Figure 3.3 JSON Object to Recommendation Resolver of MATR Core............................... 19
Figure 3.4 JSON Object from Core to Codelets ................................................................ 22
Figure 3.5 JSON Object from Codelets to Core ................................................................. 22
Figure 3.6 Application of Codelets based on the Source of the Node ................................. 24
Figure 3.7 MATR GUI Initial Page....................................................................................... 26
Figure 3.8 New Node Dialog MATR GUI ........................................................................... 28
Figure 3.9 Node Review Dialog MATR GUI ....................................................................... 28
Figure 3.10 Local Prover Dialog MATR GUI ....................................................................... 29
Figure 4.1 AND Elimination - Axiom and Goal is Added .................................................... 34
Figure 4.2 AND Elimination Inference Node with Achieved Goal....................................... 34
Figure 4.3 More Than One Axiom and Goals ................................................................. 35
Figure 4.4 AND Elimination and AND Introduction ...................................................... 36
Figure 4.5 FORALL Elimination Axiom and Goal is Added .......................................... 36
Figure 4.6 FORALL Elimination and AND Elimination ................................................. 37
Figure 4.7 Gödelian Speed Up Theorem .................................................................. 39
Figure A.1 JSON Object from MATR Core to Codelets .............................................. 44
Figure A.2 JSON Object from Codelets to Core ......................................................... 45
Figure A.3 MATR Core to MATR REST API or MATR REST API to GUI .................. 46
Figure A.4 REST API to MATR Core or GUI to REST API ........................................ 47
ABSTRACT

Automated Theorem Provers (ATP) are software programs which carry out inferences over logico-mathematical systems, often with the goal of finding proofs to some given theorem. ATP systems are enormously powerful computer programs, capable of solving immensely difficult problems. Currently, many automated theorem provers exist like E, vampire, SPASS, ACL2, Coq etc. However, all the available theorem provers have some common problems: (1) Current ATP systems tend not to try to find proofs entirely on their own. They need help from human experts to supply lemmas, guide the proof, etc. (2) There is not a single proof system available which provides fully automated platforms for both First Order Logic (FOL) and other Higher Order Logic (HOL). (3) Finally, current proof systems do not have an easy way to quickly deploy and reason over new logical systems, which a logic researcher may want to test.

In response to these problems, I introduce the MATR framework. MATR is a platform-independent, codelet-based (independently operating processes) proof system with an easy-to-use Graphical User Interface (GUI), where multiple codelets can be selected based on the formal system desired. MATR provides a platform for different proof strategies like deduction and backward reasoning, along with different formal systems such as non-classical logics. It enables users to design their own proof system by selecting from the list of codelets without needing to write an ATP from scratch.
CHAPTER 1: INTRODUCTION/BACKGROUND

Machina Arachne Tree-based Reasoner (MATR) is a Python-based natural deduction proof system designed to enable simultaneous reasoning of many types (deductive, inductive, analogical, etc.) over box structures. It is not sustainable to create a new prover for every new logic a researcher may want to test, such as cognitive calculi [1]; thus, instead of committing to a single proof strategy, the MATR framework provides a platform to work on multiple proof strategies like natural deduction with sub-proofs, partial proofs, or non-deductive methods like analogy, induction, backward reasoning, etc. Switching between different formalisms are possible in MATR. The “little workers” (known as codelets in MATR) that do most of the work are interchangeable, well defined, and modifiable.

One motivation to design and develop MATR is to provide a platform to logic researchers who want to continue research on both classical and non-classical logic, in the spirit of logical pluralism [2] by providing an automated reasoner which can be configured in such a way that no matter what the underlying logic or inference rules are, they could have it start reasoning very quickly. Logical Pluralism is the view that there is more than one correct logic [3], e.g., classical and intuitionist sentential logics, as well as logics dealing with new forms of expression like intentional logics and second-order-logics. As per Carnap’s view on logical pluralism [4], logicians should first develop a language as they like and judgment can be made later based on the outcomes [3]. Another motivation behind MATR is to support parallelism of a proof. Sometimes it may happen that a group of researchers, who are working on different parts of the same complex
proof, are physically not located in the same place. MATR allows multiple users to work on the same proof simultaneously, so that modifications made by one user can be seen by others.

To satisfy these motivations, MATR is designed to accomplish several goals. First, MATR should be easily usable by other programs, and automated reasoning agents. For example, a robot may want to make API calls to MATR to reason about something. Thus, MATR must have an API-only core reasoner (which we have been calling the backend) that stores everything necessary for reasoning, like the codelets (inference rules), the current state of the proof, etc. Backend operations are GUI independent; that is, the user might not use the GUI at all, or the user might not want to use our provided GUI at all, instead using their own custom GUI (in which case it would need to have access to the current proof state through the backend module).

Second, MATR’s backend should be entirely operable in local mode, without requiring any internet connection to run. Any network-related issues should be completely isolated from the backend module; and the module shouldn't even need to know whether it's operating in a network mode or not.

Third, MATR should also be easy to install by individual users who do not want any sort of complex installation process. As MATR backend is hosted through web services and GUI calls backend using REST API, user does not need to install the backend in their system. The entire installation process on the end-user’s side requires downloading only the MATR GUI package. The intended end-user of MATR is anyone who wants to quickly implement a reasoning system, test out a new logic, or test out new inference rules, so that they don't have to reprogram an entire prover from scratch every time. This could be a member of a research lab, students working on projects, etc.
Fourth, MATR should be usable by a group of people working together collaboratively on an extremely large proof. Let’s say, on one computer, someone might be trying to work on the proof through the GUI and the person selects some source nodes, a target node, some codelets, and eventually send a command to the reasoner to try to draw a proof between them. On another computer, someone else might be trying to manually re-arrange some nodes by dragging things around, re-sizing boxes, etc. for the same proof. Users should be able to do these things without worrying about synchronization conflicts.

The most developed types of automated theorem provers are first-order theorem provers, as First-Order Logic (FOL) is sufficiently expressive to capture concepts in an instinctive way. A number of fully automated systems have been developed using FOL [5] [6]. Although higher-order logics have higher expressivity than first-order logic, theorem proving in higher order logic is not as developed as first-order logic [7] (in no small part because of the limitations described by Gödel’s incompleteness theorems). A definitive automated deduction system exhibiting a capacity to solve mathematical problems was the Stanford Pascal Verifier created by David Luckham at Stanford University, which was dependent on the Stanford Resolution Prover [8].

Some important systems which have won at least one CASC (CADE ATP System Competition) [9], a yearly competition of first-order systems, are discussed below. A high-performance prover for full first-order logic is “E” based on purely equational calculus [10]. Another ATP system is “Otter”, based on first-order resolution and paramodulation [11]. A goal-directed, high-performance system is “SETHEO” [12] [13]. It is based on elimination calculus. “Vampire” which was developed by Andrej Voronkov and Krystof Hoder at Manchester University, has won the CADE ATP System Competition in the most prestigious CNF (MIX) division for eleven years (1999, 2001–2010), [14] [15]. “Waldmeister” was developed based on
unit-equational first-order logic. This has won the CASC UEQ division for fourteen consecutive years (1997–2010) [16]. Another first-order logic theorem prover is “SPASS” which is developed by Automation of Logic research group at the Max Planck Institute for Computer Science [17].

All the above-mentioned systems either are restricted to FOL, or heavily rely on interaction with the human users to supply lemmas, guide proofs, etc. From Gödel’s incompleteness theorems [18] we know that even an automated first-order theorem prover may not terminate while looking for a proof. However, much can be done with carefully-designed heuristics. With MATR, we hope to provide a platform for first-order and higher-order logics so that such heuristics can be quickly developed and tested in the future.
CHAPTER 2: FUNCTIONAL DETAILS

In this chapter, we will discuss the functional details of our in-house automated theorem prover, Machina Arachne Tree-based Reasoner (MATR). It is an open-source, platform-independent lightweight application with an easy to use graphical user interface. MATR is designed to enable simultaneous reasoning of many proof strategies like natural-deduction with subproofs, partial proofs, non-deductive methods (analogy, induction etc.) and backward reasoning over box structures.

Two of the most important functional units of MATR are the MATR Core and codelets. MATR Core accepts input from end users and performs a syntactic check with the help of a parser which is embedded into the Core. The Core initiates the workspace (details are in the later part of this chapter), recommends codelets (independent methods capable of performing specific tasks) without micromanaging them, sends the present status of the proof to codelets and modifies the proof workspace based on the suggestions from codelets. It also synchronizes the workspace among multiple users to support parallelism of the proof.

Codelets are small, specialized, easily-interchangeable programs which act independently. They are capable of performing everything from very specific jobs (such as executing individual inference rules) to extremely large jobs (such as codelets which embed entire ATPs). Codelets receive the state of the proof from MATR Core to analyze the workspace. They send suggestion to MATR Core to modify the proof workspace to achieve the goal. A list of codelets is associated
with each formal system in MATR. Thus, switching to a new formal system only requires changing the list of codelets, without having to modify MATR workflow and iteration process.

As pictured in Figure 2.1, a user enters axioms, and goals (both well-formed formula) to initiate the proof.

![MATR Core and Codelets Interaction](image)

**Figure 2.1 MATR Core and Codelets Interaction**

Users may select a set of inference rules they want to use in a particular proof. For each iteration, the system first looks at the current state of the proof and sends it to the codelets. Relevant codelets are applied to the formulas and they send suggestions to the Core. Based on these suggestions, the recommendation resolver (see section 3.1.3 for more details), one of the most important parts of the Core, modifies the proof workspace. If the proof is complete (i.e. if the goal is achieved), the system terminates the proof. In cases where the goal is not achieved, but no inference rule can be applied to any of the non-explored nodes or all the available nodes are explored (discussed in section 2.3), the system terminates the proof. But if more iterations are possible, the system continues to repeat the above-mentioned steps. When faced with undecidable problems, this may lead to an infinite loop that must be terminated through a predetermined timeout.
Below are the functional components of MATR in detail.

2.1 Workspace

The workspace is essentially the current state of the proof. With every new proof a new workspace is created, and the workspace is updated according to the progress of the proof, i.e. based on the formulas and applied inference rules. The workspace consists of a proof system ID (i.e. name of the proof strategy), logic ID (i.e. the name of the formal systems proof is using, which could be either classical or non-classical logic), list of the boxes, and list of nodes. In the following section more details are given about the box, and list of nodes (primary unit of a proof, for details check section 2.3 for more detail) within that workspace. In MATR, a user can have multiple workspaces.

2.2 Box

A box is a grouping of nodes, some of which may be distinguished as axioms or goals of that box. Every node is part of exactly one parent box. A MATR workspace for a single proof may
contain multiple boxes, and boxes can be nested within each other. An initial, root box, also called the “super box”, is created for every proof where axioms are provided by the user (see Figure 2.3). A box may have multiple children boxes which are known as “sub-boxes” in MATR (see Figure 2.3). For each subproof one sub-box is created.

The boxes are essentially isomorphic with Fitch style [5] natural deduction proofs. For an example of this, please refer to Figure 2.2, where each box in MATR style (refer proof at right hand side) corresponds with a vertical line in Fitch style (refer proof at left hand side). Indented vertical lines are created for each subproof in Fitch style. Likewise, sub-boxes are created in MATR style within the super box for each subproof.

Each box is associated with an auto-generated unique id (Box Id) for an entire proof. Every box has exactly one parent box and can have more than one child box. Apart from parent and children box Ids, every box consists of a list of premises, targets, and nodes (details are in the following subsection). Each box has a flag called “checked”. When a box is first created, it is not flagged as checked. A box is flagged as checked when all codelet iterations over that box are terminated. The checked flag of a box is different from the checked flag of a node (see section 2.3). Every formula appears only once inside of a box. Boxes themselves can be the sources and targets of edges. Moreover, it must be noted from Figure 2.3, that the reiteration codelet allows nodes to essentially be directly re-copied into sub-boxes.

2.3 Nodes

Nodes are the primary unit of a proof. There are two type of nodes: formula nodes (which contain formulas) and inference nodes (which displays inferences and link formula nodes), which are displayed as circles and rectangles, respectively. Where not otherwise specified, we will typically be referring to formula nodes, when we just say ‘nodes’. Formula nodes never point
directly to other formula nodes; likewise, inference nodes never directly point to other inference nodes.

Figure 2.3 Super Box and Sub-Box in MATR Workspace

The input formulas entered by the users are added to the initial box as nodes. Seven parameters are used to depict the state of a node within a box. One important property of a node is the “checked” flag. The premises of a proof are flagged as checked from the starting point. Nodes directly generated from premises are also flagged as checked. Once a node within a box is flagged checked, even after application of a chain of inference rules, it retains its checked property within that box for the entire proof. Core uses checked flags to know if a proof is done. If a node is flagged as checked, it means that the formula in the node follows only from checked nodes, or that the formula in the node is an axiom of its parent box. If a node is “checked” in a box, it will always be considered as checked within the children boxes as well (this property is not symmetric).

Every formula node, except for axioms, are pointed to by atleast one inference node. If a formula node $f$ is pointed to by an inference node $I$, the $f$ should be marked checked if (1) all
formula nodes pointing to $I$ are also checked, and (2) $I$ is a deductive inference. In the current version of MATR, all inference nodes correspond to deductive inferences.

Another important property of a node is the “explored” flag. Nodes, when all possible inference rules are applied at least once, are marked as explored. The content of a node contains formulas that were either entered by the users or generated from codelets. Nodes in MATR always have direct paths either from axioms or to goals$^1$. If the new node is directly generated from an axiom or from a node whose source is an axiom, the source of the new node would also be an axiom. In the same way, if the new node is directly generated from a goal or from a node whose source is goal, the source of the new node would be goals as well.

Each node is associated with a list of next nodes and a list of previous nodes. These help Core identify the position of a node within a box. At any point of time in a proof, a user can add, modify or remove a node explicitly from a box. In the next iteration, Core will consider the modified version of the workspace as the current state of the proof and transfer it to the codelets. Figure 2.4 shows how different nodes within a workspace look in MATR.

![Figure 2.4 Different Nodes in MATR Workspace](image)

Figure 2.4 Different Nodes in MATR Workspace

$^1$ This may not be the case for future versions of MATR, which may instead allow for codelets that introduce non-deductive inferences, or insert intermediate lemmas as part of “proof goals” which may not be connected to existing bodes in a workspace
2.4 Axioms

An axiom is a node in a box which is always flagged as checked within that box and any sub-boxes. One box can have more than one axiom. An axiom is the first node in a box. The nodes directly generated from the axiom are called next nodes of that axiom. In this chapter, the axioms are connected to the top edge of its parent box with a small line (Figure 2.5). This helps to separate the axioms from the rest of the nodes within that box.

2.5 Goals

The goal is a node in a box which is the target of that box. Initially, a goal is not flagged as checked. In MATR, an iteration can be initiated from a goal (backward reasoning, see in the later part of this chapter) but the nodes generated from a goal are not flagged as checked. The nodes directly pointing to the goal are called previous nodes of that goal. When all goals are checked, the proof is considered complete and iterations over that box are terminated. When all the goals of a super box are checked, the entire proof is terminated. In this chapter, the goals are connected to the bottom edge of their parent boxes with a small line (Figure 2.5). This helps to visually distinguish the goals from the rest of the nodes within that box.

Figure 2.5 Sample Proof – Goal is Not Checked Because the Top-Right Node is not Checked
Consider Figure 2.5., The node with content “ϕ” is an axiom and it is flagged as checked. The node with content “ϕ → ϕ” is neither an axiom nor a goal and it is not flagged as checked. The node with content “ϕ” is a goal within that box. Now the inference rule “modus ponens” is applied to the nodes “ϕ” and “ϕ → ϕ”. This is carried out by a codelet, which recommends a node with content “ϕ”. But as the content of goal is “ϕ”, and in a box, two nodes with same content cannot be present, thus, the inference node modus ponens directly points to the goal “ϕ”. Here the goal is not checked, as all the nodes pointing to the goal are not flagged as checked. Thus, we cannot say the goal is achieved yet.

![Diagram](image)

**Figure 2.6 Goal is Achieved and Checked**

Consider Figure 2.6 which is similar to the proof of figure 2.5 with a few exceptions. Here the node with content “ϕ → ϕ” is also an axiom, and so it is flagged as checked. As all the nodes pointing to the goal are checked, the goal “ϕ” is also flagged as checked. In MATR, codelets can have an entire box as a premise, referring to Figure 2.7. Here the codelet conditional introduction has considered the entire sub-box as a premise and has generated the node with content “a → b”.

12
Certain codelets (e.g. reiteration) can have premises in the parent box, and conclusions in the child boxes (Figure 2.3). But it cannot be the other way around.

MATR is capable of partial proofs and backward reasoning. Every non-checked node within the workspace can reason in a backward direction, which helps MATR to achieve goals quickly and efficiently. We will describe backwards reasoning next.

Referring to Figure 2.8, nodes colored blue are generated by backward reasoning and nodes with color green are generated by forward reasoning. Initially, the axioms “p” and “p→q” and the goal “q ∨ r” were in the workspace (Figure 2.8 (a)). The node “q ∨ r” was not checked. The system reasoned in the backward direction and the “∨ intro” codelet applied to it and generated two nodes “q” and “r”. Both the nodes “q” and “r” were not checked” as these are generated from a non-
checked node “q ∨ r” (refer Figure 2.8 (b)). The modus ponens codelet was applied to nodes “p” and “p → q” and points to the already existing node “q”.

Figure 2.8 Step by Step Forward and Backward Reasoning in a Single Proof
As both “p” and “p → q” were checked, node “q” was flagged as checked (Figure 2.8 (c)). As node “q ∨ r” is followed from node “q” which is now checked, the node “q ∨ r” is also flagged as checked (Figure 2.8 (d)).
CHAPTER 3: TECHNICAL DETAILS

In this chapter, we are going to discuss the design, architecture, workflow, and different modules of MATR along with the technologies used in the development, build, and deployment of MATR. While developing MATR, one of the biggest challenges was making decisions about the technologies to be used in MATR. One important consideration affecting this decision making process, was that MATR has to be platform-independent so that it can run in any operating system. The second consideration was that technologies needed to be chosen in such a way that a user could easily install MATR in their system without installing many external libraries and packages. To achieve this, it became necessary to perform all complicated jobs (like adding new boxes or nodes into the workspace, choosing inference rules, application of inference rules to a node etc.) in a backend server. In the current design of MATR, the GUI calls necessary functions from the backend using RESTful web services and displays the results in the front end. Thus, end users only need to install the GUI of MATR to start working on it. Use of REST API helped to serve another purpose of designing MATR, i.e. having an API-only core reasoner. So, the users who might not be interested in using MATR GUI or want to build their own custom GUI can use it. Moreover, it becomes easier for other applications or ATPs to use functionalities of MATR.

The entire MATR architecture can be divided into four modules – Core (MATRCore), Codelets (MATR Codelets), Web Services (MATRCoreREST) and Graphical User Interface (MATR GUI). Figure 3.1 visualizes the MATR architecture. We will discuss every module in detail in the following sections.
3.1 Core

MATR Core is the heart of the application with the built-in parser. Core communicates with all other components (e.g. codelets, GUI, etc.) through a REST API. The primary tasks of the Core are to initiate the workspace, recommend codelets, add or edit new nodes, and add or edit boxes in the workspace based on the recommendations made by the codelets, synchronize the workspace among multiple users, etc. The Core is written in Java 8. In the following paragraphs, every sub-module of the Core will be discussed in detail.

3.1.1 Parser

The parser is embedded with the MATR Core. The parser checks whether a user entered formula is in the syntactically correct form. A syntactically correct formula is known as a well-formed formula (wff). For a possible infinite alphabet of propositional symbols $\zeta$, the set of wffs is the smallest set such that any symbol ‘a’ is a wff, when $a \in \zeta$. If P and Q are wffs, then formulas “$\neg P$”, “P $\lor$ Q”, “P $\land$ Q”, “P $\rightarrow$ Q” etc. are also in wffs. If “P” is wff, then the formula “(P)” is also a wff.
The parser accepts input formulas in LISP-style S-expressions [19]. The S-expressions [20] of the parser are in parenthesized prefix notation [21]. The BNF grammar used by MATR’s FOL Parser is shown in Figure 3.2.

![BNF Grammar](image)

Figure 3.2 BNF Grammar Used by MATR FOL Parser

The parser detects the missing tokens along with wrong tokens in the input formulas and guides users to correct the formulas by throwing errors. The parser also helps the system to identify predicates, functions, and variables of the input formulas and stores these in the workspace for later use. In the present version of MATR, the parser is fully functional for the First-Order predicate logic. We anticipate, in the future, making the parser an interchangeable part of the system like codelets, so that non-FOL formulas can be parsed and considered wffs.

### 3.1.2 Codelets Recommender

Codelets recommender is an integral part of MATR Core, which recommends which inference rules should be applied to the workspace. If the user chooses inference rules explicitly from the GUI then codelets recommender sends those selected inference rules to codelets; otherwise, codelets recommender chooses the inference rules based on the formal systems selected by the user.
3.1.3 Recommendation Resolver

Recommendation Resolver is another of important parts of the Core. It receives details of the nodes, the boxes, and the name of the applied inference rules to the nodes from the codelet. It then updates the workspace accordingly. The input of the recommendation resolver is in JSON format, sent by the codelets. An input JSON looks like Figure 3.3:

```
{
   "andElim": [{
       "prevNode": "(AND (a) (b))",
       "nodeContent": "(a)",
       "source": "axioms",
       "actions": "addNode",
       "boxid": 1
   }, {
       "prevNode": "(AND (a) (b))",
       "nodeContent": "(b)",
       "source": "axioms",
       "actions": "addNode",
       "boxid": 1
   }]
}
```

Figure 3.3 JSON Object to Recommendation Resolver of MATR Core

The key of a JSON object is the name of the inference rule, which in this case is “andElim” (for AND Elimination). Other possible values of the key are “andIntro”, “forallElim”, “forallIntro”, “existElim”, “existIntro”, etc. based on the codelets which generated the current node (these are discussed in Chapter 2). The value is an array of JSON objects whose keys are “prevNode” (Previous node), “nodeContent” (the node generated after the inference rule of the key is applied), and “source” (the source of the new node), and can either be axioms or goals. Another two keys are “boxid” which denotes the id of the box where the current node is going to be added and “action”. In Figure 3.3, the action is defined as “addNode” which represents adding a current node “(a)” in the workspace where box id is 1, the previous node is “(AND (a) (b))”, and the source is “axioms”. Other possible values of “action” are “updateNode”, “deleteNode”, “addBox”, “updateBox”, “deleteBox” etc.
Following are the supporting methods which help the recommendation resolver to modify the workspace correctly based on the progress of a particular proof.

3.1.3.1 Add Node

This method is responsible for adding new nodes to the workspace based on box ID (discussed in Section 2.2), previous node and source of the node. Within a box, nodes are always unique. This method first checks if, within the same boxid, any other formula node with the same content exists or not. If the same node exists, then its previous node is updated as per the received JSON object. It also associates the inference node with the current node based on the inference rule which generated the current node.

3.1.3.2 Update Node

In MATR, a user can change any node within a workspace explicitly from the GUI. The details of the modified node, like the current content of the node, previous node, box id, and source are provided along with the action. In this case, the action” is “updateNode” which is sent to the MATR Core. This method modifies the workspace by updating the node.

3.1.3.3 Delete Node

In MATR, a user can delete a node from the workspace at any point in time. After node deletion, MATR will continue the proof based on the present workspace. Recommendation Resolver identifies a node to be deleted based on “box id”, “previous node”, “source” and “nodeContent”. Previous nodes and next nodes of that particular node are marked as “0”.

3.1.3.4 Add Box

This method is responsible to add a new box to the workspace. The box id of every box is unique in MATR. Once an add box request is received by the Recommendation Resolver along with the parent box id and child box ids, it modifies the workspace accordingly. As per the box
data structure, every box must have axioms and goals, which are added to the box after the creation of a new box. In the next iteration, nodes of this newly created box are considered and sent to the codelets for inference rule application.

3.1.3.5 Update Box

If a user wants to modify the axioms or goals of a particular box, updated nodes are sent to the MATR core along with the box id. In the next iteration, MATR considers the update box details.

3.1.3.6 Delete Box

If a user wants to delete a box, the box id of that box is sent to the backend and the parent box id and child box id of that box are marked as “0”.

3.1.3.7 Check if Goals Achieved

After adding a new node or updating or deleting a node, this method is called to check if the goal of a box is checked. If all the goals of a particular box are checked, iteration over that box is finished. When all goals of the super box are checked, the entire proof is terminated.

3.1.3.8 Checked Nodes

All the axioms in a workspace are considered checked. All the nodes whose source are axioms are also marked as checked. If, for a node, the previous node is checked, then even though the source of that node is “goals”, that specific node is marked as checked.

3.1.4 Communication with Codelets

MATR core communicates with codelets using a REST API. Codelets act as a REST server and Core as a REST client. Core sends all the unexplored nodes with the “boxid”, “content”, “previous node”, “next node”, and “source” to the codelets in JSON format. One example of a requested JSON array is shown in Figure 3.4.
Here two unexplored nodes “(AND (a) (b))” and “(c)” are sent to the codelets with “source”, “nextNodes”, “previousNodes” and “boxid”.

The response the codelets send to the core is shown in Figure 3.5.

From the above JSON, the core understands that for the inference “andElim” (AND Elimination), “addNode” (adding a new node into the workspace) is performed twice. In one case, node “(a)” is added to the boxid 1, where the previous node is “(AND (a) (b))” and source is
“axioms”. In the other case, the node “(b)” is added to the workspace where box id is 1, the previous node is “(AND (a) (b))” and the source is “axioms”.

3.1.5 Connection with GUI

The MATR core has a REST API suite on top of it. First, the GUI sends a request to core through MATRCoreRest and a response from the Core is also sent to the GUI through REST web services. The GUI then sends the workspace details in JSON to MATRCoreRest and MATRCoreREST calls the proofFormula-() methods of MATRCore. MATRCore then sends the Proof System Id, Logic Id, and a list of axioms and goals to Core. The Core then updates the workspace based on the iterations over the nodes. When the proof is terminated or all the nodes in the workspace are explored, Core sends the current workspace status to the GUI through REST web services in JSON format. Please see the sample JSON object from MATR Core to GUI in APPENDIX A for further details.

3.2 Codelets

Codelets are independently acting, limited programs that have specific jobs and abilities. In MATR, there are individual codelets for each of the inference rules. Codelets are written in Python 3.6 and Flask, and all codelets are hosted in the REST web service suite so that Core can easily send the required nodes from GUI to codelets to apply inference rules. As we are using the REST web service, the user does not need to install the codelets in their system; mere installation of the GUI will allow them to use the entire application. It must be noted here that despite installing just the GUI for MATR, an end user will be able to create and use custom codelets.

Each formal system is associated with a list of codelets. A user can also select a group of codelets in the GUI to create their own formal system. Under this situation, only the selected codelets will be used in the entire proof. If the user does not select any codelets from the list of
codelets from the GUI explicitly, all codelets associated with that specific logic system will be used in the entire proof. Codelets can work in parallel. Multiple codelets can be applied to a single formula.

Codelets input node details in JSON format and also return actions in JSON format based on how the core’s recommendation resolver changes the workspace. Please see the input and output JSON objects of codelets in APPENDIX A.

Each individual codelet takes boxid, node content and source of the node as input. Node content and source helps to choose a specific codelet. Refer to Figure 3.6, where in both the cases the input formula, is the same: “(AND (a) (b))”. In Figure 3.6(a) the source of the node “(AND (a) (b))” is the goals, so the prover reasoned in the backward direction and the name of applied codelet became “AND Introduction”. But in Figure 3.6(b), the node “(AND (a) (b))” is an axiom, so the prover reasoned in the forward direction and the name of the applied codelet became “AND Elimination”.

![Figure 3.6 Application of Codelets based on the Source of the Node](image)

(a) (b)
3.3 REST Webservice

On top of the MATR Core, there is a REST API suite which sends the request from GUI to the MATR Core and sends the response from the backend to the GUI. MATR Core REST is written in Java 8 and Apache Tomcat 7.0 is used to deploy the web services.

MATR REST Webservice is enabled with Cross-Origin Resource Sharing (CORS). This brings in more security to the system as it does not accept a request from other clients. Cross-origin resource sharing is a process that enables JavaScript on a website page to make solicitations to a domain other than the one where it has been initiated. The CORS strategy works by including explicit HTTP headers that tell the program that a downloaded site page ought to be permitted to make web solicitations to the given domains. It likewise, adds data to train a browser to permit just certain HTTP strategies (GET/POST) which are available on those domain URLs.

3.4 GUI

The MATR GUI helps users to interact with the MATR backend and visualize the proofs step by step. It calls the MATRCoreRest API to use methods from the Core and codelets. The technologies used in the MATR GUI is easy to use and interactive with drag and drop features where the user can initiate a proof, modify a proof and see the output.

The GUI sends a JSON message to the REST API to call methods from the backend and gets a response in JSON format. The GUI then parses the JSON message and draws the graph from it. A request from the GUI to REST API and response to GUI from REST API are in APPENDIX A.

The initial page of MATR GUI is as shown in Figure 3.7. Across the left sidebar are the primary controls. In order, they are:
3.4.1 Add Axiom

Adds a new axiom node to the workspace. This button allows clicking on the workspace to add a node, at which point the New Node Dialog opens up to request a formula for this node.

![MATR GUI Initial Page](image)

Figure 3.7 MATR GUI Initial Page

3.4.2 Add Node

Adds a new intermediate node to the workspace. (Same as above)

3.4.3 Add Goal

Adds a new goal node to the workspace. (Same as above)

3.4.4 Run Prover

Sends added axioms and goals from GUI to the backend and fills in any inferences the prover sends back. Any intermediate nodes can be deleted. At the bottom right are the controls for zooming and centering the graph.

3.4.5 Context Menu

Any node can be right-clicked to enter a context menu. Depending on the kind of node clicked, different options will be available. All of them in order are described in Table 1 below.
### 3.4.6 Hotkeys

Some keyboard buttons can be pressed for faster controls as shown in Table 2.

#### Table 1 Context Menu

<table>
<thead>
<tr>
<th>Context</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prove this</td>
<td>Start the Local Prover Dialog, for requesting inferences from the backend</td>
</tr>
<tr>
<td>Review Node</td>
<td>Opens the Node Review Dialog</td>
</tr>
<tr>
<td>Remove</td>
<td>Delete a node from the graph. This will also uncheck any nodes following it, and delete any edges connected to it</td>
</tr>
</tbody>
</table>

#### Table 2 Hotkeys with Functionalities in MATR GUI

<table>
<thead>
<tr>
<th>Hotkey</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delete</td>
<td>Remove a node from the graph</td>
</tr>
<tr>
<td>=</td>
<td>Zoom in</td>
</tr>
<tr>
<td>-</td>
<td>Zoom out</td>
</tr>
<tr>
<td>0</td>
<td>Zoom to fit</td>
</tr>
<tr>
<td>Enter</td>
<td>Add intermediate node</td>
</tr>
<tr>
<td>Ctrl+a</td>
<td>Select all</td>
</tr>
<tr>
<td>Escape</td>
<td>Unselect all</td>
</tr>
<tr>
<td>Ctrl+z</td>
<td>Undo</td>
</tr>
<tr>
<td>Ctrl+shift+z</td>
<td>Redo</td>
</tr>
</tbody>
</table>

### 3.4.7 New Node Dialog

When a new node is added, the New Node Dialog page will be opened. Enter a formula in the Input Formula box and confirm it when ready. The formula will be sent to the backend for
verification. If the backend accepts this formula, the node will be added to the proof graph. New Node dialog is shown in Figure 3.8.

![New Node Dialog MATR GUI](image)

**Figure 3.8 New Node Dialog MATR GUI**

### 3.4.8 Node Review Dialog

When the Node Review Dialog is opened, it presents various information. The user can review that information and update the node if required. Node Review dialog is shown in figure 3.9.

![Node Review Dialog MATR GUI](image)

**Figure 3.9 Node Review Dialog MATR GUI**
3.4.9 Local Prover Dialog

When the Local Prover Dialog is opened, it presents an interface for running a proof involving some subsection of the proof graph. In Figure 3.10, it has been opened on the node (b), and other nodes can be marked for proving by clicking on them. Clicking PROVE sends just those nodes to the backend and integrates any response into the visible proof graph.

![Local Prover Dialog MATR GUI](image)

Figure 3.10 Local Prover Dialog MATR GUI
CHAPTER 4: TESTING

Development of the first version of MATR for natural-deduction FOL, is complete. However, the development of of codelets for non-classical logic is at its very initial stage. All the backend modules like the MATRCoreProofer (Core part of MATR), MATRCodeletsREST (Codelets – inference rules), MATRCoreREST (REST suite on top of MATR Core) have been tested individually and found to be working as per the expectations in the preliminary testing phase. The next step is to test the application rigorously. We will discuss the test plan, deployment plan and some sample proofs in this chapter.

4.1 Test Plan

Before developing the test cases, it is important to have a test plan to know which functionalities should be working in the present version. Below are the strategies and functionalities which we have included in our test plan:

1. GUI testing should begin with a comparison of the appearance of MATR GUI with the mock-up screens of the design document which was prepared at the very initial phase of MATR GUI development. Also, we need to check if there is any error in the server log. Testing includes (a) a comparison of the list of codelets populated for FOL in the GUI with the expected list of codelets, (b) a feature to select few codelets from the populated codelet list, (c) checking functionalities of “add box”, “add node”, “prove formula”, and “step by step prove” available in the left sidebar of the MATR GUI (see Section 3.4), (d) checking functionalities of the items which
are available on node context menu like “Review Node”, “Remove Node”, “Prove This”, etc., (e) selecting a subsection of a proof and trying to achieve the target using only those subsections, (f) checking functionalities of hotkeys (see section 3.4.6), and (g) comparing the color code of nodes available in the GUI with the design document (see Figure 2.4). GUI is the topmost layer in MATR design and is directly interacted with by end users. So, if expected results from the GUI is obtained, it suggests that all modules of the backend would also work fine. Thus, integration of GUI with the backend is very crucial. In cases where the GUI does not provide expected results for any proof, the same proof from the MATRCore console and the MATR GUI will have to be tested separately. If both results are the same, then it can be said that the integration of front end with backend is likely correct, but if the results are not coherent, then the issue is with the backend. However, if the backend is correct, then debugging from MATRCoreREST and MATR GUI end have to be initiated.

2. Testing correctness of the parser is also very important as it guides the users to enter correct formulas from GUI. The parser should check the input formula based on the logic a user has selected; e.g., the formulas which are not correct for FOL may be correct for higher order logic. E.g. if a user enters the formula “(EXIST x (IFF (x) (P x)))” for first-order predicate logic, the FOL parser should throw an error because the quantifier “EXIST” is applied to a predicate, but in first order logic, a quantifier can only be applied to a variable. However, it should be noted that the same formula is acceptable in syntax of second-order logic. In the error message, the parser should print what is wrong in the user input and what the parser was
expecting. Let’s say if a user enters a formula as “(AND (a) b))”, an error message in the GUI should be “Error: expecting (, got b instead.” Moreover, for formula “(AND (a) (b)” the error message should be “Error: reached end of formula while expecting more content.”

3. It is important to test the inference rules along with the proof style. For this, it was decided to test (a) each inference rule individually, (b) proofs where multiple inference rules will be applied, (c) proofs which can be proved by backward reasoning (see Figure 2.8), (d) proofs which deal with multiple boxes and SOL formulas (see speed up theorem in the following subsection), and (e) proofs which can be proved by contradiction.

4.2 Deployment

The python based MATRCodeletREST has been successfully completed. The JAR file of MATRCoreProofer (MATRCoreProofer.jar) is exported and added to the MATRCoreREST. Apache Tomcat 7.0 has been installed and MATRCoreREST has been deployed onto it. No errors or warnings appeared in the deployment log of the Tomcat server. MATR GUI has been downloaded to the client machine and the config file has been updated with the IP address of the tomcat server where MATRCoreREST is deployed.

4.3 Test Execution

In this section, the progress with testing of MATR will be discussed.

4.3.1 GUI

The appearance of MATR GUI has been compared to the mock up screens and found to be exactly the same. Axioms, goals, nodes, and boxes have been added from the GUI. The details of a node can be reviewed by right-clicking on the individual node and selecting review option from
the context menu. The local prover dialog is working as expected; a few nodes from the workspace were selected to achieve a specific target. As the implementation of first-order logic (FOL) is now virtually complete, the codelets relevant to FOL are populated in the GUI and codelets from the list can be selected by the user.

4.3.2 Integration

The MATR GUI has been integrated with the MATRCoreREST and the proofs from GUI were tested. As the results from the GUI matched with the backend, it can be concluded that the integration of MATR GUI with the backend is successful.

4.3.3 Parser

The parser was only tested informally. A few formulas which are not wffs were entered to test the parser. Every time it displayed an appropriate error message (discussed on point 2 of Section 4.1) to guide the user. We anticipate the parser will be replaced with a more robust and customizable one in future versions of MATR.

4.3.4 Inference Rules

Tests for each inference rule, such as proofs with both forward and backward reasoning, proofs with multiple axioms and goals, and proofs which can be proved by contradiction, were performed. Every time the expected output was achieved. Below a few sample proofs are discussed.

4.3.4.1 Sample Proof 1

The axiom of the proof is “(AND (AND (a) (b)) (c))” and the goal of the proof is “(c)”. Figure 4.1 shows how the GUI appears after adding these nodes. “Run Prover” at the left sidebar is clicked to display the output.
In Figure 4.2 we can see that when the “AND Elimination” inference rule is applied to the axiom “(AND (AND (a) (b)) (c))”, the generated nodes are “(c)” and “(AND (a) (b))”. But in the given proof, the goal is “(c)” thus, another node with the same content “(c)” would not be added in the workspace, as two nodes with the same content cannot be present within the same box. The goal “(c)” is flagged as “checked” and the proof is terminated as expected.

Figure 4.1 AND Elimination - Axiom and Goal is Added

Figure 4.2 AND Elimination Inference Node with Achieved Goal
4.3.4.2 Sample Proof 2

In this proof, more than one axiom and goal are entered where both the axiom and goal are wffs. This proof requires shows both forward and backward reasoning (Figure 4.4).

Axioms of the proof are “(AND (a) (b))” and “(AND (c) (d))”. Goals of the proof are “(AND (a) (d))” and “(AND (b) (c))”. Please refer Figure 4.3 to see the workspace when multiple axioms and goals are added.

![Figure 4.3 More Than One Axiom and Goals](image)

When the prover is run by the user, “AND Elimination” is applied to both axioms “(AND (a) (b))” and “(AND (c) (d))”. “AND Introduction” is applied to both the goals “(AND (a) (d))” and “(AND (b) (c))”. It was observed that the system achieved the final stage on its own. Please refer to Figure 4.4 to see the proof in complete state. Both goals are flagged “checked” as these follow from the checked nodes.

4.3.4.3 Sample Proof 3

In this proof “FORALL Elimination” is tested. This proof progresses in the forward direction to achieve the goal.
The axiom of the proof is “(FORALL x (AND (P x) (S x)))” and the goal of the proof is “(P s)”. Please refer to Figure 4.5, which displays the initial state of the proof before the application of any inference rules.

Figure 4.4 AND Elimination and AND Introduction

Referring to Figure 4.6, first “FORALL Elimination” is applied to the axiom “(FORALL x (AND (P x) (S x)))” resulting in “(AND (P s) (S s))”. Next “AND Elimination” is applied to
“(AND (P s) (S s))” and one of the generated nodes is the goal “(P s)”. So, the proof achieves the goal “(P s)” and it is flagged as checked.

Figure 4.6 FORALL Elimination and AND Elimination

4.3.4.4 Speed Up Theorem

So far we have seen that MATR is working well for FOL formulas. As the design motivation behind MATR was to provide researchers with a platform to reason over multiple formal systems, it is important to show that other than FOL, MATR can work efficiently for other formal systems like second order logic (SOL). It is not always enough to achieve the goal by iterating over the input formulas, for a larger proof with multiple axioms and goals; sometimes it is more important to achieve the goals quickly and efficiently. In a framework like MATR with more than 30 codelets, and where system can use nested boxes, the choice of codelets plays an important role to simplify the proof and achieve the goals quickly. We have tried to prove the Gödelian Speedup Theorem [22] by using MATR. This is meant as a proof-of-concept, as its proof uses atleast one SOL formula, and a nested box. Here we use a simplified version of the speedup theorem inspired by Peter Smith’s notation (Theorem 17.2) [22], which uses Quine corners to represent the overloaded primitive recursive function that transforms well-formed formula(w.f.f.)
into objects in our domain of discourse. For example, if $\varphi$ is a w.f.f. in our logical system, then $B_f(\varphi)$ is not a w.f.f. But $B_f(\neg \varphi)$ is a wff, since $\neg \varphi$ is an object corresponding to a natural number. We also use the notation of drawing a line above an integer object to represent the Gödel numeral of that integer. For details, see [22].

We seek to prove if $T$ is a “nice” theory (using Smith’s definition of nice), then for any primitive recursive function $f$, there is a w.f.f. $\varphi$ such that: $T \vdash \neg B_f(\varphi)$ and $T \vdash \varphi$. The following definitions are necessary in order to make this proof work:

Definition 1(D1): $LE_f(\gamma, \bar{u}) \equiv_{\text{def}} \exists_k (F(\gamma, k) \land u \leq k)$

Definition 2(D2): $B_f(\gamma) \equiv_{\text{def}} \exists_u (LE_f(\gamma, \bar{u}) \land Prf(\bar{u}, \gamma))$

Definition 3(D3): $NB_f(\gamma) \equiv_{\text{def}} \neg B_f(\gamma) \equiv \forall u \neg LE_f(\gamma, \bar{u})$

Definition 4(D4): Proves $(\gamma) \equiv \exists_u Prf(\bar{u}, \gamma)$

In MATR, node content may be replaced by equivalent content from the definitions. In the current version of MATR definitions are already declared in a proof. The user can explicitly enter the definitions at the very beginning of the proof. For simplicity, a simplified version of the definitions have been used to prove the speed up theorem, e.g. $\gamma$ is represented by QB$_{\neg f}$, $\bar{u}$ is represented by bar_u etc.

The axiom of the proof is “((EXIST y (IFF (y) (NB_f QB_y))))”. Here the existential quantifier is applied to the predicate, thus making it a formula outside the scope of classical FOL. Again, this is a proof-of-concept step demonstrating MATR’s usefulness as a platform to test both classical and non-classical logics.

Figure 4.7 pictures the screenshot of the proof obtained from MATR. The system has used a “proof by contradiction” strategy to achieve the goal.
Figure 4.7 Gödelian Speed Up Theorem
CHAPTER 5: CONCLUSION

In the above chapters, functionalities and technical details of the current version of MATR have been discussed. We have also seen a few sample proofs along with a proof of the speed up theorem using non-classical logic. Based on the tests in chapter 4, we can conclude that the first version of MATR has achieved most of its design goals (see Chapter 1, Introduction/Background for more details). MATR is an open-source, automated theorem proving platform which can be installed in any operating system such as Windows, MacOS, Linux etc. Users have the flexibility to change the proof strategy and formal system easily. Independently acting codelets are associated with each formal system and help the user to choose their own hybrid formal system by choosing a list of codelets from the GUI.

MATR makes it easy for a group of researchers to collaborate on an extremely large proof. Moreover, reasoning in both forward and backward directions helps the system to find goals much easier. Apart from these, the installation steps of MATR are simple. All the backend projects are hosted through RESTful web services and the end users only need to download the MATR GUI package. MATR gives the flexibility to end users to use their own custom GUI since the functionality of MATR backend is completely independent of MATR frontend.

Although MATR achieves most of its design goals, we can still consider a few enhancements on top of the current version of MATR. First, while codelets necessary for FOL have been completed, we can still continue our work to develop codelets for non-classical logics. Second, implementation of core-level heuristics to choose the right codelets (see section 3.1.2) for
a proof is a future area of research for the optimization of a proof. Third, in the present version of MATR, no database is associated with it, so there is no provision to store the state of proof. In the next version of MATR, we plan to add a database on the MATR REST API layer so that it can store the state of proof for later use. Fourth, with respect to the GUI, the next plan is to implement an autocomplete feature so that based on the user's input, the system could provide suggestions to correct or complete the formula. It will reduce the chance of users writing wrong input formulas.
REFERENCES


APPENDIX A: JSON OBJECTS

In this section, we can see the input and output JSON objects for MATR Core, Codelets, MATR Core REST web services, and GUI.

A.1 MATR Core to Codelets

```
[{
  "checked": true,
  "explored": false,
  "content": "(AND (a) (b))",
  "source": "axioms",
  "nextNodes": [],
  "previousNodes": ["(AND (AND (a) (b)) (c))"],
  "boxid": 1
}, {
  "checked": true,
  "explored": false,
  "content": "(c)",
  "source": "axioms",
  "nextNodes": [],
  "previousNodes": ["(AND (AND (a) (b)) (c))"],
  "boxid": 1
}]
```

Figure A.1 JSON Object from MATR Core to Codelets
A.2 MATR Codelets to Core

Figure A.2 JSON Object from Codelets to Core
A.3 MATR Core to MATR REST Web Services/ MATR REST Web Service to GUI

Figure A.3 MATR Core to MATR REST API or MATR REST API to GUI
A.4 REST API to MATR Core/GUI to REST API

```
[
'checked': False,
'explored': False,
'content': '{AND (AND (a) (b)) (c)}',
'source': 'axioms',
'nextNodes': [],
'previousNodes': ['0'],
'boxid': 1
],
[
'checked': False,
'explored': False,
'content': '{b}',
'source': 'goals',
'nextNodes': ['0'],
'previousNodes': [],
'boxid': 1
]
```