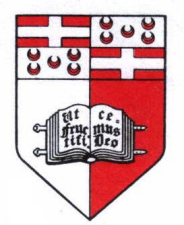
Implementing Proof Systems for the Intuitionistic Propositional Logic

Veronica Zammit

Supervisor: Dr. Adrian Francalanza



Faculty of ICT
University of Malta

May 27, 2011

Submitted in partial fulfillment of the requirements for the degree of B.Sc. I.C.T. (Hons.)

Faculty of ICT

Declaration

I, the undersigned, declare that the dissertation entitled:

Implementing Proof Systems for the Intuitionistic Propositional Logic

submitted is my work, except where acknowledged and referenced.

Veronica Zammit

May 27, 2011

Acknowledgements

This final year project would not have been possible without the help and assistance through the guidance of my supervisor, Dr. Adrian Francalanza, who provided the relevant material, knowledge and guidance for my study during these last few months. I am very grateful for his support and concern.

Finally, I thank my family and friends for being there when I needed them the most, for their support, love and advice. Thank you so much.

Abstract

Logical reasoning is something human beings consistently utilise to make decisions. The simplest calculus for proof search is the Natural Deduction. However, it is harder for an automated system to automate proof search using this calculus. At every step it has to decide the next step on the basis of:

- which inference rule to apply at every decomposition level?
- what sub goals should we prove in our pursuit to prove the final goal?
- if the proof search is in a stuck state should we backtrack the proof search or terminate it?

Analysis shows that if the proof search is carried out using the Sequent Calculus the number of choices required at each step are reduced. Hence our approach to minimise the high level of non determinism in Natural Deduction is to compute the proof search in the Sequent Calculus then translate it to the Natural Deduction.

The advantage of this approach is that although the final proof search is expressed in the Natural Deduction Calculus the actual proof search is computed in the Sequent Calculus which reduces non determinism during proof computation. Furthermore, for the points highlighted above proof search in the Sequent Calculus eliminates the possibility of backtracking and restricts the number of inference rules applicable for certain propositions. Another advantage of this approach is that since the proof search is carried out in the Sequent Calculus, if the proof search in the Sequent Calculus is correct, by the correspondence between the Sequent Calculus and the Natural Deduction we are guaranteed that the same proof translated to Natural Deduction is also correct.

Contents

1 Introduction **6**

1.1 Project Motivation 6

1.2 Problem Analysis 7

1.3 Proposal 8

2 Background **11**

2.1 Propositional Logic for Intuitionistic Logic 11

2.1.1 Intuitionistic Logic 11

2.1.2 Logic Operators 12

2.2 Model Theory 13

2.3 Proof Theory 14

3 Formal Logic **17**

3.1 Natural Deduction 17

3.1.1 Inference rules for Natural Deduction 17

3.1.2 Top Down and Bottom Up Techniques for Proof Search 19

3.1.3 Examples 20

3.1.4 Problems in Natural Deduction 22

3.2 Intercalation Calculus 25

3.2.1 Inference rules for Intercalation Calculus 25

3.2.2 Examples 26

3.2.3 Intercalation Calculus' Advantages 28

3.2.4 Problems in Intercalation Calculus 28

3.3 Sequent Calculus 30

3.3.1 Inference rules for Sequent Calculus 30

3.3.2 Examples 32

3.3.3 Sequent Calculus' Advantages 33

3.3.4 Problems in Sequent Calculus 34

3.4 Proof Terms 35

3.4.1 Inference Rules for Proof Terms 35

3.4.2 Reduction 36

3.4.3 Example 37

3.4.4 Analysis . . . 38
3.5 Conclusion . . . 40

4 Disjunction Logical Connective **41**
4.1 Inference Rules for Disjunction . . . 42
4.1.1 Natural Deduction . . . 42
4.1.2 Analysis . . . 44
4.1.3 Intercalation Calculus . . . 44
4.1.4 Proof Terms . . . 46
4.1.5 Sequent Calculus . . . 48
4.2 Analysis . . . 50

5 Proof Search Procedure **51**
5.1 Reducing Non Determinism in Proof Search . . . 51
5.1.1 The Cut-Elimination Rule . . . 51
5.1.2 Analysing the relation between the Natural Deduction, the Intercalation Calculus and the Sequent Calculus . . . 52
5.1.3 Choice of antecedent . . . 56
5.1.4 Focusing . . . 56
5.2 Methodology . . . 57
5.2.1 System's Functionality . . . 57
5.2.2 Proof Search Transition Methodology . . . 58
5.3 Conclusion . . . 62

6 System Implementation **64**
6.1 Language Selection . . . 64
6.2 Implementation Overview . . . 64
6.3 Data Structure . . . 65
6.4 Module Description . . . 66
6.4.1 Parser . . . 66
6.4.2 Automated Theorem Provers . . . 70
6.4.3 Proof Search Translators . . . 73
6.5 Achievements . . . 76
6.6 Errors and Limitations . . . 77
6.6.1 Notation . . . 77
6.6.2 Limitations . . . 77
6.6.3 Errors . . . 78

7 Evaluation **81**
7.1 Introduction . . . 81
7.2 Testing Plan . . . 81
7.3 Proof Correctness . . . 82

7.3.1 Logical Connectives' Inference Rules Evaluation . . . 83
7.3.2 Proof Terms . . . 89
7.3.3 Proof Search Evaluation (using combinations of logical connectives) . . . 92
7.3.4 Invalid Judgments . . . 93
7.4 Proof Search Quality . . . 93
7.5 Stress Testing . . . 94
7.6 Reduced Non Determinism from Proof Search . . . 96
7.6.1 Translating from the Intercalation Calculus . . . 97
7.6.2 Translating from the Sequent Calculus to the Intercalation Calculus . . . 97
7.7 Performance . . . 101
7.8 Conclusion . . . 107

8 Conclusion **108**
8.1 Reduce Non Determinism from Proof Search . . . 108
8.2 Proof Search Correctness . . . 109
8.3 Future Work . . . 110
8.3.1 Loop Checking module . . . 110
8.3.2 More restrictive Sequent Calculus . . . 110
8.3.3 Extending the system to include Predicate Logic Proof Searching . . . 111
8.3.4 Interface Improvement . . . 111

A Inference Rules **112**
A.1 Natural Deduction . . . 112
A.2 Intercalation Calculus . . . 112
A.3 Sequent Calculus . . . 112
A.4 Proof Terms . . . 113

B Proof for Theorem 6 **114**

C Proof Search Referred to in Section 7.4 **117**

D CD Contents **120**

List of Figures

1.1 Proof Search Tree: Sequence of Hypothetical Deductions 8
1.2 Proof Search Transition 9
1.3 Graphical Representation of the Proposed System. (The Intercalation Calculus is an intermediary calculus that facilitates the transition from the Natural Deduction to the Sequent Calculus.) 10

2.1 Truth Tables for the logical connectives $\wedge$,$\vee$ and $\rightarrow$ 14

3.1 Sequent Calculus Transformation [19] 30

5.1 Proof Search Transition 53

6.1 General Design: Module Interaction 65
6.2 Proposition Tree for (A$\wedge$(B$\wedge$C)) 66

7.1 Graph of Number of Reductions against Test Cases for the Intercalation Calculus and Sequent Calculus 106
7.2 Graph of Number of Reductions against Test Cases for methods 1 and 2 106

8.1 Graphical Representation of the system to reduce non determinism during proof search in Natural Deduction. 109

List of Tables

2.1 Precedence Tables for the logical connectives $\wedge$,$\vee$ and $\rightarrow$ 13
2.2 Truth table for the expression (A$\wedge$B)$\models$(B$\wedge$A) 14

6.1 Lexeme and Token correspondence 67
6.2 Step-by-Step Parsing 70
6.3 System's Logical Connective Representation 77

7.1 System's Logical Connective Representation 82

1. Introduction

1.1 Project Motivation

Academia has long acknowledged that there are several real world problems that are hard for humans to solve. The reasons are that humans are error prone and some problems require a reasonable amount of information handling which humans may not have time and resources available to handle these problems.

Logic is the study of correct reasoning, something we as human beings use to help us choose one path and reject another. In our daily life, before we take decision we evaluate known information and the possible outcome of every decision to try to choose the best path.

We argue that logic is formal, but what do we exactly mean? To analyse logic formality we take different scenarios. For instance if one says:
Scenario 1:

$$\textit{It is raining and he has an umbrella.} \tag{1.1}$$

$$\textit{If it is raining then it is bad weather.} \tag{1.2}$$

For logic it is no different than saying
Scenario 2:

$$\textit{John is a gardener and a carpenter.} \tag{1.3}$$

$$\textit{If John is a gardener then he knows how to plant plants.} \tag{1.4}$$

Although both scenarios have a different meaning, we reason about them in the same way.

For instance, to conclude that *It is bad weather* from the first scenario we need to know that *It is raining*. Such information is extracted from the known information (1.1). Similarly, if we want to know that *John knows how to plant plants*, we have to verify that *John is a gardener*. Again, such information is given and we can extract it from (1.3).

We can represent these scenarios formally, in terms of identifiers. For example, we can reduce scenario 1 as:

$$\text{P} = \textit{It is raining.}$$
$$\text{Q} = \textit{He has an umbrella.}$$
$$\text{R} = \textit{It is bad weather.}$$

$$\textit{It is raining and he has an umbrella.} = P \textit{ and } Q$$
$$\textit{If it is raining then it is bad weather.} = \textit{If } P \textit{ then } R$$

Hence, to conclude *It is bad weather (R)* we can argue that knowing that *P and Q is true* then we know that *P is true* and *Q is true*. Knowing that *If P is true then R is true* and *P*

is true then we know that *R is true.* Although we argued using *P, Q* and *R* in terms of this scenario, *P, Q* and *R* are just identifiers which can represent any scenario.

If we choose to represent the second scenario using *P, Q* and *R* we end up with the same information that is

$$\text{P} = \textit{John is a gardener.}$$
$$\text{Q} = \textit{John is a carpenter.}$$
$$\text{R} = \textit{He knows how to plant plants.}$$
$$\textit{John is a gardener and a carpenter.} = P \textit{ and } Q$$
$$\textit{If John is a gardener then he knows how to plant plants.} = \textit{If } P \textit{ then } R$$

Thus to reason about *He knows how to plant plants (R)* we follow the same reasoning using *P, Q* and *R* as we did for Scenario 1.

Therefore, formality is when we argue about relations between conjunctions *(and)* and implications *(if...then)* in form of thoughts irrelevant what the meaning of these thoughts is [15]. In fact, in both cases we reduced the given facts into syntax and we just manipulated syntax to extract information. The advantage of this approach is that if we verify that a relation between two identifiers hold then every scenario that can be represented with this particular relation holds.

1.2 Problem Analysis

One of the easiest techniques for proof search in propositional logic is Natural Deduction. This kind of proof calculus defines two types of inference rules; introduction and elimination rules. These inference rules and axioms are applied systematically to hypotheses (a collection of propositions specified at the beginning of the proof search) and hypothetical derivations (deduced propositions from hypotheses or other derivations) to produce a formal proof of the conclusive proposition.

Let us, for simplicity reasons, consider the conjunction logical operator from the Natural Deduction. We introduce rules that allow us to eliminate and introduce the conjunction operator. We know that for a statement *A and B* to hold we should know that both *A* and *B* hold on a separate basis. On the other hand, knowing that statement *A and B* holds we know that statement *A* holds on its own and statement *B* holds on its own. Thus, we represent this information into rules which can be applied whenever a statement contains a conjunction (denoted by $\wedge$).

$$\frac{A\ true \quad B\ true}{A \wedge B\ true} \wedge_I$$

$$\frac{A \wedge B\ true}{A\ true} \wedge_{E1} \quad \frac{A \wedge B\ true}{B\ true} \wedge_{E2}$$

Example 1.2.1. Given:

$$\textit{It is raining and he has an umbrella} \tag{1.5}$$
$$\textit{If it is raining then it is bad weather} \tag{1.6}$$

RTP: Is it true that *It is bad weather*?

If we had to verify whether this statement is true using an automated theorem prover, the theorem prover would have produced a tree similar to figure 1.1. In its attempts to prove

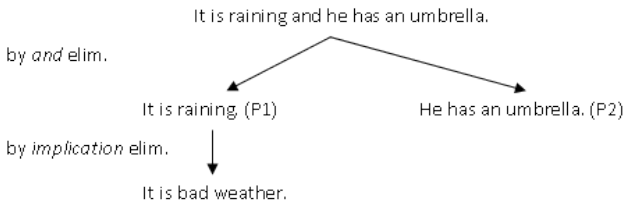


Figure 1.1: Proof Search Tree: Sequence of Hypothetical Deductions

whether the statement *It is bad weather*, the automated function has to obtain statement *It is raining*. It starts from statement (1.5) and the function has to choose which rule to apply for decomposition between $\wedge_{E1}$ and $\wedge_{E2}$. In no particular order, it chooses to apply the rule $\wedge_{E1}$ and extract the statement *It is raining*. With this information and the information in (1.6) an automated system concludes that the statement *It is bad weather* is true.

However, if the system opts to apply the rule $\wedge_{E2}$ on statement (1.5) then it extracts the statement *He has an umbrella*. At this point the system reaches a *stuck state* that is the decomposition has diverted from the main goal and further decompositions from this statement are redundant. Therefore, in such situations the system has to backtrack.

Since there exists two rules that for the same premise (known statement) these output different statements, during computation the system usually ends up with redundant statements. This was the case when in example 1.2.1 the system applied both the rules $\wedge_{E1}$ and $\wedge_{E2}$ where the application of $\wedge_{E2}$ is redundant. For such redundant cases the system has to backtrack.

Example 1.2.1 points out the fact that using this calculus (Natural Deduction) the next computation is underspecified. After every operation the system has to choose the next step to apply:

- should the system backtrack or decompose a statement?
- if the system opts to decompose which rule should it apply for decomposition?

Thus, in Natural Deduction we have to constantly evaluate the context (through backtracking the proof search) to choose the best proposition. The resultant arbitrary choices (non determinism) are imposed by the definition of the inference rules for this kind of calculus.

1.3 Proposal

Proof search automation using the Natural Deduction calculus is not ideal because the nature of the calculus itself introduces multiple ways how the proof search can continue. Ideally, we should use a calculus which by the definition of its rules guides us how to conduct the proof search

[6] and minimises the unpredictable computation during proof search. Gentzen[9] therefore introduced the Sequent Calculus, a refined version of the Natural Deduction.

This calculus defines new rules for logical operators; "left" rules (equivalent to introduction rules) and "right" rules (equivalent to elimination rules). By the definition of these rules the calculus guides who ever is computing the proof search which path should one follow. Furthermore, by the definition of the calculus itself a proof search never backtracks, which means that if a proof search is in a stuck state then it terminates.

There exists a relation between the Natural Deduction and the Sequent Calculus in such a way that propositions in the Natural Calculus can be translated into equally expressive expressions in the Sequent Calculus and vice versa [19]. The relation between the Natural Calculus and the Sequent Calculus is defined by Soundness for the Natural Deduction. Soundness for the deductive system S with respect to the deductive system S' is the property that every true proposition in S is also true in S' [18]. Thus, every true proposition in the Natural Deduction is also true in the Sequent Calculus. More important is the fact that by completeness (Completeness for the deductive system S with respect to the deductive system S' is the property that every true proposition in S' is also true in S.[18]) for Natural Deduction we know that every true proposition in the Sequent Calculus is also true in the Natural Deduction.

Therefore, to limit any computation problems in the Natural Deduction it is more appropriate to translate the expression from Natural Deduction to Sequent Calculus, produce the proof search in the latter environment and then, without losing any formality in the proof, it is translated back to Natural Deduction (graphically represented in figure 1.2). Using such methodology we are guaranteed correctness for the proof search outputted in the Natural Deduction. Thus, the system has to check correctness for the proof search computed in the Sequent Calculus only because correctness for the proof search in the Natural Deduction then follows.

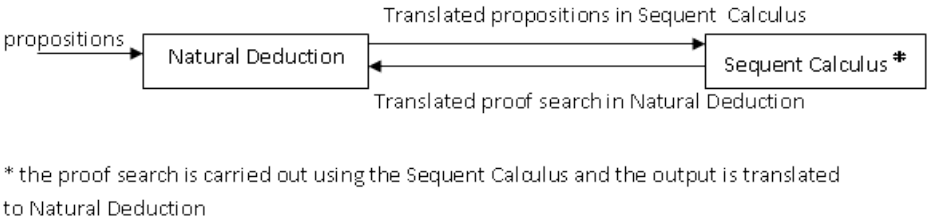


Figure 1.2: Proof Search Transition

Finally, the proof search will also be outputted in terms of the lambda calculus. The Curry-Howard isomorphism[10] is the relation between programs and proofs where a proof is the type for the program. Moreover, the lambda calculus representation of the proof gives a new practical perspective to the system. The type assigned to a particular judgment is the lambda encoding for the proof search for that judgment. Thus, by the one-to-one correspondence between the Natural Deduction calculus and Proof Terms, from the type assigned to the judgment one can compute the proof search in Natural Deduction.

Primarily, the proposed system is an automated theorem prover with the main aims are to

produce a correct proof search in the Natural Deduction and to find a methodology to reduce non determinism for proof search in Natural Deduction. Nevertheless, with the addition of lambda calculus representation of the proof, the system will show the relation between proofs and programs. Thus the aims for this project are:

1. Produce an automated theorem prover (ATP) for the Natural Deduction calculus. This ATP should provide a better automated proof searching in comparison to proof searches computed directly in the Natural Deduction. Nonetheless, the proof searches computed by this ATP should be correct, precise and concise that is redundant decompositions should be omitted in order to produce a concise proof search which is easier for the reader to comprehend.

2. Produce an encoding in lambda calculus which represents the actual proof search. This will show the actual relation between proofs and programs by the representation of types for proof searches.

The proposed system is illustrated in figure 1.3.

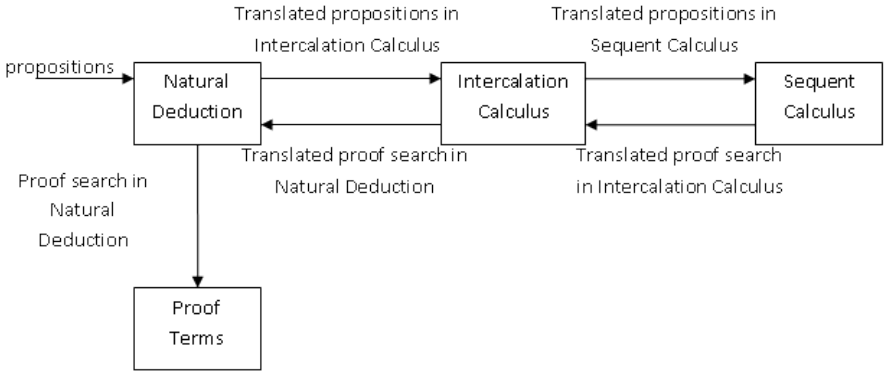


Figure 1.3: Graphical Representation of the Proposed System. (The Intercalation Calculus is an intermediary calculus that facilitates the transition from the Natural Deduction to the Sequent Calculus.)

2. Background

2.1 Propositional Logic for Intuitionistic Logic

Propositional logic is all about reasoning on formal structures according to predefined list of axioms and inference rules [4]. At its core, it is based on judgments, propositions and the basic formalization of proving that A is true or false. A judgment can be a fact which evidence exists for. For example, *The wall is white* is a valid judgment because by looking at the wall we can establish whether the judgment is true or false. The action of looking at the wall is the evidence for the judgment. Additionally, deducing whether the judgment is true or false is what forms the proposition.

In propositional logic we usually translate a declarative sentence into a symbolic character to abstract away certain information so that we can reason about the relations between the propositions rather than the actual meaning of the propositions. For example, the information *The wall is white* is usually defined as proposition A.

2.1.1 Intuitionistic Logic

Intuitionistic logic is the application of intuitionism in logic. Logic is based on reasoning, understanding whether a judgment is true or false and giving logical meaning to logical connectives. Hence, from the intuitionistic point of view, a statement is either true or false if there exists a verification (proof) that sustains the truth or falseness of the statement. This definition makes it immediately clear that both the classical laws of the excluded middle and double negation elimination are not axioms in intuitionistic logic.

Law of Double Negation

Usually, in Classical Logic knowing that

It is not true that John does not like baseball

one concludes

John likes baseball.

From an intuitionist point of view this is unacceptable because by definition, for a judgment to be true there must exist enough information to sustain it (proof). In this case the known information is *It is not true that John does not like baseball* but we do not have enough information to conclude *John likes baseball.* For intuitionists it is not acceptable that if ¬ A leads to a contradiction A must be *true* (*not* is denoted by the symbol ¬)[29]. Therefore at no point during proof search the decomposition $P = \neg\neg P$ is allowed.

Law of Excluded Middle

For Classical logic the proposition *John speaks English OR John does not speak English* (P3) is always *true*. The reason is that if the proposition *John speaks English is true* then proposition *(P3) is true*. If the proposition *John speaks English is false* then the proposition *John does not speak English is true* for which *P3* is also *true*. Therefore, in any case for Classical logicians the proposition *John speaks English OR John does not speak English* is always *true*.

On the other hand, by the Intuitionistic Logic definition the proposition *John speaks English OR John does not speak English* is true only if there is enough information that sustains the propositions. This means that the introduction of statements like *(P3)* during proof search without them being justified is disallowed in proof searching for Intuitionistic Logic.

2.1.2 Logic Operators

Propositions can be combined using a number of logical operators (logical connectives) as defined by the following definitions:

- $\wedge$: Given two propositions, A and B, and both are proved to be true then their conjunction is denoted as $A \wedge B$. Example: *The wall is white and the floor is slippery* is a valid conjunction as long as the propositions *The wall is white* and *The floor is slippery* are both justified (proven) to be true.

- $\vee$: Given two propositions, A and B, we conclude that $A \vee B$ is true if at least one of them is true. Therefore the declarative sentence *The wall is white or it is grey* can be denoted as $A \vee B$ (the disjunction of A and B) if either of A and B is true.

- $\rightarrow$: Given two propositions, A and B, $A \rightarrow B$ means that B is a logical consequence of A that is *If A then B*. An implication is considered to be false for the case when A is true and B is false. Otherwise the implication $A \rightarrow B$ is true. Example: *If a student passed the exam then, the student must have studied for the exam*

- $\neg$: The negation of the proposition A is usually denoted as $\neg A$. However in this document the negation of any proposition will be considered as an implication between the specified proposition and falsehood ($\perp$). In other words, for the negation of a proposition to be true the actual proposition must be false. Thus, if $\neg A$ is true then A must be false. Example: the proposition *The student has not passed the exam* is equivalent to say *It is false to say that the student has passed the exam* ($A \rightarrow \perp$).

Table 2.1 shows the precedence of the logical connectives, starting from the one with the highest precedence. The negation operator is omitted because as described the negation is considered to be an implication with falsehood being the logical consequence of the proposition.

Precedence	Logical Connective
1.	$\wedge$
2.	$\vee$
3.	$\rightarrow$

Table 2.1: Precedence Tables for the logical connectives $\wedge$,$\vee$ and $\rightarrow$

2.2 Model Theory

By the definition of propositional logic (Section 2.1), we can form a hypothetical judgment. It has the form of

$$\frac{J_1 \dots J_n}{J}$$

where the sequence $J_1 \dots J_n$ is a list of atomic propositions. An atomic proposition is a proposition P which cannot be broken down into further propositions. Furthermore, J is the succedent judgment or the goal which one has to prove from the sequence of judgments $J_1 \dots J_n$ [28] The process to deduce the conclusive judgment J is called hypothetical deduction. [21]

This process defines a methodology to verify and prove conclusive judgments, but it does not provide the reader with a way how to analyse the meaning (semantic) of the logical connectives. The semantic relationship notified as $J_1 \dots J_n \models J$ is based on the interpretation of truth in relation to the premises and conclusion; and how the application of logical operators act upon these truths [12].

Model Theory defines how to apply logic to structures [31]. A model is a structure that defines the interpretation of logic reasoning. Propositional logic uses a model which contains only True or False. For simplicity reasons we usually use structures which contain T or F to represent True and False respectively as in (2.1).

$$M_{\{T,F\}} = \left\{ p \;\middle|\; \begin{array}{r} \exists x \in \{T,F\} \cdot \; x = T \; then \; p = True \\ x = F \; then \; p = False \end{array} \right\} \tag{2.1}$$

We can have a different structure to represent logic. For instance, we can take the set of natural numbers as our logical model. We specify that whenever the element is 0 then that is considered as True. On the hand, whenever the element is greater than 0 then that is considered as False.

$$M_{\mathbb{N}} = \left\{ p \;\middle|\; \begin{array}{r} \exists x \in \mathbb{N} \cdot \; x = 0 \; then \; p = True \\ x > 0 \; then \; p = False \end{array} \right\} \tag{2.2}$$

We can have a different structure to represent logic. For instance we can take the set of natural numbers as our logical model. We specify that whenever the element is 0 then that is considered as True, and whenever the element is greater than 0 then that is considered as False.

Model (2.2) points out one of the advantage of using propositional logic. If we had to use model (2.2) we have to handle infinite sets when reasoning about truth values. On the other hand, propositional logic has only two elements therefore one can easily reason about truth values using a finite set.

Each logical connective has a different interpretation for truth. For instance given two propositions, A and B, then for $A \wedge B$ to be true both A and B have to be true. Whilst for $A \vee B$ if at least one is true then $A \vee B$ is true. The definition of truth for a particular logical connective is usually listed in a truth table which is a table that lists the value of the logical connective for valuations or models of propositional logic (every possible value for the atomic propositions).

A	B	A∧B
T	T	T
T	F	F
F	T	F
F	F	F

A	B	A∨B
T	T	T
T	F	T
F	T	T
F	F	F

A	B	A→B
T	T	T
T	F	F
F	T	T
F	F	T

Figure 2.1: Truth Tables for the logical connectives ∧,∨ and →

Example 2.2.1. RTP: (A∧B)⊨(B∧A)
In this example we are required to check whether the expression (A∧B)⊨(B∧A) is semantically correct. Thus, we work out the truth table (table 2.2) according to the information given in figure 2.1. If the sequence of truths and falses for both sub expressions is equal (like this case) then we conclude that the expression is semantically correct.

A	B	(A∧B)	(B∧A)
0	0	0	0
0	1	0	0
1	0	0	0
1	1	1	1

Table 2.2: Truth table for the expression (A∧B)⊨(B∧A)

Unlike syntax, the semantics of propositional logic shows how a proposition is not a consequence of another. That is, we can show that the judgment J cannot be proved for a specific element from the list of judgments $J_1 \dots J_n$ whenever the truth table shows that for a specific element J is false. Generally, in order to show that there does not exist a proof for a specific hypothetical judgment it is sufficient to provide a model of $J_1 \dots J_n$ which is not a model of J. [12]

2.3 Proof Theory

Computing a truth table for every time that one wants to reason about propositions is not always feasible. For a given formula composed from n atomic propositions, the corresponding

truth table consists of 2^n valuations or models of propositional logic. Being an exponential formula, it is feasible for a small number of n [18]. Another way to reason about propositions is using Proof Theory.

Proof Theory is when given a set of axioms and rules that hold for that particular system, one is capable to derive (by the application of axioms and inference rules) whether certain facts hold or not [25].

Axioms: Axioms or basic truths are facts that are always considered to be true and can be introduced in the proof search without the need to quote any propositions as premises. An example of an axiom is the introduction of truth. Since truth is always true then there is no need to prove other propositions to introduce truth.

Inference Rules: In order to deduce conclusions from known information one must have some rules which allow him/her to do so. Inference rules are rules that give true conclusive judgments from a collection of premises which are assumed (or proved) to be true.

$$\frac{\begin{array}{c} Premise_1 \\ Premise_2 \\ \vdots \\ Premise_n \end{array}}{Conclusion}$$

In trying to characterise the inference rules for intuitionistic logic, logical connectives like *and* and *or* are represented in symbolic logic. Using this methodology a set of rules is presented which correspond to the semantics of such logical terms and finally a formal representation of succedent judgments is given. The definitions of both the axioms and inference rules are bounded to the calculus being considered.

Proof theory is usually associated with syntax as compared to model theory which is usually associated with semantics. This interpretation to logic is purely mechanical because deductions are performed according to a set of inference rules or basic truths. That being said, it does not mean that proof theory is not as meaningful as model theory but in proof theory importance is given to the proof construction and the meaning of the proposition is irrelevant. In fact for a particular proposition in Proof Theory we might have infinite number of equivalent statements in model theory. For instance the statements referred to in Section 1.1 were *It is raining and he has an umbrella.* and *John is a gardener and a carpenter.* has different semantic meaning while in proof theory these can be represented in syntax as (A∧B).

Example 2.3.1. RTP: (A∧B)⊢(B∧A)

Where as in example 2.2.1 we had to compute the truth table for the expression (A∧B)⊨(B∧A) in proof theory we produce a proof. We recall from the introduction section the rules for the conjunction operator.

$$\frac{A \wedge B}{A} \wedge_{E1} \qquad \frac{A \wedge B}{B} \wedge_{E2} \qquad \frac{A \quad B}{A \wedge B} \wedge_I$$

The proof search for the expression (A∧B)⊢(B∧A) first opts to obtain the propositions A and B separately by the application of the $\wedge_E$ rules on the given proposition *(A∧B)*. It then

applies the rule $\wedge_I$ using the propositions A and B which were proved before.

$$\dfrac{\dfrac{\dfrac{}{(A \wedge B)}\,given\ proposition}{(B)}\wedge_{E2} \qquad \dfrac{\dfrac{}{(A \wedge B)}\,given\ proposition}{(A)}\wedge_{E1}}{(B \wedge A)}\wedge_I$$

In comparison to truth tables, the proofs computed in proof theory tend to be more concise. Using proof theory rather than model theory we produce a proof without the need to perform exhaustive or case-by-case analysis of the propositions. Futhermore, we are constantly reasoning about finite and small domain of propositions unlike model theory where one has to consider all the models possible to reach a solution.

3. Formal Logic

The term “formal logic" refers to the process of using a rigorous method to deduce whether arguments expressed symbolically are true or false. There exists several methodologies how this can be achieved whereby all the methodologies mentioned in this document have the same expressivity but each of which has a different level of non determinism in the process of deducing succedents from antecedents. This document focuses on four different proving techniques which are: Natural Deduction, Intercalation Calculus, Sequent Calculus and Proof Terms.

3.1 Natural Deduction

Proposed by Gerhard Gentzen[9], Natural Deduction is a simple proof system for both Classical and Intuitionistic Logic. Gentzen's point of departure was to find the connection between logical connectives and their mathematical reasoning. He defined introduction and elimination rules for each logic connective, these are then applied to propositions to deduce conclusive judgments. Gentzen even pointed out that these rules define the operators. In fact, he argued that the introduction rules alone are sufficient for this purpose and that the elimination rules are “no more, in the final analysis, than consequences of these definitions”.[3] Natural Deduction is a simplified proof system since the definition of each inference rule has no other references to other connectives. This is usually referred to as the orthogonality of the logical connectives [19]. This modular approach helps the reader to fully understand the logical flow in a proof without referring to other operators or connectives.

3.1.1 Inference rules for Natural Deduction

In order to show all the hypotheses visible at each derivation step it is better to localise the hypothetical judgments

$$\begin{array}{c} \overline{J_1\ true}^{k_1} \ldots \overline{J_n\ true}^{k_n} \\ \vdots \\ J\ true \end{array}$$

in the form of

$$k_1 : J_1 \ldots k_n : J_n \vdash J$$

With reference to the notation, the variable k_n in $k_n : J_n\ true$ means that “k_n is a proof of $J_n\ true$". The sequence of judgments $k_1 : J_1 \ldots k_n : J_n$ (usually refer to as antecedents) is generally minimised into Γ or . (in case the left hand side of the turnstile is empty) and it is usually referred to as the context of a proof. All the variables within the sequence $k_1 \ldots k_n$ must be distinct so that any ambiguity is eliminated. Additionally, any of the judgments $J_1 \ldots J_n$ can be duplicated as it is possible to have the same conclusion from the same premise but in

different scopes.

The inference rules for the Natural Deduction Calculus are:

Conjunction: If both A and B are proven to be true then $A \wedge B$ is true. This is concluded in the conjunction introduction rule:

$$\frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \wedge B} \wedge_I$$

On the other side, knowing that A *and* B is true then one can deduce that both A and B are proved to be true by the conjunction elimination rule.

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A} \wedge_{E_L} \quad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B} \wedge_{E_R}$$

Implication: To conclude that $A \supset B$ is true we have to assume that A is true and deduce that B is true from the assumption.

$$\frac{\Gamma, u : A \ \vdash B}{\Gamma \vdash A \supset B} \supset^{I^u}$$

The implication introduction rule includes a scope which defines the boundaries where the hypotheses are true. Therefore, taking into consideration the following example (the notation . in line 4 represents an empty antecedent list)

Example 3.1.1. RTP: . $\vdash (P \wedge Q) \supset Q$

1. $\overline{u : P \wedge Q \vdash u : P \wedge Q}^{\textit{given as an antecedents}}$
2. $\overline{u : P \wedge Q \vdash P}^{\wedge_{E_L}}$
3. $\overline{u : P \wedge Q \vdash Q}^{\wedge_{E_R}}$
4. $\overline{. \ \vdash (P \wedge Q) \supset Q}^{\supset^{I^u}}$

In example 3.1.1 proposition P (line 2) is only valid within the scope u and hence it is intuitionistically correct to conclude $u : P \wedge Q \vdash P$ by the application of rule $\wedge_{E_L}$. For both examples 3.1.1 and 3.1.2 the scope u is closed by the rule $\supset^{I^u}$ in line 4. Therefore, it is wrong to conclude P in line 5 for example 3.1.2.

Example 3.1.2.

1. $\overline{u : P \wedge Q \vdash u : P \wedge Q}^{\textit{given as an antecedents}}$
2. $\overline{u : P \wedge Q \vdash P}^{\wedge_{E_L}}$
3. $\overline{u : P \wedge Q \vdash Q}^{\wedge_{E_R}}$
4. $\overline{. \ \vdash (P \wedge Q) \supset Q}^{\supset^{I^u}}$
5. $\overline{. \ \vdash P}^{?}$

The implication elimination rule states that if $A \supset B$ *is true* and A is also proven to be true

then it follows that B *is true.*

$$\frac{\Gamma \vdash A \supset B \qquad \Gamma \vdash A}{\Gamma \vdash B} \supset E$$

Truth: Truth is always true and therefore no premises are required to introduce Truth. Consequently, no information can be extracted from a Truth premise therefore there is no elimination rule for Truth.

$$\frac{}{\Gamma \vdash \top} \top I$$

Falsehood: Falsehood can never be true and one should not derive it hence there is no introduction rule for falsehood. However, there can be cases when one derives a contradiction (like the case $A \wedge \neg A$). In that case from falsehood one can derive anything.

$$\frac{\Gamma \vdash \bot}{\Gamma \vdash C} \bot E$$

Although there exist introduction and elimination rules for the negation connective, in this document the negation connective is represented as $\neg A \equiv A \supset \bot$ so all the implication rules will apply for this notation.

The last thing to discuss about Natural Deduction regards the *hypothesis* rule. Natural Deduction rules deduce propositions which are on the right hand side (RHS) of the turnstile symbol. However, sometimes during proof search we require to apply inference rules on propositions which are on the left hand side (LHS) of the turnstile symbol. To solve this problem we should have a rule which transfers "left" propositions to "right" propositions. Moreover, since propositions on the LHS are usually hypotheses, which are implied at the beginning of the proof search, then it follows that these can be proven without any other information required.

$$\frac{}{\Gamma, A \vdash A} hyp$$

3.1.2 Top Down and Bottom Up Techniques for Proof Search

Before we give examples of how to prove judgments we should first give a brief description of the different techniques to use in the process to compute the proof.

Example 3.1.3. RTP: C,D $\vdash$ $(C \wedge D)$

$$\frac{\dfrac{}{C,\ D \vdash D} Hypothesis \qquad \dfrac{}{C,\ D \vdash C} Hypothesis}{C,\ D \vdash C \wedge B} \wedge_I$$

Proof searches using the top down technique start proving from the top (using the given antecedents) and proceeds to prove the succedent. Therefore, for example 3.1.3 the system starts proving the judgments $C, D \vdash D$ and $C, D \vdash C$ separately and justifies each proof with the *Hypothesis* rule. It then applies the rule $\wedge_I$ to obtain the judgment $C, D \vdash C \wedge D$.

On the other hand, when using the bottom up technique the proof search starts from the succedent and proceeds to the top. Therefore, for example 3.1.3 we start proving from the judgment $C, D \vdash C \wedge D$. In order to obtain an *and* operator between propositions C and D then we need to prove propositions C and D separately. Thus, judgments $C, D \vdash D$ and $C, D \vdash C$ are justified by the *Hypothesis* rule.

$$\frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \wedge B} \wedge_I$$

We apply these notations for inference rules. For instance, if we define that we are reading the rule $\wedge_I$ top down then the premises are $\Gamma \vdash A$ and $\Gamma \vdash B$ while the conclusion is $\Gamma \vdash A \wedge B$. However, if we declare that we are reading bottom up therefore we are proving from the succedent proceeding to the top then the premise is $\Gamma \vdash A \wedge B$ while the succedents are $\Gamma \vdash A$ and $\Gamma \vdash B$.

One point worth mentioning at this point regards the proof notation. Throughout this document we apply inference rules that require more than one premise. Usually, documents present proof searches using the same format as example 3.1.3 however, in this document we might explain examples that have long judgments thus it would not be feasible to present the sub proofs for the premises next to each other. For this reason we opt to present the sub proofs for the premises in a linear format and indentation guides the reader to establish the sub proofs. Hence we reformat example 3.1.3 as example 3.1.4.

Example 3.1.4. RTP: C,D $\vdash$ $(C \wedge D)$

$$\Big|\ \frac{}{C,\ D \vdash C}\,Hypothesis$$

$$\Big|\ \frac{}{C,\ D \vdash D}\,Hypothesis$$
$$\frac{}{C,\ D \vdash C \wedge D}\,\wedge_I$$

3.1.3 Examples

Example 3.1.5. RTP: $D, (A \wedge B) \wedge C \vdash A$

It seems that when one starts proving the first judgment there are no inference rules to apply because the right judgment is already an atomic proposition (it cannot be decomposed any further), and inference rules for the Natural Deduction are applied to the succedent of a judgment (in this case the succedent is A). Hence, the best solution is to change to top down techniques in proof search. We apply the Hypothesis rule to the list of antecedents and obtain two judgments since we have two antecedents:

$$\frac{}{D, (A \wedge B) \wedge C \vdash D}\,hyp \tag{3.1}$$

$$\frac{}{D, (A \wedge B) \wedge C \vdash (A \wedge B) \wedge C}\,hyp \tag{3.2}$$

Since the Natural Deduction Calculus does not guide us which judgment should we choose we start from judgment (3.1). However decomposition for this judgment is not possible as the succedent D is already an atomic proposition which leads the proof search into a stuck state.

Whenever the proof search is in a stuck state it can either terminate or backtrack and consider other possibilities. Thus we backtrack and consider judgment (3.2).

$$\frac{}{D, (A \wedge B) \wedge C \vdash (A \wedge B) \wedge C}\,hyp$$

Now we can apply the $\wedge_E$ rule. We have to choose which rule from the two ($\wedge_{E_1}$, $\wedge_{E_2}$) available to apply. In such cases it is better to consult the final goal of the proof. In this case it is A therefore it is useless to apply the $\wedge_{E_2}$ rule and deduce the proposition C because it will lead the proof search into a stuck state. Applying the $\wedge_{E_1}$ rule results in

$$\dfrac{\dfrac{}{D, (A \wedge B) \wedge C \vdash (A \wedge B) \wedge C}\,hyp}{\dfrac{D, (A \wedge B) \wedge C \vdash (A \wedge B)}{\vdots}\,\wedge_{E_1}}$$
$$D, (A \wedge B) \wedge C \vdash A$$

The next decomposition is another $\wedge_E$. Similarly to the previous step it is useless to apply $\wedge_{E_2}$ therefore one must apply $\wedge_{E_1}$ which terminates this proof.

$$\dfrac{\dfrac{\dfrac{}{D, (A \wedge B) \wedge C \vdash (A \wedge B) \wedge C}\,hyp}{D, (A \wedge B) \wedge C \vdash (A \wedge B)}\,\wedge_{E_1}}{D, (A \wedge B) \wedge C \vdash A}\,\wedge_{E_1}$$

Example 3.1.6. RTP: $(P \supset Q) \supset R \vdash Q \supset R$

The given hypothesis is $(P \supset Q) \supset R$ while the goal to prove is $Q \supset R$. Similar to example 3.1.5, although we can choose to start proving using top down techniques we opt to start from the bottom and proceed to the top. Since the succedent is not an atomic proposition we choose to apply the $\supset_I$ to compose the implication $Q \supset R$. This rule defines the scope u for the proposition Q, thus Q *is true* is only valid within the scope u.

$$\dfrac{\dfrac{\vdots}{(P \supset Q) \supset R, u : Q \vdash R}}{(P \supset Q) \supset R \vdash Q \supset R}\,\supset_{I^u}$$

At first glance it seems that the proof search has to stop at this point because there are no elimination rules that can be applied on $(P \supset Q) \supset R$ or Q to prove R. However by considering $(P \supset Q) \supset R$ one can make the intuition that in order to prove R using $\supset_E$ one has to prove the implication between P and Q. This step unveils one of the disadvantages of the Natural Deduction calculus. During decomposition we often have to make intuitions in order to continue proof tree computation, otherwise the proof search is stuck.

$$\Big|\overline{(P \supset Q) \supset R, u : Q \vdash (P \supset Q) \supset R}^{hyp}$$

$$\dfrac{\begin{array}{|l}\vdots\\ \overline{(P \supset Q) \supset R, u : Q \vdash P \supset Q}\end{array}}{\dfrac{\overline{(P \supset Q) \supset R, u : Q \vdash R}^{\supset E}}{\overline{(P \supset Q) \supset R \vdash Q \supset R}^{\supset I^u}}}$$

The step to prove $P \supset Q$ is straight forward because since Q is an antecedent then $P \supset Q$ follows from the introduction rule for the implication connective.

$$\Big|\overline{(P \supset Q) \supset R, u : Q \vdash (P \supset Q) \supset R}^{hyp}$$

$$\dfrac{\dfrac{\begin{array}{|l}\overline{(P \supset Q) \supset R, u : Q, v : P \vdash Q}^{hyp}\\ \overline{(P \supset Q) \supset R, u : Q \vdash P \supset Q}^{\supset I^v}\end{array}}{\overline{(P \supset Q) \supset R, u : Q \vdash R}^{\supset E}}}{\overline{(P \supset Q) \supset R \vdash Q \supset R}^{\supset I^u}}$$

3.1.4 Problems in Natural Deduction

Due to the non deterministic nature of the Natural Deduction Calculus we can obtain different proof searches for the same judgment. The calculus does not define a specific methodology how to apply the inference rules during computation thus there are many paths that the proof tree computation can take in order to provide a valid proof. Care should be taken which path to choose for a correct proof otherwise computation leads to a stuck state or terminates with an invalid proof. In this section we point out the factors which causes non determinism in Natural Deduction. These are:

- Choice of the appropriate succedent: certain inference rules conclude more than one principal formula[1] from the same active formula such as the $\wedge_E$. Natural Deduction does not define which rule should be applied at a certain point during decomposition. It gives us a set of rules that help us extract information from proven propositions or given propositions (antecedents). There can be cases during proof search when the wrong information is extracted therefore we must backtrack the proof search to a point where we can apply different inference rules that can establish the conclusion. One way to determine which principal formula should be chosen, therefore which inference rule should be applied, is to study the final goal of the proof and choose accordingly. However this is not intuitionally trivial.

  To show an instance of this factor we recall the proof search computed in example 3.1.5. The proof search was carried out using a top down technique therefore we started from the intuition

[1] a proposition of the conclusion that is not in the premise [8]

$$\frac{}{D, (A \wedge B) \wedge C \vdash (A \wedge B) \wedge C}\,hyp$$

At this point we can apply the rules $\wedge_{E_1}$ or $\wedge_{E_2}$. If we had to apply the rule $\wedge_{E_2}$ the proof search is automatically in a stuck state because from the proposition C there is no way to obtain proposition A which is our goal.

$$\dfrac{\dfrac{}{D, (A \wedge B) \wedge C \vdash (A \wedge B) \wedge C}\,hyp}{D, (A \wedge B) \wedge C \vdash C}\,\wedge_{E_R}$$

$\vdots$ *stuck state*

On the other hand the application of the rule $\wedge_{E_L}$ on the same judgment terminates the proof search.

$$\dfrac{\dfrac{\dfrac{}{D, (A \wedge B) \wedge C \vdash (A \wedge B) \wedge C}\,hyp}{D, (A \wedge B) \wedge C \vdash (A \wedge B)}\,\wedge_{E_L}}{D, (A \wedge B) \wedge C \vdash A}\,\wedge_{E_L}$$

- Forming intuitions during proof tree computation: during proof computation one can reach a point that although the proof search is stuck applying backtracking would not help the proof search. Usually such cases require the intuition of first proving another proposition then this proposition leads the proof search to a termination point.

  We recall example 3.1.6. The proof search for the judgment $(P \supset Q) \supset R \vdash Q \supset R$ followed a bottom up technique in example 3.1.6. After decomposing the judgment using the rule $\supset_{I^u}$ the proof search was in a state where no rule can be applied without further computation.

$$\dfrac{\dfrac{\vdots}{(P \supset Q) \supset R, u : Q \vdash R}}{(P \supset Q) \supset R \vdash Q \supset R}\,\supset_{I^u}$$

  Thus during computation we have to make the *intuition* that we first need to prove the proposition $P \supset Q$ then by the application of the rule $\supset_{I^v}$ we can terminate the proof search.

$$\left|\frac{}{(P \supset Q) \supset R, u : Q \vdash (P \supset Q) \supset R}\,hyp\right.$$

$$\dfrac{\left|\dfrac{\dfrac{}{(P \supset Q) \supset R, u : Q, v : P \vdash Q}\,hyp}{(P \supset Q) \supset R, u : Q \vdash P \supset Q}\,\supset_{I^v}\right.}{\dfrac{(P \supset Q) \supset R, u : Q \vdash R}{(P \supset Q) \supset R \vdash Q \supset R}\,\supset_{I^u}}\,\supset_E$$

  The process to come up with a correct intuition is already a non trivial task for human beings that can reason outcomes let alone for automated systems. For an automated system to come up with this intuition it has to first check all the antecedents in correspondence

to all the inference rules and try to establish whether applying an inference to one of the antecedents proves the succedent. If not then the automated system has to go through all the antecedents to analyse the relation between antecedents in terms of propositions and try to form the intuition. This kind of non determinism includes a lot of back tracking for an automated system. For each check performed on elements within the antecedents' list, the automated system takes paths which might lead to stuck states therefore it has to backtrack.

Similarly, inference rules like $\bot_E$ require an intuition to choose the principal formula. Usually when the rule $\bot_E$ is applied the conclusion (reading top down) is not $\bot$. This is the case in the judgment $A \wedge B, B \supset \bot \vdash C$ where the proposition C is not within any of the propositions in the antecedents. At this point the system has to make the intuition to prove the judgment $A \wedge B, B \supset \bot \vdash \bot$, otherwise the proof search will end up in a stuck state.

$$\Big|\overline{A \wedge B, B \supset \bot \vdash B \to \bot}^{hyp}$$

$$\begin{array}{l}\Big|\ \overline{A \wedge B, B \supset \bot \vdash A \wedge B}^{hyp} \\ \ \ \overline{A \wedge B, B \supset \bot \vdash B}^{\wedge_{E_L}} \\ \hline A \wedge B, B \supset \bot \vdash \bot \end{array}^{\supset_E}$$
$$\overline{A \wedge B, B \supset \bot \vdash C}^{\bot_E}$$

- Choice of backtracking or applying an inference rule: When applying top-down techniques in proof search, at every decomposition level one has to choose whether to apply an inference rule to an active formula or else backtrack. Usually one aims to decompose by applying an inference rule but there may be cases when one has to backtrack because the proof search has diverted from the actual goal. This is clearly shown in the following proof search:

$$\overline{(A \wedge B) \wedge C \vdash (A \wedge B) \wedge C}^{hyp}$$
$$\overline{(A \wedge B) \wedge C \vdash (A \wedge B)}^{\wedge_{E_L}}$$
$$\vdots$$
$$\overline{(A \wedge B) \wedge C \vdash C}$$

Reading top down, both the rules $\wedge_{E_L}$ and $\wedge_{E_R}$ can be applied, however the application of both will not lead the proof to a valid proof $\therefore$ backtrack

It can also be the case that the current proof search is in a stuck state therefore one has to go back and apply other inference rules in order to complete the proof search.

3.2 Intercalation Calculus

As a restricted version of Natural Deduction, the Intercalation Calculus limits the proof search space by disallowing undesired moves and permitting only meaningful derivations. [2]

Intercalation Calculus defines the antecedents of a judgment as uses ($A \downarrow$) and the succedent as either a *verification* ($A \uparrow$) or a *use* ($A \downarrow$). A proposition A is a *verification* if there exists a normal deduction (evidence) which proves it.

On the other hand, a *use* means that the proposition can be used during proof search without being proven. *Uses* are extracted from the hypotheses until the conclusion is proved (using elimination rules) in a top-down technique.[21] A new judgment is introduced for *uses*, A↓ which means that the judgment A "may be used". For instance, if we had to prove A from $A \wedge B$ then we apply the rule $\wedge_E$ and say that A is a *use* because apart that it is the succedent of an elimination rule, it was derived from the proposition $A \wedge B$ which has already been proven (as it is our hypothesis).

3.2.1 Inference rules for Intercalation Calculus

Conjunction Rules:

$$\frac{\Gamma \vdash A \uparrow \quad \Gamma \vdash B \uparrow}{\Gamma \vdash A \wedge B \uparrow} \wedge_I$$

$$\frac{\Gamma \vdash A \wedge B \downarrow}{\Gamma \vdash A \downarrow} \wedge_{E_L} \qquad \frac{\Gamma \vdash A \wedge B \downarrow}{\Gamma \vdash B \downarrow} \wedge_{E_R}$$

Implication Rules:

$$\frac{\Gamma, u : A \downarrow \vdash B \uparrow}{\Gamma \vdash A \supset B \uparrow} \supset I^u$$

$$\frac{\Gamma \vdash A \supset B \downarrow \qquad \Gamma \vdash A \uparrow}{\Gamma \vdash B \downarrow} \supset E$$

Truth:

$$\frac{}{\Gamma \vdash \top \uparrow} \top I$$

Falsehood:

$$\frac{\Gamma \vdash \bot \downarrow}{\Gamma \vdash C \uparrow} \bot E$$

Hypothesis:

$$\frac{}{\Gamma, A \vdash A \downarrow} hyp$$

When the proof search is carried out using both the bottom up and top down techniques then

the proof search comes to a point when both derivations meet half way having the upper derivation as a *use* while the lower derivation as a *verification*.

$$\dfrac{\dfrac{}{(P \wedge Q) \vdash (P \wedge Q)\downarrow}^{hyp}}{\vdots}$$
$$\dfrac{(P \wedge Q) \vdash P \downarrow}{(P \wedge Q) \vdash P \uparrow}^{?}$$
$$\vdots$$
$$(P \wedge Q) \vdash (P \vee Q) \uparrow$$

For these situations the calculus requires an inference rule which allow an **atomic proposition** to be changed from a *use* to a *verification* (reading top down).

$$\dfrac{\Gamma \vdash A \downarrow}{\Gamma \vdash A \uparrow} \downarrow\uparrow$$

Now that we have introduced the inference rules for this calculus we analyse how the introduction of *uses* and *verifications* affects the proof search. This calculus defines elimination rules as a *use* and introduction rules as a *verification*. More important it restricts their application and states that elimination rules should only be applied in a top down technique while introduction rules should only be applied in a bottom up technique.

Moreover, the introduction rules introduced so far are all deterministic. Thus, to minimise the need to constantly change the proof search technique (from bottom up to top down and vice versa), the Intercalation Calculus suggests that the system starts proving using the bottom up technique until an atomic proposition is reached then it changes the direction to top down technique and prove the atomic proposition. Thus, change between techniques is performed **only once**. Using this methodology one only has to backtrack for the proof search computed top down. Since the introduction rules are deterministic and these are applied in a bottom up technique then backtracking while using a bottom up technique is totally eliminated.

3.2.2 Examples

In this section we compute examples 3.1.5 and 3.1.6 using the intercalation Calculus. Then we analyse the advantages of using the Intercalation Calculus over Natural Deduction and problems for the Intercalation Calculus.

Example 3.2.1. RTP: $D\downarrow, (A \wedge B) \wedge C \downarrow \vdash A \downarrow$

Intercalation Calculus states that we should start proving using a bottom up technique until we reach an atomic proposition. The succedent of our judgment is already an atomic proposition therefore we proceed to use a top down technique.

At this point we apply the *Hypothesis* rule. However, since we have two antecedents we should decide on which antecedent the rule should be applied. In example 3.1.5 we analysed that if we apply the rule on the antecedent $D\downarrow$ we reach a stuck state hence we have to backtrack. Therefore, we apply the *Hypothesis* rule on the antecedent $(A \wedge B) \wedge C \downarrow$.

$$\frac{}{D\downarrow, (A \wedge B) \wedge C \downarrow \vdash (A \wedge B) \wedge C \downarrow}^{hyp}$$

To reach the atomic proposition $A\uparrow$ the proof search takes the same path as example 3.1.5. Therefore we can apply the $\wedge_{E_1}$ rule otherwise we reach a stuck state.

$$\dfrac{\dfrac{\dfrac{}{(A \wedge B) \wedge C \downarrow \vdash (A \wedge B) \wedge C \downarrow}^{hyp}}{(A \wedge B) \wedge C \downarrow \vdash (A \wedge B) \downarrow}^{\wedge_{E_1}}}{\vdots}$$
$$(A \wedge B) \wedge C \downarrow \vdash A \uparrow$$

The next decomposition is another $\wedge_E$. Similarly to the previous step it is useless to apply $\wedge_{E_2}$ therefore one must apply $\wedge_{E_1}$.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{(A \wedge B) \wedge C \downarrow \vdash (A \wedge B) \wedge C \downarrow}^{hyp}}{(A \wedge B) \wedge C \downarrow \vdash (A \wedge B) \downarrow}^{\wedge_{E_1}}}{(A \wedge B) \wedge C \downarrow \vdash A \downarrow}^{\wedge_{E_1}}}{\vdots}$$
$$(A \wedge B) \wedge C \downarrow \vdash A \uparrow$$

Now we are in a situation where we have achieved the atomic proposition A but it is a *use* rather than a *verification*. Therefore we apply the rule $\downarrow\uparrow$ which changes a *use* into a *verification*. The final proof tree is:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{(A \wedge B) \wedge C \downarrow \vdash (A \wedge B) \wedge C \downarrow}^{hyp}}{(A \wedge B) \wedge C \downarrow \vdash (A \wedge B) \downarrow}^{\wedge_{E_1}}}{(A \wedge B) \wedge C \downarrow \vdash A \downarrow}^{\wedge_{E_1}}}{(A \wedge B) \wedge C \downarrow \vdash A \uparrow}^{\downarrow\uparrow}$$

Example 3.2.2. RTP: $(P \supset Q) \supset R \downarrow \vdash Q \supset R \uparrow$

As the Intercalation Calculus defines that proof search should start from the conclusion then we start proving proposition $Q \supset R \uparrow$ using bottom up techniques. The only rules to apply at this moment is $\supset_I$. This rule defines the scope u for the proposition Q, thus Q *is true* is only valid within the scope u.

$$\dfrac{\vdots}{(P \supset Q) \supset R \downarrow, u : Q \downarrow \vdash R \uparrow}$$
$$(P \supset Q) \supset R \downarrow \vdash Q \supset R \uparrow^{\supset_I u}$$

The proof search has reached an atomic proposition therefore we must change to top down technique. It is important that we look at what we want to prove to get the exact list of antecedents. Almost all the inference rules persist the list of antecedents without neither adding or removing elements. Therefore, we start proving from the judgment

$(P \supset Q) \supset R\downarrow, u : Q\downarrow \vdash (P \supset Q) \supset R\downarrow$
however it seems that we reached a stuck state. Same as example 3.1.6 we have to make the intuition that in order to prove $R\uparrow$ we have to prove the sub proof $(P \supset Q) \supset R\downarrow, u : Q\downarrow \vdash P \supset Q\uparrow$

$$\Big|\overline{(P \supset Q) \supset R\downarrow, u : Q\downarrow \vdash (P \supset Q) \supset R\downarrow}^{hyp}$$

$$\dfrac{\Big|\begin{array}{l}\vdots\\ \overline{(P \supset Q) \supset R\downarrow, u : Q\downarrow \vdash P \supset Q\uparrow}\end{array}}{\dfrac{\overline{(P \supset Q) \supset R\downarrow, u : Q\downarrow \vdash R\uparrow}^{\supset_E}}{\overline{(P \supset Q) \supset R\downarrow \vdash Q \supset R\uparrow}^{\supset_{I^u}}}}$$

To prove $P \supset Q\uparrow$ we start from the bottom since we need to prove a *verification* and apply the rule $\supset_E$. We reached an atomic proposition therefore we change to top down technique for which we prove proposition $Q\downarrow$ by *Hypothesis* rule. Once again we apply the rule $\downarrow\uparrow$ to change a *use* into a *verification*.

$$\Big|\overline{(P \supset Q) \supset R\downarrow, u : Q\downarrow \vdash (P \supset Q) \supset R\downarrow}^{hyp}$$

$$\dfrac{\dfrac{\Big|\begin{array}{l}\overline{(P \supset Q) \supset R\downarrow, u : Q\downarrow, v : P\downarrow \vdash Q\downarrow}^{hyp}\\ \overline{(P \supset Q) \supset R\downarrow, u : Q\downarrow, v : P\downarrow \vdash Q\uparrow}^{\downarrow\uparrow}\\ \overline{(P \supset Q) \supset R\downarrow, u : Q\downarrow \vdash P \supset Q\uparrow}^{\supset_{I^v}}\end{array}}{\overline{(P \supset Q) \supset R\downarrow, u : Q\downarrow \vdash R\downarrow}^{\supset_E}}}{\dfrac{\overline{(P \supset Q) \supset R\downarrow, u : Q\downarrow \vdash R\uparrow}^{\downarrow\uparrow}}{\overline{(P \supset Q) \supset R\downarrow \vdash Q \supset R\uparrow}^{\supset_{I^u}}}}$$

3.2.3 Intercalation Calculus' Advantages

The advantage of using the Intercalation Calculus over Natural Deduction is that the former minimises the number of inference rules applicable at a certain point; while proving bottom up only introduction rules are applicable and while proving using a top down technique only elimination rules are applied. Furthermore, unlike proof search using the Natural Deduction that constantly changes the direction if it is in a stuck state, proof searching in the Intercalation Calculus changes direction once. It starts proving bottom up and when it decomposed the succedent to an atomic proposition it changes to a top down technique.

3.2.4 Problems in Intercalation Calculus

Although the Intercalation Calculus minimises the number of inference rules one has to choose from for a decomposition and reduces the number of times the system has to change between top down and bottom up techniques, the calculus still has a lot of non deterministic factors. We recall the non deterministic factors for Natural Deduction and check where the Intercalation Calculus differs.

- Choice of the appropriate succedent: From the analysis carried out for Natural Deduction we concluded that this problem is caused by non deterministic rules like $\wedge_E$ which for the same premise they output different conclusions. Unfortunately, the problem persists in this calculus because even for this calculus the rule $\wedge_E$ concludes two different conclusions.

- Forming intuitions during proof tree computation: In Intercalation Calculus there can be cases when during proof tree computation the only way to proceed forward is to make some intuitions based on the information given by the antecedents. This is the case for both examples 3.1.6(computed in Natural Deduction) and 3.2.2(computed in Intercalation Calculus) where we had to make the intuition to prove $P \supset Q$ in order to prove R. Example 3.2.2 shows that this non deterministic factor persists in the Intercalation Calculus.

- Choice of backtracking or applying an inference rule: Although this non deterministic factor is not totally eliminated, using the Intercalation Calculus it is minimised. While a proof search in Natural Deduction usually alternates between top down and bottom up techniques several times to prove its goal, a proof search computed in the Intercalation Calculus starts proving using a bottom up technique until it reaches an atomic proposition then it changes to top down technique and tries to prove the atomic proposition derived by the bottom up procedure. Thus, a proof search is usually composed from two proofs which then are joined by the rule $\downarrow\uparrow$. The subproof at the bottom is computed using the bottom up technique where it applies only introduction rules. Since the introduction rules defined so far are all *deterministic* then the succedent follows by the rule application and we do not have to backtrack thus we do not have to choose between applying an inference rule or backtrack. We recall example 3.2.2 where each decomposition in the bottom proof is deterministic.

$$
\left.\begin{array}{l}
\left|\dfrac{}{(P \supset Q) \supset R\downarrow, u : Q\downarrow \vdash (P \supset Q) \supset R\downarrow}\,hyp\right. \\
\\
\left|\begin{array}{l}
\dfrac{}{(P \supset Q) \supset R\downarrow, u : Q\downarrow, v : P\downarrow \vdash Q\downarrow}\,hyp \\
\dfrac{(P \supset Q) \supset R\downarrow, u : Q\downarrow, v : P\downarrow \vdash Q\uparrow}{}\,\downarrow\uparrow \\
\dfrac{(P \supset Q) \supset R\downarrow, u : Q\downarrow \vdash P \supset Q\uparrow}{}\,\supset_{I^v}
\end{array}\right. \\
\dfrac{(P \supset Q) \supset R\downarrow, u : Q\downarrow \vdash R\downarrow}{}\,\supset_E
\end{array}\right\} \textit{Upper Proof}
$$

$$
\left.\begin{array}{l}
\dfrac{(P \supset Q) \supset R\downarrow, u : Q\downarrow \vdash R\uparrow}{}\,\downarrow\uparrow \\
\dfrac{(P \supset Q) \supset R \vdash Q \supset R\uparrow}{}\,\supset_{I^u}
\end{array}\right\} \textit{Bottom Proof}
$$

On the other hand, the second proof (upper proof) starts proving from the top proceeding to the goal. This proof has the problem that not all the inference rules are deterministic. For instance the *Hypothesis* rule is non deterministic therefore if the proof search takes the wrong path (chooses the wrong succedent) it has to backtrack. Thus this non deterministic factor applies to only one part of the proof in Intercalation Calculus unlike Natural Deduction where it is a problem for the whole proof.

3.3 Sequent Calculus

Both in Natural Deduction and Intercalation Calculus the introduction rules are defined in a bottom up technique while elimination rules are defined in a top down technique. Preferably, a proof system proceeds either top-down only or vice versa (that is bottom up only), because proving using different direction techniques could complicate the proof. Moreover, when these proofs are computed using machines, these techniques create non-deterministic situations which are very hard to solve. Figure 3.1 shows that Sequent Calculus transforms all the eliminations rules into left rules hence we obtain a method to proof propositions starting proving from the conclusion and proceeding to the premises.

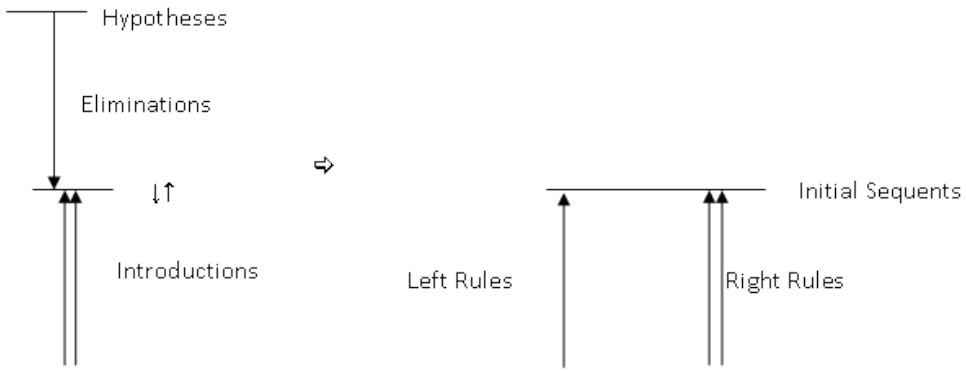


Figure 3.1: Sequent Calculus Transformation [19]

With the introduction of sequent calculus two different judgments are introduced; *A right* and *A left*. A sequent has the form of

$$A_1 \; left, A_2 \; left \ldots A_n \; left \vdash A \; right$$

The sequent is considered to be valid if all the left sequents are true and any of the right sequents are true. In other words, a sequent can be interpreted as

$$A_1 \; left \wedge A_2 \; left \ldots \wedge A_n \; left \vdash C_1 \; right \vee C_2 \; right \vee \ldots C_n \; right$$

3.3.1 Inference rules for Sequent Calculus

In Sequent Calculus, elimination rules are referred to as *left* rules while introduction rules are referred to as *right* rules. For notation purposes, the left and right annotations are usually omitted and left and right judgments are usually implied by their position.

Left Rules

When we analyse the left rules it is clearly evident that these are different from the elimination rules in the Intercalation Calculus. These rules act on the antecedents of the sequent while the succedent (which in the notation used is always notated as C) is never modified. At first glance, left rules may seem to have redundant antecedents because (reading bottom up) the antecedent

on which the rule is acting is usually carried on to the sequence of antecedents in the conclusion. Lets consider the rule $\wedge_{L_2}$:

$$\frac{\Gamma, A \wedge B, B \Rightarrow C}{\Gamma, A \wedge B \Rightarrow C} \wedge_{L_2}$$

Reading bottom up: the premise's antecedent $\Gamma, A \wedge B$ is not equal to the succedent's antecedent $\Gamma, A \wedge B, B$. The proposition B is carried on with the antecedents along the proof search.

The reason for this persistence of antecedents reflects the use of assumptions and hypotheses in Natural Deduction. In other words, since Sequent Calculus works in a bottom up technique if a hypothesis is not persisted then the benefit to conclude different conclusions from the same premise is lost.

Conjunction Rules:

$$\frac{\Gamma, A \wedge B, A \Rightarrow C}{\Gamma, A \wedge B \Rightarrow C} \wedge_{L_1} \quad \frac{\Gamma, A \wedge B, B \Rightarrow C}{\Gamma, A \wedge B \Rightarrow C} \wedge_{L_2}$$

Implication Rule:

$$\frac{\Gamma, A \supset B \Rightarrow A \quad \Gamma, A \supset B, B \Rightarrow C}{\Gamma, A \supset B \Rightarrow C} \supset L$$

Falsehood:

$$\frac{}{\Gamma, \bot \Rightarrow C} \bot L$$

Right Rules

On the other hand, the right inference rules for the Sequent Calculus are the same as the introduction rules for the Intercalation Calculus since the introduction rules for the latter calculus are already in a bottom-up technique.

Conjuction Rule:

$$\frac{\Gamma \Rightarrow A \quad \Gamma \Rightarrow B}{\Gamma \Rightarrow A \wedge B} \wedge_R$$

Implication Rule:

$$\frac{\Gamma, A \Rightarrow B}{\Gamma \Rightarrow A \supset B} \supset R$$

Truth:

$$\frac{}{\Gamma \Rightarrow \top} \top R$$

Init:

$$\frac{}{\Gamma, A \Rightarrow A} init$$

3.3.2 Examples

Example 3.3.1. RTP: $D, (A \wedge B) \wedge C \Rightarrow A$

Since Sequent Calculus uses only the bottom up technique the choice whether to start the proof search from the bottom and proving towards the top or else uses a top-down technique is eliminated. Usually when proving using the Sequent Calculus the right rules are given precedence over the left rules. Since the proposition on the right hand side of the sequent is an atomic proposition then one should apply a left inference rule. At this point we have to traverse the list of antecedents and check which antecedent can establish the succedent. For instance the first antecedent is proposition D however this does not lead us to proposition A, therefore we consider the next antecedent. The antecedent $(A \wedge B) \wedge C$ contains an A therefore we start decomposing this proposition. The next choice to make is whether to apply the $\wedge_{L_1}$ or $\wedge_{L_2}$ rule. Taking into consideration that the proposition to prove is A it is useless to apply the $\wedge_{L_2}$. It is better to opt for $\wedge_{L_1}$.

$$\dfrac{\dfrac{\vdots}{D, (A \wedge B) \wedge C, A \wedge B \Rightarrow A}}{D, (A \wedge B) \wedge C \Rightarrow A}\wedge_{L_1}$$

At this point the same argument is applied. The application of $\wedge_{L_1}$ is more meaningful rather than applying the other left rule. Once $\wedge_{L_1}$ is applied the proof search for this sub-proof is finished because there is a copy of the right sequent within the left sequence hence applying the *init* rule justifies the sequent.

$$\dfrac{\dfrac{\dfrac{}{D, (A \wedge B) \wedge C, A \wedge B, A \Rightarrow A}init}{D, (A \wedge B) \wedge C, A \wedge B \Rightarrow A}\wedge_{L_1}}{D, (A \wedge B) \wedge C \Rightarrow A}\wedge_{L_1}$$

Example 3.3.2. RTP: $(P \supset Q) \supset R \Rightarrow Q \supset R$

The first non-deterministic step is to choose whether to apply the left or right rule for the implication connective. As already done in the previous example of the Sequent Calculus the right rule is chosen over the left rule.

$$\dfrac{\dfrac{\vdots}{((P \supset Q) \supset R), Q \Rightarrow R}}{((P \supset Q) \supset R) \Rightarrow (Q \supset R)}\supset_R$$

The next proposition to consider is $((P \supset Q) \supset R)$ because it is the only one that can lead us to prove R. By applying the left implication rule this is decomposed into two sub-proofs. The second sub-proof is justified by the *init* rule.

$$\left|\begin{array}{l}\vdots\\ \overline{((P \supset Q) \supset R), Q \Rightarrow (P \supset Q)}\end{array}\right.$$

$$\left|\overline{((P \supset Q) \supset R), Q, R \Rightarrow R}^{init}\right.$$

$$\frac{\overline{((P \supset Q) \supset R), Q \Rightarrow R}^{\supset_L}}{\overline{((P \supset Q) \supset R) \Rightarrow (Q \supset R)}^{\supset_R}}$$

Similar to the first step carried out during this proof search, the sequent $((P \supset Q) \supset R), Q \Rightarrow (P \supset Q)$ has two inference rules that can be applied to it; $\supset_L$ and $\supset_R$. If $\supset_L$ is applied then one has to prove the sequents $((P \supset Q) \supset R), Q \Rightarrow (P \supset Q)$ and $((P \supset Q) \supset R), Q, R \Rightarrow (P \supset Q)$. Both sub-proofs can be proved, however, if during proof search the inference rule $\supset_R$ is never applied then the first subproof will end up in an infinite loop as one tends to apply the rule $\supset_L$ infinitely. The second subproof will end up in a stuck state. Thus, in cases when both the left rule ($\supset_L$) and the right rule ($\supset_R$) are both applicable then it is better to opt for the right rule.

$$\left|\begin{array}{l}\overline{((P \supset Q) \supset R), Q, P \Rightarrow Q}^{init}\\ \overline{((P \supset Q) \supset R), Q \Rightarrow (P \supset Q)}^{\supset_R}\end{array}\right.$$

$$\left|\overline{((P \supset Q) \supset R), Q, R \Rightarrow R}^{init}\right.$$

$$\frac{\overline{((P \supset Q) \supset R), Q \Rightarrow R}^{\supset_L}}{\overline{((P \supset Q) \supset R) \Rightarrow (Q \supset R)}^{\supset_R}}$$

3.3.3 Sequent Calculus' Advantages

One of the advantages of the Sequent Calculus is that by the definition of the inference rules it guides which path one shall choose during proof search which leads to the conclusion. In comparison to both the Natural Deduction and the Intercalation Calculus the Sequent Calculus reduces the number of choices that has to be performed during proof computation. In this section we compare certain non deterministic features for Natural Deduction and analyse how these are handled in the Sequent Calculus.

Forming intuitions during proof tree computation: In Natural Deduction we discussed that some times during proof search on has to make certain intuitions in order to continue. Proof searches in the Sequent Calculus are more intuitive as these follow the inference rules. The inference rules for this calculus are defined in such a way that they guide us to how the proof search should proceed. The problem of the examples computed in the Natural Deduction calculus and the Intercalation Calculus was that at certain points the proof search reaches a stuck state and in order to continue the proof one must make some intuitions. This problem is eliminated for the Sequent Calculus because by the application of the inference rules we are guided what

we should proof next. For instance the hardest problem in examples 3.1.6 (Natural Deduction) and 3.2.2 (Intercalation Calculus) was to make the intuition that in order to prove proposition R we should first prove $P \supset Q$. In example 3.3.2 which the proof search is computed in the Sequent Calculus, one of the premises to the rule $\supset_L$ is $((P \supset Q) \supset R), Q \Rightarrow (P \supset Q)$ therefore the system is guiding us which propositions we have to prove. At no point during proof search in the Sequent Calculus we had to make intuitions to complete the proof search.

Choice of backtracking or applying an inference rule: Before decomposing using the Natural Deduction we had to check which is the most feasible choice whether to backtrack or decompose. This is not the case in Sequent Calculus. Any proof proved in the Sequent Calculus uses the bottom up technique and if it is in a stuck state the proof search never backtracks but opts to apply an inference rule. Since antecedents and other propositions decomposed by the left rules are usually persisted, whenever the proof search is in a stuck state it does not backtrack to already proved propositions because these are persisted as antecedents. Therefore, in any situation the proof search applies inference rules. If it is the case that no inference rule applied on one of the antecedents is meaningful to the proof search then the proof search is terminated. This means that the choice to backtrack or apply an inference rule is totally eliminated in the Sequent Calculus.

3.3.4 Problems in Sequent Calculus

Although the Sequent Calculus solves any problems created by the inference rules' definition for the Natural Deduction, it still persists some others.

- The choice of which active part[2] to choose[8]: this choice is meaningful when the premise is a sequent (in Sequent Calculus) because when applying an inference rule to a sequent one has to decide to which proposition (from the list of antecedents on the left hand side of the sequent) the inference rule is going to be applied.

  For instance in example 3.3.1 we had the judgment $D, (A \wedge B) \wedge C \Rightarrow A$ which has two active parts D and $(A \wedge B) \wedge C$. It is insignificant to choose the proposition D and apply any left sequent inference rules because clearly the proof search will end up in a stuck state.

  However as seen in example 3.3.1 when the proposition $(A \wedge B) \wedge C$ is chosen the proof search terminates.

- Choice of the appropriate rule: The Natural Deduction calculus has some rules that for the same premise these give different conclusion. One of them is $\wedge_E$ where knowing that the proposition $A \wedge B$ is true then one can conclude that both A is true and B is true. Thus during proof tree computation one has to decide which rule to apply. This problem persists in the Sequent Calculus. Rules like $\wedge_L$ have different conclusions for the same premise which still one has to choose which one to apply.

[2] a proposition in the premise that is not in the conclusion [8]

3.4 Proof Terms

What if proofs can be interpreted as programs and hence one can extract a program from a proof?

A valid proof corresponds to a program which is semantically correct as well. Generally stated as the Curry-Howard isomorphism the correspondence between the Natural Deduction and simply-typed lambda calculus was originally noted by Howard [10]. This isomorphism provided the Intuitionistic Logic a language to reason about properties of programs.[19]

The correspondence between proofs and programs is usually illustrated by the notation *M:A* where M is a proof for A. In a localised hypothetical format this is expressed as M is a proof term for the proposition A under the hypotheses of A [19]. By the definition of Curry-Howard isomorphism this notation can be interpreted as M is a program of type A.
In brief the principles behind the correspondence between proofs and programs are:

- For every deduction of A true there is a proof term M and deduction of *M:A*
- For every deduction of *M:A* there is a deduction of A true
- The correspondence between proof terms M and deduction of A true is a bijection.[22]

3.4.1 Inference Rules for Proof Terms

Conjunction: A conjunction is made from two premises hence a proof term for a conjunction is a pair of proofs for the two premises. The elimination rule for conjunction is simply the extraction of the element from the pair of proofs.

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash N : B}{\Gamma \vdash \langle M, N \rangle : A \wedge B} \wedge_I$$

$$\frac{\Gamma \vdash M : A \wedge B}{\Gamma \vdash \pi_1 M : A} \wedge_{E_L} \quad \frac{\Gamma \vdash M : A \wedge B}{\Gamma \vdash \pi_2 M : B} \wedge_{E_R}$$

Implication: An implication can be interpreted as a function that maps a proof of A into a proof of B. This function is annotated as $\lambda x : A$ which is an abstraction in the lambda calculus. The bounded variable x in the lambda abstraction bounds the use of the hypothesis labeled u in both the proof term M and the implication succedent B. On the other hand the proof term in implication elimination is annotated as an application in the lambda calculus MN.

$$\frac{\Gamma, u : A \vdash M : B}{\Gamma \vdash \lambda u : A.M : A \supset B} \supset_I$$

$$\frac{\Gamma \vdash M : A \supset B \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B} \supset_E$$

Truth:

$$\frac{}{\Gamma \vdash \langle\rangle : \top} \top_I$$

Falsehood: Falsehood is annotated as abort. During computation this is not usually evaluated as a valid program will never attempt to evaluate a term of the form abort M.[22]

$$\frac{\Gamma \vdash M : \bot}{\Gamma \vdash \mathbf{abort}^C\ M : C}\bot_E$$

Hypothesis:

$$\frac{}{\Gamma_1, x : A, \Gamma_2 \vdash x : A}hyp$$

3.4.2 Reduction

When one reduces a proof term into another, one is simulating how programs are executed. A reduction has the form of $M \Rightarrow_R M'$ that is M reduces into M'. A full computation is actually a sequence of reductions until the judgment M' is in its normal form.

Analysed from a certain level of abstraction proof theory terms like introduction rules and elimination rules correspond to constructors and destructors (respectively) in programming languages.

Conjunction:

$$\pi_1 \langle M, N\rangle \Rightarrow_R M$$

$$\pi_2 \langle M, N\rangle \Rightarrow_R N$$

Implication:

$$(\lambda u : A.M)\, N \Rightarrow_R [N/u]\, M$$

There are no reduction rules for truth and falsehood.

A reduction rule is applied whenever a constructor introduces the type by an introduction rule and it is immediately eliminated by an elimination rule. Therefore, it is used to cancel out redundant decompositions [30]. We analyse this using examples 3.4.1 and 3.4.2.

Example 3.4.1. RTP: $a : A, b : B \vdash b : B$

$$\Big|\overline{a : A, b : B \vdash b : B}^{hyp}$$

$$\dfrac{\dfrac{\Big|\overline{a : A, b : B \vdash a : A}^{hyp}}{a : A, b : B \vdash < b, a >: B \wedge A}^{\wedge_I}}{a : A, b : B \vdash fst < b, a >: B}^{\wedge_{E_L}}$$

$$\overline{a : A, b : B \vdash b : B} \qquad \textit{reduced by } \pi_2 \langle M, N\rangle \Rightarrow_R N$$

Example 3.4.1 shows a derivation for the judgment *a:A, b:B* $\vdash$ *b:B*. The proof search first proves propositions *a:A* and *b:B* then applies the rule $\wedge_I$ to decompose <*b,a*>:*B*$\wedge$*A*. Using this rule it also assigns the type <*b,a*> to the proposition *B*$\wedge$*A*. To obtain proposition *B* it then

applies the rule $\wedge_{E_L}$ which assigns the type *(fst <b,a>)* to proposition *B*. At this point we can apply a reduction rule to reduce the type *(fst <b,a>)*. Thus we apply the reduction rule $\pi_2 \langle M, N \rangle \Rightarrow_R N$ to the pair *<b,a>* which reduces to type *b*.

The judgment *a:A, b:B* $\vdash$ *b:B* can be justified by the *Hypothesis* rule. The proof search for example 3.4.1 takes certain steps which are redundant to the proof tree therefore it applies the reduction rule to cancel out these steps.

Example 3.4.2. RTP: $a : A, b : B \vdash b : B$

$$\Big|\ \frac{}{a : A, b : B \vdash b : B}\,hyp$$
$$\frac{}{a : A, b : B \vdash \lambda a : A.b : A \supset B}\,\supset_I$$

$$\frac{\Big|\overline{a : A, b : B \vdash a : A}\,hyp}{a : A, b : B \vdash (\lambda a : A.b)\,(a) : B}\,\supset_E$$
$$\overline{a : A, b : B \vdash b : B} \quad \textit{reduced by } (\lambda u : A.M)\,N \Rightarrow_R [N/u]\,M$$

Example 3.4.2 shows a derivation for the judgment *a:A, b:B* $\vdash$ *b:B*. For this example the proof search opts to apply the $\supset_E$ rule therefore it starts proving the premises *a:A, b:B* $\vdash$ λ*a:A.b:A*$\supset$ *B* and *a:A, b:B* $\vdash$*a:A*. The type assigned to the proposition B after this is deduced using the rule $\supset_E$ is an application.

While reducing Lambda Calculus applications we have to be careful regarding free and bound variables. In the function to the left *(*λ*a:A.b)* the variable *a* is bounded while for the right function *(a)* the variable *a* is free. While applying the reduction we have to eliminate the possibility of a variable capture situation. To ensure that every variable retains its variable status (bound or free) we rename the bound variables for the left function therefore we rename the bound variable *a* to *y* - $(\lambda y : A.b)\,(a)$. Now we apply the reduction rule $(\lambda u : A.M)\,N \Rightarrow_R [N/u]\,M$ and reduce the Lambda Calculus application to *(b)*.

The judgment *a:A, b:B* $\vdash$ *b:B* can be justified by the *Hypothesis* rule. Both proof tree computation for examples 3.4.1 and 3.4.2 take certain steps which are redundant to the proof tree therefore we apply the reduction rule to cancel out these steps.

3.4.3 Example

Example 3.4.3. RTP: $x : (P \supset Q) \supset R \vdash \lambda u{:}\mathrm{Q}.(x)(\lambda v{:}\mathrm{P}.u){:}\mathrm{Q} \supset R$

For a proof search in Proof Terms we can prove it using the top down technique or the bottom up. Since the succedent of this judgment is an implication then we use a bottom up technique to apply the rule $\supset_I$. We should be careful when using this technique in Proof Terms because type computation depends on the premises and when proving bottom up one is proving what the premise is thus one has no access to the type unless the decomposition is justified by the *Hypothesis* rule. So whenever we use the bottom up technique we first compute the proof tree computation in terms of propositions, then we use a top down technique to compose the types.

Starting bottom up the first rule to apply is the $\supset_I$ thus;

$$\dfrac{\dfrac{\vdots}{x : (P \supset Q) \supset R, u : Q \vdash \_ : R}}{x : (P \supset Q) \supset R \vdash \_ : Q \supset R}\supset_{I^u}$$

Similar to the proof search in example 3.1.6 at this point we have to make the intuition to prove proposition $P \supset Q$ so that we can apply the rule $\supset_E$ and obtain proposition R. Thus, we prove the premises $x : (P \supset Q) \supset R, u : Q \vdash x : (P \supset Q) \supset R$ and $(P \supset Q) \supset R, u : Q \vdash \_ : P \supset Q$.

$$\begin{array}{ll}
1 & \left|\dfrac{}{x : (P \supset Q) \supset R, u : Q \vdash x : (P \supset Q) \supset R}hyp\right. \\
 & \\
2 & \left|\dfrac{}{x : (P \supset Q) \supset R, u : Q, v : P \vdash u : Q}hyp\right. \\
3 & \left|\dfrac{}{x : (P \supset Q) \supset R, u : Q \vdash _ : P \supset Q}\supset_{I^v}\right. \\
4 & \dfrac{}{x : (P \supset Q) \supset R, u : Q \vdash _ : R}\supset_E \\
5 & \dfrac{}{x : (P \supset Q) \supset R \vdash _ : Q \supset R}\supset_{I^u}
\end{array}$$

Now that the proof search has been computed we can start computing the types. We take a top down technique for this computation. Judgments justified by the *Hypothesis* rule (lines 1 and 2) are assigned the antecedent type. For the subproof $(P \supset Q) \supset R, u : Q \vdash \_ : P \supset Q$ we follow the rule $\supset_{I^v}$ and assign the type $(\lambda v : P.u)$. The judgment in line 5 is decomposed using the rule $\supset_E$ where it takes the sub proofs from line 1 and lines 2-3 as its premises. Therefore, the type assigned is the application of the types x and $(\lambda\ v{:}\mathrm{P}.u)$. Finally, since the last decomposition is justified by the rule $\supset_{I^u}$ the type for the proposition $Q \supset R$ is $\lambda u{:}\mathrm{Q}.(x)(\lambda\, v{:}\mathrm{P}.u)$.

$$\begin{array}{ll}
1 & \left|\dfrac{}{x : (P \supset Q) \supset R, u : Q \vdash x : (P \supset Q) \supset R}hyp\right. \\
 & \\
2 & \left|\dfrac{}{x : (P \supset Q) \supset R, u : Q, v : P \vdash u : Q}hyp\right. \\
3 & \left|\dfrac{}{x : (P \supset Q) \supset R, u : Q \vdash (\lambda v : P.u) : P \supset Q}\supset_{I^v}\right. \\
4 & \dfrac{}{x : (P \supset Q) \supset R, u : Q \vdash (x)\,(\lambda v : P.u) : R}\supset_E \\
5 & \dfrac{}{x : (P \supset Q) \supset R \vdash \lambda u : Q.\,(x)\,(\lambda v : P.u) : Q \supset R}\supset_{I^u}
\end{array}$$

Before termination, one has to check the proof term for any reductions and in this case there are none therefore we terminate proof search.

3.4.4 Analysis

For the previous calculi we analysed the non deterministic factors for that particular calculus. The calculus for Proof Terms is based on the Natural Deduction thus it has the same non deterministic factors. However, in this section we analyse the advantage of Proof Terms.

The inference rules for the Natural Deduction and Proof Terms are practically the same other than the fact that the rules for Proof Terms contain the type computation for each rule. This correspondence make it immediately clear that there exists a one-to-one correspondence between the Natural Deduction and the Proof Terms calculi. Thus from a proof in Natural Deduction one can construct the type derivation. Additionally, from the type derivation one can construct the proof search in the Natural Deduction [30].

Example 3.4.4. Correspondence between Natural Deduction and Proof Terms
In this example we recall the proof search for example 3.1.5 and using the correspondence between the Natural Deduction and Proof Terms we construct the type derivation in Proof Terms.

$$\begin{array}{l} \dfrac{\dfrac{}{D, (A \wedge B) \wedge C \vdash (A \wedge B) \wedge C}{}^{hyp}}{D, (A \wedge B) \wedge C \vdash (A \wedge B)}{}^{\wedge_{E_1}} \\ \dfrac{}{D, (A \wedge B) \wedge C \vdash A}{}^{\wedge_{E_1}} \end{array} \quad \longrightarrow \quad \begin{array}{l} \dfrac{\dfrac{}{b : D, a : (A \wedge B) \wedge C \vdash a : (A \wedge B) \wedge C}{}^{hyp}}{b : D, a : (A \wedge B) \wedge C \vdash \mathit{fst}\ a : (A \wedge B)}{}^{\wedge_{E_1}} \\ \dfrac{}{b : D, a : (A \wedge B) \wedge C \vdash \mathit{fst}\ (\mathit{fst}\ a) : A}{}^{\wedge_{E_1}} \end{array}$$

The left proof is computed in the Natural Deduction Calculus. The right proof is the corresponding proof search in Proof Terms. The terms defined for the right proof follow the inference rules for this calculus. Therefore, since the first decomposition (reading top down) is justified by the rule $\wedge_{E_1}$ the corresponding type is the first element of the proposition type on which the rule was applied, in this case a*:(* $A \wedge B$*)*$\wedge C$ thus *fst a:(*$A \wedge B$*)*. Similarly for the last decomposition (line 2 and 3).

Example 3.4.5. Correspondence between Proof Terms and Natural Deduction
Knowing the type for a particular judgment we can extract the proof search. In this example we consider the type definition for the judgment $b : D, a : (A \wedge B) \wedge C \vdash \mathit{fst}\ (\mathit{fst}\ a) : A$ and construct its equivalent proof search in Natural Deduction.

We start constructing the proof search in Natural Deduction from the bottom proceeding to the top. We start extracting information from the type *fst (fst a)*:A from the rightmost expression, that is **fst***(fst a)*. The only rule that requests the first element of a pair is $\wedge_{E_1}$. Thus, we know that the premise for the decomposition $\wedge_{E_1}$ should be a conjunction between the left proposition A and a right proposition. We consult the antecedent to conclude that the right proposition for the conjunction should be proposition B.

$$\frac{}{b : D, a : (A \wedge B) \wedge C \vdash \mathit{fst}\ (\mathit{fst}\ a) : A} \quad \longrightarrow \quad \frac{\dfrac{\vdots}{D, (A \wedge B) \wedge C \vdash (A \wedge B)}}{D, (A \wedge B) \wedge C \vdash A}{}^{\wedge_{E_1}}$$

Similarly, from judgment $(A \wedge B) \wedge C \vdash (A \wedge B)$ and the type *(fst a)* (it has been reduced by the previous decomposition) we know that the rule $\wedge_{E_1}$ was applied since the type requests the first element of a pair, and the premise for the decomposition is a conjunction between the left proposition $(A \wedge B)$ and a right proposition. Again we consult the antecedent and conclude that the premise is the judgment $(A \wedge B) \wedge C \vdash (A \wedge B) \wedge C$ which is justified by the *Hypothesis* rule.

$$\frac{}{b : D, a : (A \wedge B) \wedge C \vdash \mathit{fst}\ (\mathit{fst}\ a) : A}\wedge_{E_1} \quad \longrightarrow \quad \frac{\dfrac{\dfrac{}{D, (A \wedge B) \wedge C \vdash (A \wedge B) \wedge C}hyp}{D, (A \wedge B) \wedge C \vdash (A \wedge B)}\wedge_{E_1}}{D, (A \wedge B) \wedge C \vdash A}\wedge_{E_1}$$

To conclude, example 3.4.5 points out the advantage of using Proof Terms. The type of a judgment in Proof Terms represents the proof search for that particular judgment. Therefore as seen in example 3.4.5, due to the correspondence between the Natural Deduction and Proof Terms, having the type for a particular judgment one can compute the corresponding proof search in Natural Deduction. The advantage of Proof Terms is that rather than presenting the whole proof search which can sometimes be very long and cumbersome, one can present the type which still shows the methodology to verify that the judgment holds. Moreover, during computation using Proof Terms if the proof search computes any redundant steps as seen in examples 3.4.1 and 3.4.2 the reduction method in Proof Terms cancels such steps out. Thus, the final type is a concise type which defines a valid proof search for a particular judgment.

3.5 Conclusion

In this chapter we gave an extensive description of four calculi; Natural Deduction, Intercalation Calculus, Sequent Calculus and Proof Terms. During analysis for each calculus some points were pointed out. From this analysis we conclude that although the Natural Deduction is the most straight forward calculus from the four studied, proof search computation contains alot of non determinism. As our aim is to automate proof search non determinism is highly undesirable. We then studied the Intercalation Calculus that although it reduces non determinism for certain cases, the main features that cause non determinism are still evident.

We discussed that the most undesired feature in Natural Deduction and Intercalation Calculus is when during computation one has to make some intuitions so that from the known antecedents one can derive the conclusion. The analysis for the Sequent Calculus shows that it is easier to compute proof search in this calculus because by the definition of the inference rules proof search computation is guided which path it should follow. Thus during proof search in Sequent Calculus one does not need to make intuitions in order to continue. We concluded that once a proof search in Sequent Calculus is in a stuck state then it terminates while in Natural Deduction it has to backtrack and try other possible paths. Therefore, we conclude that *proof search computation in the Sequent Calculus performs better in comparison to the equivalent proof search in the Natural Deduction* and a feasible solution to solve automation problems in Natural Deduction is to compute the proof search in the Sequent Calculus and output its equivalent in Natural Deduction.

We analysed proof search and types in Proof Terms. We concluded that although proof search computation has the same disadvantages for Natural Deduction, the type computed for the conclusion is an encoding for the actual proof search. Thus, by the one to one correspondence between the Natural Deduction and Proof Terms, knowing the type for a particular judgment we actually know the proof search which verifies that the judgment holds.

4. Disjunction Logical Connective

Propositional logic is very powerful since a substantial amount of reasoning can be represented using either the *And* or the *Implication* logical connectives. This approach is very useful when designing circuits [27]. Implementing circuits using the same gate chips for the actual logical connectives reduces the expenses and the amount of resources redundancy. From a proof search point of view, implementing automated proof searches using the least possible amount of logical connectives reduces the system's complexity significantly. In classical logic the disjunction logical connective is usually substituted by conjunction expressions as defined by the De Morgan's Law which states that

$$(A \supset \bot) \wedge (B \supset \bot) = (A \vee B) \supset \bot \tag{4.1}$$

$$(A \supset \bot) \vee (B \supset \bot) = (A \wedge B) \supset \bot \tag{4.2}$$

However intuitionistic logic does not follow the full De Morgan's Law equality. Since intuitionistic logic does not support the law of excluded middle thus it prohibits the introduction of $P \vee (P \supset \bot)$ (equivalent to $P \vee (\neg P)$ as we represent $\neg C$ as $(C \supset \bot)$) during proof search then the judgment $(A \wedge B) \supset \bot \vdash (A \supset \bot) \vee (B \supset \bot)$ cannot be proved.

Example 4.0.1. RTP: $(A \wedge B) \supset \bot \vdash (A \supset \bot) \vee (B \supset \bot)$

Before we start the proof search we should first define the rules for disjunction introduction. We argue that if we know that proposition $R \vee S$ holds then it is sufficient to know that at least one of the propositions holds. Thus

$$\frac{\Gamma \vdash R}{\Gamma \vdash R \vee S} \vee_{I_L} \quad \frac{\Gamma \vdash S}{\Gamma \vdash R \vee S} \vee_{I_R}$$

Knowing these rules we start computing the proof tree. If we start from the top, by the *Hypothesis* rule we prove the proposition $(A \wedge B) \supset \bot$ but we are stuck since we do not have enough information to conclude $A \wedge B$ yet.

$$\frac{}{(A \wedge B) \supset \bot \vdash (A \wedge B) \supset \bot}\, Hypothesis$$

Therefore we change to bottom up and apply an $\vee_{I_L}$ rule.

$$\dfrac{\dfrac{}{(A \wedge B) \supset \bot \vdash (A \supset \bot) \vee (B \supset \bot)}\, Hypothesis}{\begin{array}{l}\vdots \\ (A \wedge B) \supset \bot \vdash (A \supset \bot)\end{array}}\, \vee_{I_L}$$

We then apply the rule $\supset_I$.

$$\frac{}{(A \wedge B) \supset \bot \vdash (A \supset \bot) \vee (B \supset \bot)}\,Hypothesis$$

stuck state

$$\frac{}{(A \wedge B) \supset \bot, A \vdash \bot}$$

$$\frac{(A \wedge B) \supset \bot, A \vdash \bot}{(A \wedge B) \supset \bot \vdash (A \supset \bot)}\,\supset_I$$

$$\frac{(A \wedge B) \supset \bot \vdash (A \supset \bot)}{(A \wedge B) \supset \bot \vdash (A \supset \bot) \vee (B \supset \bot)}\,\vee_{I_L}$$

At this point we reached a stuck state. We do not have enough information to conclude $A \wedge B$ so that by the rule $\supset_E$ on the proposition *(A∧B)*$\supset \bot$ we conclude $\bot$ which implies anything. Thus the judgment *(A∧B)*$\supset \bot \vdash$*(A*$\supset \bot$*)*$\vee$ *(B*$\supset \bot$*)* does not hold.

On the other hand, (4.1) holds hence we can apply it to change disjunction propositions into others composed from *And* and *Implication* logical connectives.

Although we can argue about non determinism using disjunction propositions in terms of *And* and *Implication*, we opt to introduce and study the inference rules for disjunction. This approach introduces us to another form of proof search non determinism as described later in this chapter.

4.1 Inference Rules for Disjunction

4.1.1 Natural Deduction

We start off by analysing the disjunction inference rules for Natural Deduction.The disjunction introduction rule may by far be the easiest inference rule because it is logical that in order to prove that $A \vee B$ *is true* it is sufficient to prove that at least either A *is true* or B *is true*.

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B}\,\vee_{I_L} \quad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}\,\vee_{I_R}$$

However, we cannot say the same for the disjunction elimination rule. Knowing that $A \vee B$ *is true* we cannot conclude for sure that A *is true* or B *is true* but we know for sure that one of the propositions *(A,B) is true*. Therefore, we can prove a conclusion C if we assume A to be true and then prove that C follows from A and also assume B to be true and prove that C follows from B. Since C has been proven to be true under both the assumptions of A and B then it follows that knowing that $A \vee B$ *is true*, C is also *true*.

$$\frac{\Gamma \vdash A \vee B \quad \Gamma, u : A \vdash C \quad \Gamma, v : B \vdash C}{\Gamma \vdash C}\,\vee E^{u,v}$$

Example

Example 4.1.1. RTP: $(A \vee B), A \supset C, B \supset C \vdash C$

Proof. We process the proof in a bottom up technique. We opt to apply the rule $\vee_{E^{u,v}}$ as the first decomposition. Thus we must proof three subproofs as defined by this rule.

$$(A \vee B),\ A \supset C,\ B \supset C\ \vdash (A \vee B)$$
$$(A \vee B),\ A \supset C,\ B \supset C,\ u : A\ \vdash C$$
$$(A \vee B),\ A \supset C,\ B \supset C,\ v : B\ \vdash C$$

If we opt to apply the $\vee_{E^{u,v}}$ rule for the subproof

$$(A \vee B), A \supset C, B \supset C, v : B \vdash C$$

we might conclude ineffective decompositions which lengthen the proof search. The best approach is to apply the $\supset_E$ rule. This decomposition requires another two subproofs, which are both justified by the hypothesis rule.

$$\dfrac{\vdots \quad \begin{array}{|l} \overline{(A \vee B), A \supset C, B \supset C, v : B \vdash B \supset C}^{hyp} \\ \\ \overline{(A \vee B), A \supset C, B \supset C, v : B \vdash B}^{hyp} \\ \hline (A \vee B), A \supset C, B \supset C, v : B \vdash C \end{array}^{\supset_E}}{(A \vee B), A \supset C, B \supset C \vdash C}^{\vee_{E^{u,v}}}$$

The subproof

$$(A \vee B), A \supset C, B \supset C, u : A \vdash C$$

is similar to the one explained above. Furthermore, the subproof

$$(A \vee B), A \supset C, B \supset C \vdash (A \vee B)$$

is justified by the hypothesis rule. Hence the final proof search is:

$$\dfrac{\begin{array}{|l} \overline{(A \vee B), A \supset C, B \supset C \vdash (A \vee B)}^{hyp} \end{array} \quad \begin{array}{|l} \overline{(A \vee B), A \supset C, B \supset C, u : A \vdash A \supset C}^{hyp} \\ \\ \overline{(A \vee B), A \supset C, B \supset C, u : A \vdash A}^{hyp} \\ \hline (A \vee B), A \supset C, B \supset C, u : A \vdash C \end{array}^{\supset_E} \quad \begin{array}{|l} \overline{(A \vee B), A \supset C, B \supset C, v : B \vdash B \supset C}^{hyp} \\ \\ \overline{(A \vee B), A \supset C, B \supset C, v : B \vdash B}^{hyp} \\ \hline (A \vee B), A \supset C, B \supset C, v : B \vdash C \end{array}^{\supset_E}}{(A \vee B), A \supset C, B \supset C \vdash C}^{\vee_{E^{u,v}}}$$

□

4.1.2 Analysis

In example 3.1.6 we discussed the non determinism for Natural Deduction. Particularly for that example we discussed that sometimes in order to achieve the final goal we need to make some intuitions during proof search. In this example we are looking at a similar problem. For example 3.1.6, we had the antecedent $(P \supset Q) \supset R$ but we could not conclude R therefore we decided to prove $(P \supset Q)$ then R follows by rule $\supset_E$. The non determinism in example 4.1.1 is the intuition for the proposition C. The intuition for example 3.1.6 was easier since we only had to look at one of the antecedents. The intuition in example 4.1.1 is more complex. Proposition C has nothing in relation with the disjunction proposition $A \vee B$ therefore during proof search we must extract enough information to form a valid intuition for C and assure that the application of the $\vee_{E^{u,v}}$ rule does not create any infinite loops. In example 4.1.1 we could have chosen to apply $\supset_E$ on either $A \supset C$ or $B \supset C$ however it is inevitable then not to apply a disjunction elimination in order to achieve C.

4.1.3 Intercalation Calculus

The definitions for the Intercalation Calculus inference rules for disjunction are similar to those for Natural Deduction. For the $\vee_I$ rules we add a verification to the succedent of both the premise and the conclusion as introduction rules are applied in top down technique hence knowing (we have a proof for) A then we can conclude (verify) $A \vee B$.

$$\frac{\Gamma \vdash A \uparrow}{\Gamma \vdash A \vee B \uparrow} \vee_{I_L} \qquad \frac{\Gamma \vdash B \uparrow}{\Gamma \vdash A \vee B \uparrow} \vee_{I_R}$$

On the other hand, the rule $\vee E^{u,v}$ is translated as

$$\frac{\Gamma \vdash A \vee B \downarrow \qquad \Gamma, u : A \downarrow \vdash C \uparrow \qquad \Gamma, v : B \downarrow \vdash C \uparrow}{\Gamma \vdash C \uparrow} \vee E^{u,v}$$

The premise $\Gamma \vdash A \vee B \downarrow$ is a *use* since it is usually proved in top down technique. Both the succedents for the other two premises are *verifications* since we have to actually prove the succedent for both premises to obtain the conclusion C.

The definition of verifications and uses for the rule $\vee_E$ guides us to how we should proceed our proof search. For instance, the premise $\Gamma \vdash A \vee B \downarrow$ is a *use* therefore we should start proof search from the top. The reason is that most of the time the proposition $A \vee B$ is an antecedent thus in most cases this premise is justified by the *Hypothesis* rule. On the other hand, for the other two premises we prove from the bottom proceeding to the top.

In comparison to Natural Deduction this definition guides us how the proof search should proceed, thus we do not have to alternate between top down and bottom up techniques but start proving using the bottom up technique and once the succedent is reduced to an atomic proposition we change to top down technique.

Example

Example 4.1.2. RTP: $(A \vee B)\downarrow, A \supset C\downarrow, B \supset C\downarrow \vdash C\uparrow$

Proof. The proof tree is composed in a bottom up technique. We apply the rule $\vee_{E^{u,v}}$ as our first decomposition. The application of this inference rule requires the proof of three sub proofs as its premises.

$$(A \vee B)\downarrow,\ A \supset C\downarrow,\ B \supset C\downarrow \vdash (A \vee B)\downarrow$$
$$(A \vee B)\downarrow,\ A \supset C\downarrow,\ B \supset C\downarrow,\ u : A\downarrow \vdash C\uparrow$$
$$(A \vee B)\downarrow,\ A \supset C\downarrow,\ B \supset C\downarrow,\ v : B\downarrow \vdash C\uparrow$$

The most appropriate rule to apply for the sub proof

$$(A \vee B)\downarrow,\ A \supset C\downarrow,\ B \supset C\downarrow,\ v : B\downarrow \vdash C\uparrow$$

is $\supset_E$ but the judgment is a *verification* while the rule is applied whenever the succedent is a *use*. Therefore, we must apply the rule $\downarrow\uparrow$ to change from a *use* to a *verification*.

$$\dfrac{\dfrac{\dfrac{\vdots}{(A \vee B)\downarrow,\ A \supset C\downarrow,\ B \supset C\downarrow,\ v : B\downarrow \vdash C\downarrow}}{(A \vee B)\downarrow,\ A \supset C\downarrow,\ B \supset C\downarrow,\ v : B\downarrow \vdash C\uparrow}\downarrow\uparrow}{(A \vee B)\downarrow,\ A \supset C\downarrow,\ B \supset C\downarrow \vdash C\uparrow}\vee_{E^{u,v}}$$

The rule $\supset_E$ requires the proof of another two sub proofs.

$$(A \vee B)\downarrow,\ A \supset C\downarrow,\ B \supset C\downarrow,\ v : B\downarrow \vdash B \supset C\downarrow \tag{4.3}$$
$$(A \vee B)\downarrow,\ A \supset C\downarrow,\ B \supset C\downarrow,\ v : B\downarrow \vdash B\uparrow \tag{4.4}$$

Similarly, to prove (4.4) we must first apply the rule $\downarrow\uparrow$ and then justify the judgment by the *Hypothesis* rule. The sub proof (4.3) is justified by the *Hypothesis* rule. Thus, the proof search for one of the sub proofs for the $\vee_{E^{u,v}}$ decomposition is:

$$\dfrac{\dfrac{\dfrac{\vdots}{\left|\begin{array}{l}\dfrac{}{(A \vee B)\downarrow,\ A \supset C\downarrow,\ B \supset C\downarrow,\ v : B\downarrow \vdash B \supset C\downarrow}hyp \\ \\ \dfrac{}{(A \vee B)\downarrow,\ A \supset C\downarrow,\ B \supset C\downarrow,\ v : B\downarrow \vdash B\downarrow}hyp \\ \dfrac{}{(A \vee B)\downarrow,\ A \supset C\downarrow,\ B \supset C\downarrow,\ v : B\downarrow \vdash B\uparrow}\downarrow\uparrow\end{array}\right.}}{(A \vee B)\downarrow,\ A \supset C\downarrow,\ B \supset C\downarrow,\ v : B\downarrow \vdash C\downarrow}\supset_E}{(A \vee B)\downarrow,\ A \supset C\downarrow,\ B \supset C\downarrow,\ v : B\downarrow \vdash C\uparrow}\downarrow\uparrow}{(A \vee B)\downarrow,\ A \supset C\downarrow,\ B \supset C\downarrow \vdash C\uparrow}\vee_{E^{u,v}} \tag{4.5}$$

Similarly, the proof for the subproof

$$(A \vee B)\downarrow,\ A \supset C\downarrow,\ B \supset C\downarrow,\ u : A\downarrow \vdash C\uparrow$$

follows the same approach as for (1.5). The sub proof

$$(A \vee B)\downarrow,\ A \supset C\downarrow,\ B \supset C\downarrow \vdash (A \vee B)\downarrow$$

is justified by the *Hypothesis* rule. Hence, the proof search for example 4.1.2 is:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{(A \vee B)\downarrow,\ A \supset C\downarrow,\ B \supset C\downarrow \vdash (A \vee B)\downarrow}hyp}{}}{}}{}}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{(A \vee B)\downarrow,\ A \supset C\downarrow,\ B \supset C\downarrow,\ u : A\downarrow \vdash A \supset C\downarrow}hyp}{}}{}}{}}{}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{(A \vee B)\downarrow,\ A \supset C\downarrow,\ B \supset C\downarrow,\ u : A\downarrow \vdash A\downarrow}hyp \quad \dfrac{}{(A \vee B)\downarrow,\ A \supset C\downarrow,\ B \supset C\downarrow,\ u : A\downarrow \vdash A\uparrow}\downarrow\uparrow}{(A \vee B)\downarrow,\ A \supset C\downarrow,\ B \supset C\downarrow,\ u : A\downarrow \vdash C\downarrow}\supset_E}{(A \vee B)\downarrow,\ A \supset C\downarrow,\ B \supset C\downarrow,\ u : A\downarrow \vdash C\uparrow}\downarrow\uparrow}{}}{}}{}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{(A \vee B)\downarrow,\ A \supset C\downarrow,\ B \supset C\downarrow,\ v : B\downarrow \vdash B \supset C\downarrow}hyp}{}}{}}{}}{}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{(A \vee B)\downarrow,\ A \supset C\downarrow,\ B \supset C\downarrow,\ v : B\downarrow \vdash B\downarrow}hyp \quad \dfrac{}{(A \vee B)\downarrow,\ A \supset C\downarrow,\ B \supset C\downarrow,\ v : B\downarrow \vdash B\uparrow}\downarrow\uparrow}{(A \vee B)\downarrow,\ A \supset C\downarrow,\ B \supset C\downarrow,\ v : B\downarrow \vdash C\downarrow}\supset_E}{(A \vee B)\downarrow,\ A \supset C\downarrow,\ B \supset C\downarrow,\ v : B\downarrow \vdash C\uparrow}\downarrow\uparrow}{}}{}}{}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{(A \vee B)\downarrow,\ A \supset C\downarrow,\ B \supset C\downarrow,\ u : A\downarrow \vdash A \supset C\downarrow}hyp \quad \dfrac{\dfrac{\dfrac{}{(A \vee B)\downarrow,\ A \supset C\downarrow,\ B \supset C\downarrow,\ u : A\downarrow \vdash A\downarrow}hyp \quad \dfrac{}{(A \vee B)\downarrow,\ A \supset C\downarrow,\ B \supset C\downarrow,\ u : A\downarrow \vdash A\uparrow}\downarrow\uparrow}{(A \vee B)\downarrow,\ A \supset C\downarrow,\ B \supset C\downarrow,\ u : A\downarrow \vdash C\downarrow}\supset_E}{(A \vee B)\downarrow,\ A \supset C\downarrow,\ B \supset C\downarrow,\ u : A\downarrow \vdash C\uparrow}\downarrow\uparrow \quad \dfrac{\dfrac{\dfrac{}{(A \vee B)\downarrow,\ A \supset C\downarrow,\ B \supset C\downarrow,\ v : B\downarrow \vdash B \supset C\downarrow}hyp \quad \dfrac{\dfrac{\dfrac{}{(A \vee B)\downarrow,\ A \supset C\downarrow,\ B \supset C\downarrow,\ v : B\downarrow \vdash B\downarrow}hyp \quad \dfrac{}{(A \vee B)\downarrow,\ A \supset C\downarrow,\ B \supset C\downarrow,\ v : B\downarrow \vdash B\uparrow}\downarrow\uparrow}{(A \vee B)\downarrow,\ A \supset C\downarrow,\ B \supset C\downarrow,\ v : B\downarrow \vdash C\downarrow}\supset_E}{(A \vee B)\downarrow,\ A \supset C\downarrow,\ B \supset C\downarrow,\ v : B\downarrow \vdash C\uparrow}\downarrow\uparrow}{(A \vee B)\downarrow,\ A \supset C\downarrow,\ B \supset C\downarrow \vdash C\uparrow}\vee_{E^{u,v}}$$

□

In comparison to Natural Deduction the Intercalation Calculus guides us which proof technique (top down or bottom up) we should be using and at what stage. Rather than keep alternating between proof techniques, we start the proof using the bottom up technique and once the succedent is reduced to an atomic proposition we change to top down technique. This is reflected in example 4.1.2 as first we reduce the conclusion to an atomic proposition then change direction starting from the top to proof the reduced proposition.

The problem in example 4.1.1 was to define the appropriate proposition C which is independent from the disjunction $A \vee B$. This problem persists for example 4.1.2.

4.1.4 Proof Terms

The inference rules for disjunction in Proof Terms are an extension of the respective rules in Natural Deduction. In this calculus, we have to consider the lambda term assigned after a decomposition. For the introduction rules ($\vee_{I_L}$, $\vee_R$) the term assigned is just an indication whether the proposition was injected on the left or right side of the proof. Taking into consideration the rule $\vee_{I_L}$ the term assigned to the proposition $A \vee B$ is $inl^B\ M$. This means that the

proposition B was injected to the proposition on the RHS of $\vee$, which in this case is proposition A.

$$\frac{\Gamma \vdash M : A}{\Gamma \vdash \mathbf{inl}^B M : A \vee B} \vee_{I_L} \quad \frac{\Gamma \vdash N : B}{\Gamma \vdash \mathbf{inr}^A N : A \vee B} \vee_{I_R}$$

The disjunction elimination rule in Natural Deduction guides us to proof a proposition from both the left and right propositions of a disjunction connective. In that way we know for sure that the proved proposition is true in either case. Hence, to obtain the correct proof term we should know which of the terms is actually true. This is computed by processing a case structure. The term **case** M **of inl** $u \supset N | \mathbf{inr}\ v \supset O$ in $\vee_{E^{u,v}}$ states that if the left proposition is true then the type assigned is the one corresponding to A, represented as N. Otherwise the term C is assigned type O.

$$\frac{\Gamma \vdash M : A \vee B \qquad \Gamma, u : A \vdash N : C \qquad \Gamma, v : B \vdash O : C}{\Gamma \vdash \mathbf{case}\ M\ \mathbf{of\ inl}\ u \supset N | \mathbf{inr}\ v \supset O : C} \vee E^{u,v}$$

Example

Example 4.1.3. RTP: $(A \vee B), A \supset C, B \supset C \vdash C$

Proof. While for the other calculi we processed the proof search using a bottom up technique in Proof Terms we are going to proceed in a top down technique since the conclusive term in a decomposition is dependant on the premises' terms. Taking into consideration the antecedents we opt to apply the $\vee_{E^{u,v}}$ rule, hence we must prove the required sub proofs.

$$c : (A \vee B), d : A \supset C, e : B \supset C \vdash c : (A \vee B)$$
$$c : (A \vee B), d : A \supset C, e : B \supset C, a : A \vdash \_ : C$$
$$c : (A \vee B), d : A \supset C, e : B \supset C, b : B \vdash \_ : C$$

The subproof

$$\text{c:}(A \vee B), d : A \supset C, e : B \supset C \vdash c : (A \vee B) \tag{4.6}$$

is justified by the *Hypothesis* rule. For the second subproof

$$c : (A \vee B), d : A \supset C, e : B \supset C, a : A \vdash \_ : C$$

we consider the antecedents $a : A$ and $d : A \supset C$. If we justify these two antecedents then we can apply the rule $\supset_{E^{a,d}}$ to justify the proposition C. Both propositions are justified by the *Hypothesis* rule as

$$\overline{c : (A \vee B),\ d : A \supset C,\ e : B \supset C,\ a : A\ \vdash d : A \supset C}^{hyp} \tag{4.7}$$

$$\overline{c : (A \vee B),\ d : A \supset C,\ e : B \supset C,\ a : A\ \vdash a : A}^{hyp} \tag{4.8}$$

Thus, by the rule $\supset_{E^{a,d}}$ using (4.7) and (4.8) we obtain

$$\Big|\overline{c:(A\vee B),d:A\supset C,e:B\supset C,a:A\vdash d:A\supset C}^{hyp}$$

$$\frac{\Big|\overline{c:(A\vee B),d:A\supset C,e:B\supset C,a:A\vdash a:A}^{hyp}}{c:(A\vee B),d:A\supset C,e:B\supset C,a:A\vdash (d(a)):C}^{\supset_E} \tag{4.9}$$

The term *(d(a))* for the proposition C is computed as defined by the rule $\supset_{E^{a,d}}$.
The proof for the third sub proof

$$\text{c:}(A\vee B),d:A\supset C,e:B\supset C,b:B\vdash \_:C \tag{4.10}$$

is similar to (4.9). Hence using sub proofs (4.6), (4.9), the proof for (4.10) and the rule $\vee_{E^{u,v}}$ we obtain the following proof search:

$$\frac{\Big|\overline{c:(A\vee B),d:A\supset C,e:B\supset C\vdash c:(A\vee B)}^{hyp}\quad \frac{\Big|\overline{c:(A\vee B),d:A\supset C,e:B\supset C,a:A\vdash d:A\supset C}^{hyp}\quad \frac{\Big|\overline{c:(A\vee B),d:A\supset C,e:B\supset C,a:A\vdash a:A}^{hyp}}{c:(A\vee B),d:A\supset C,e:B\supset C,a:A\vdash (d(a)):C}^{\supset_E}\quad \frac{\Big|\overline{c:(A\vee B),d:A\supset C,e:B\supset C,b:B\vdash e:B\supset C}^{hyp}\quad \frac{\Big|\overline{c:(A\vee B),d:A\supset C,e:B\supset C,b:B\vdash b:B}^{hyp}}{c:(A\vee B),d:A\supset C,e:B\supset C,b:B\vdash (e(b)):C}^{\supset_E}}{c:(A\vee B),d:A\supset C,e:B\supset C\vdash (case\ c\ inl\ a\ \supset\ (d(a))\ |inr\ b\ \supset\ (e(b))):C}^{\vee_E}$$

□

A proof search in Proof Terms is identical to a proof search in Natural Deduction with the addition of lambda terms for every decomposition. Hence example 4.1.4 has the same problems as mentioned for example 4.1.1.

4.1.5 Sequent Calculus

The inference rules for the disjunction in Sequent Calculus are very similar to those defined in the Natural Deduction. Since proof search in Sequent Calculus is conducted in bottom up technique only and we are constantly adding new propositions on the LHS of the sequent with every decomposition then we do not need the premise $\Gamma \Rightarrow A\vee B$ as it is already one of the

antecedents. Hence the rules are:

$$\frac{\Gamma \Rightarrow A}{\Gamma \Rightarrow A \vee B} \vee_{R_1} \quad \frac{\Gamma \Rightarrow B}{\Gamma \Rightarrow A \vee B} \vee_{R_2}$$

$$\frac{\Gamma, A \vee B, A \Rightarrow C \quad \Gamma, A \vee B, B \Rightarrow C}{\Gamma, A \vee B \Rightarrow C} \vee_L$$

Example

Example 4.1.4. RTP: $(A \vee B), A \supset C, B \supset C \Rightarrow C$

Proof. Proof search in the Sequent Calculus is carried out using bottom up technique. In this case the right proposition is an atomic proposition therefore we proceed by applying a left rule on one of the antecedents. The most appropriate rule to apply is $\vee_L$ for which we have to proof two sub proofs.

$$(A \vee B), A \supset C, B \supset C, A \Rightarrow C$$
$$(A \vee B), A \supset C, B \supset C, B \Rightarrow C$$

For the first sub proof (reading bottom up) we are required to prove

$$(A \vee B), A \supset C, B \supset C, B \Rightarrow C \tag{4.11}$$

Similar to the other proof searches in other calculi, we opt to apply an *implication* rule. To prove (4.11) we choose the $\supset_L$ rule. Moreover, to justify our judgment by the $\supset_L$ rule we need to prove

$$(A \vee B), A \supset C, B \supset C, B \Rightarrow B$$

$$(A \vee B), A \supset C, B \supset C, B, C \Rightarrow C$$

Both judgments are trivially proved by the application of the *init* rule. Hence we obtain the following sub proof:

$$\begin{array}{l} \left| \overline{(A \vee B), A \supset C, B \supset C, B \Rightarrow B}^{init} \right. \\ \\ \dfrac{\left| \overline{(A \vee B), A \supset C, B \supset C, B, C \Rightarrow C}^{init} \right.}{(A \vee B), A \supset C, B \supset C, B \Rightarrow C}^{\supset_L} \end{array} \tag{4.12}$$

The proof for the second subproof required by the $\vee_L$ rule is proved using the same methodology as proof (4.12). Hence the final proof search for this example is:

$$\dfrac{\dfrac{\Big|}{(A \vee B), A \supset C, B \supset C, A \Rightarrow A}^{init} \quad \dfrac{\dfrac{\Big|}{(A \vee B), A \supset C, B \supset C, A, C \Rightarrow C}^{init}}{(A \vee B), A \supset C, B \supset C, A \Rightarrow C}^{\supset_L} \quad \dfrac{\dfrac{\Big|}{(A \vee B), A \supset C, B \supset C, B \Rightarrow B}^{init} \quad \dfrac{\dfrac{\Big|}{(A \vee B), A \supset C, B \supset C, B, C \Rightarrow C}^{init}}{(A \vee B), A \supset C, B \supset C, B \Rightarrow C}^{\supset_L}}{(A \vee B), A \supset C, B \supset C \Rightarrow C}^{\vee_L}$$

□

4.2 Analysis

In the previous section we discussed, through examples, that non determinism during proof search is minimised substantially if we opt to compute the proof search using the Sequent Calculus rather than the Natural Deduction. When we analysed example 4.1.1, we said that non determinism created by the disjunction elimination rule ($\vee_E$ - ref. to $A.1$) results from the choice of proposition C which has no relation to the disjunction proposition A$\vee$B. The computation for example 4.1.5, which is computed in the Sequent Calculus, does not show that the non determinism created by the disjunction elimination rule for the Natural Deduction can be reduced by the disjunction left rule ($\vee_L$ - ref. to $A.3$) for the Sequent Calculus. The problem of choosing an appropriate proposition C is still visible in the Sequent Calculus disjunction left rule.

However, computing the proof search in Sequent Calculus still results in a proof with less non determinism even if the disjunction left rule is applied during proof search. Contrary to the disjunction elimination rule for the other calculi, the disjunction left rule requires only two premises. Hence, non determinism is reduced because any resultant non determinism from the sub proof $\Gamma \vdash A \vee B$ is totally eliminated as the Sequent Calculus' disjunction left rule does not require the mentioned subproof as a premise.

5. Proof Search Procedure

In chapter 3 we discussed that due to the nature of the inference rules for the Natural Deduction, proof search in this calculus contains a certain amount of non determinism. We discussed the Sequent Calculus and concluded that in comparison to the Natural Deduction, the Sequent Calculus' inference rules guide the user to take the correct path during proof search thus minimising the level of non determinism. In this section we discuss several methods how to reduce non determinism from proof search.

Additionally we discuss the methodology planned for the system. We set to present a system which has more than one methodology for proof computation using the Natural Deduction. In this system we distinguish between a verifier which outputs whether a judgment holds or not, and automated theorem provers which output the actual proof tree computation. For the automated theorem provers we focus on the Intercalation Calculus and the Sequent Calculus as these two calculi contain less non determinism in comparison to Natural Deduction. Since we are required to output the proof search in Natural Deduction we then focus on calculi translators for Natural Deduction to translate the actual proof search computed in either the Intercalation Calculus or the Sequent Calculus to Natural Deduction. Finally, we focus on translation between proof search in Natural Deduction to Proof Terms which is carried out by the one-to-one correspondence between the two calculi as described in Section 3.4.

5.1 Reducing Non Determinism in Proof Search

Each inference rule application consists of two possibilities; either a deterministic deduction or a non-deterministic deduction. The former is the most desirable because deterministic inference rules like $\wedge_I$ (ref $A.1$), define only one conclusion from a single or a set of premises therefore an immediate decomposition. On the other hand, inference rules like the $\wedge_E$ (ref $A.1$), create non-deterministic situations because for such rules the same precedent has more than one conclusion. In this section we are going to view different methodologies how we can minimise non determinism from Natural Deduction calculus.

5.1.1 The Cut-Elimination Rule

So far, the Sequent Calculus described has been lacking a way to introduce lemmata. The corresponding rule to introducing a lemma in Sequent Calculus is the Cut rule formally defined as

$$\frac{\Gamma \vdash A\ right \qquad \Gamma, A\ left \vdash C\ right}{\Gamma \vdash C\ right}\ cut$$

It states that in order to derive C from Γ, it is sufficient to prove that A is derivable from Γ and C is derivable from Γ and the additional hypothesis A.

This rule is highly undesirable in a logic system because although a derivation from Γ to A

justifies the use of A, the lemma A cannot be verified because it is subjective (a lemma which has to be chosen for that particular proof)[24]. Secondly, since A is subjective it creates a non-deterministic situation when such proofs are computed using a digital machine. We can solve this problem by using the Cut-Elimination theorem which states that every sequent of the form $\Gamma \Longrightarrow C$ which is derivable in the Sequent Calculus using the Cut rule is also derivable in the same calculus without Cut. Hence, the Cut-Elimination theorem proves that the rule of Cut is redundant in the Sequent Calculus.

The proof of Cut-Elimination is based on the admissibility of the Cut theorem which states:

Theorem 1. If $\Gamma \Longrightarrow$ A and Γ,A$\Longrightarrow$ C then $\Gamma \Longrightarrow$ C. [19]

Proof (see [19] page 11). The proof for Theorem 1 is a constructive proof that is, it shows how F is derived from the derivations of D and E in the following transformation

$$\Gamma \overset{D}{\Longrightarrow} A \text{ and } \Gamma, A \overset{E}{\Longrightarrow} C \text{ to } \Gamma \overset{F}{\Longrightarrow} C.$$

The proof proceeds by structural induction on the given derivation. Thus for a complete proof one must consider every logical connective derivation. Since the theorem has two premises then for every logical connective for the first premise one must consider every logical connective for the second premise. We consider the case where

$$D = \dfrac{\overset{D_1}{\Gamma, A_1 \Rightarrow A_2}}{\Gamma \Rightarrow A_1 \supset A_2}\supset_R \qquad E = \dfrac{\overset{E_1}{\Gamma, A_1 \supset A_2 \Rightarrow A_1} \quad \overset{E_2}{\Gamma, A_1 \supset A_2, A_2 \Rightarrow C}}{\Gamma, A_1 \supset A_2 \Rightarrow C}\supset_L$$

Proof:

1. $\Gamma \Rightarrow A_1 \supset A_2$ — *given by D*
2. $\Gamma, A_1 \supset A_2 \Rightarrow A_1$ — *given by E_1*
3. $\Gamma \Rightarrow A_1$ — *by IH on lines 1 and 2*
4. $\Gamma, A_1 \Rightarrow A_2$ — *given by D_1*
5. $\Gamma \Rightarrow A_2$ — *by IH on lines 3 and 4*
6. $\Gamma, A_1 \supset A_2, A_2 \Rightarrow C$ — *given by E_2*
7. $\Gamma, A_2 \Rightarrow C$ — *by IH on lines 1 and 6*
8. $\Gamma \Rightarrow C$ — *by IH on lines 5 and 7*

□

5.1.2 Analysing the relation between the Natural Deduction, the Intercalation Calculus and the Sequent Calculus

From the analysis carried out in Chapter 3 we concluded that the level of non determinism during proof search is minimised if the proof search is carried out in the Sequent Calculus rather than in Natural Deduction. We argued that the rules' definition for both the Intuitionistic and the Sequent Calculus restricts (guides) the user to follow certain paths and eliminates other during proof tree computation.

In this section we analyse the relation between calculi. We define theorems so that for every judgment in the Natural Deduction we can interpret the judgment into the Intercalation Calculus and vice versa (Theorems 2 and 3). Similarly, we define other theorems to interpret judgments in the Sequent Calculus from a proven judgment in the Intercalation Calculus (Theorem 4) and vice versa (Theorem 6). By the application of the modus ponens rule then we can obtain a relation between the Natural Deduction and the Sequent Calculus. Given a judgment in the Natural Deduction we can check whether it holds and compute its proof search by translating the judgment to the Intercalation Calculus then to the Sequent Calculus and compute the proof search in the Sequent Calculus. Once proof computation terminates we then translate the proof tree to the Intercalation Calculus then to the Natural Deduction. A visual explanation of this concept is shown in Fig. 5.1.

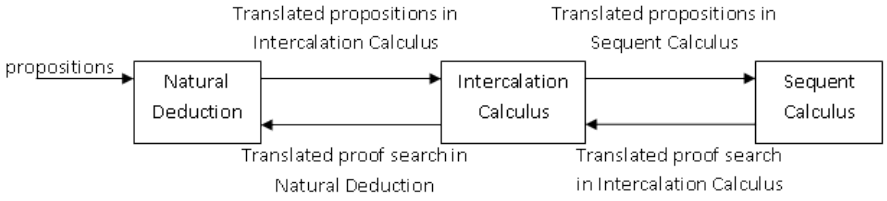


Figure 5.1: Proof Search Transition

The advantage of this methodology is that although the final proof is presented in the Natural Deduction it contains the same level of non determinism for proof searches computed using the Sequent Calculus.

Relation between Natural Deduction and Intercalation Calculus

The Intercalation Calculus is derived from Natural Deduction hence there exists a relation between the two. The rules for Intercalation Calculus are almost identical to those for Natural Deduction. The difference is that the Intercalation Calculus classify propositions as *verifications* and *uses*. To relate the two calculi we should find a methodology to introduce *verifications* and *uses* to a Natural Deduction judgment and other means to eliminate the *verifications* and *uses* from an Intercalation Calculus judgment thus we obtain an equivalent judgment. We now consider the theorems that define the relation between the two calculi.

Theorem 2. (Natural Deduction to Intercalation Calculus)
If $\Gamma \vdash A$ then $\Gamma \vdash A\uparrow$ and
If $\Gamma \vdash A$ then $\Gamma \vdash A\downarrow$. [19]

Proof (see [19] page 34). The proof is by structural induction on the given derivation. In other words, the complete proof should be computed from proofs proving the first and second clause of the theorem for every logical connective derivation. The proof is very straight forward. For every inference rule in the Natural Deduction calculus, we apply the inductive hypothesis on the premises to obtain the premises for the equivalent rule in the Intercalation Calculus. The

conclusion then follows by the application of the Intercalation Calculus rule. Here we consider the case for the rule $\supset_E$

Case:

$$\frac{\Gamma \vdash A \supset B \qquad \Gamma \vdash A}{\Gamma \vdash B} \supset_E$$

1.	$\Gamma \vdash A \uparrow$	*ih (1)* on $\Gamma \vdash A$
2.	$\Gamma \vdash A \supset B \downarrow$	*ih (2)* on $\Gamma \vdash A \supset B$
3.	$\Gamma \vdash B \downarrow$	$\supset_I$ for the Intercalation Calculus using lines 1 and 2

Theorem 3. (Intercalation Calculus to Natural Deduction)
If $\Gamma \vdash A \uparrow$ *then* $\Gamma \vdash A$ *and*
$If \Gamma \vdash A \downarrow$ *then* $\Gamma \vdash A$. [19]

Proof (see [19] page 32). Similar to the proof for Theorem 2, the proof for Theorem 3 is a proof by structural induction on the given derivation. Again the complete proof for completeness is computed by proving the theorem for every logical connective defined. Similarly to Theorem 2 we consider the case for $\supset_E$

Case:

$$\frac{\Gamma \vdash A \supset B \downarrow \qquad \Gamma \vdash A \uparrow}{\Gamma \vdash B \downarrow} \supset_E$$

1.	$\Gamma \vdash A$	*ih (1)* on $\Gamma \vdash A \uparrow$
2.	$\Gamma \vdash A \supset B$	*ih (2)* on $\Gamma \vdash A \supset B \downarrow$
3.	$\Gamma \vdash B$	$\supset_I$ for the Natural Deduction using lines 1 and 2

□

Relation between Intercalation Calculus and Sequent Calculus

By proving sound[1] and completeness[2] for the Sequent Calculus one is defining a relation between the Intercalation Calculus and the Sequent Calculus.
In order to prove that the Sequent Calculus is sound and complete it is sufficient to prove that there exists a relation between the Intercalation Calculus and the Sequent Calculus and vice-versa. Hence to prove soundness for this calculus we must prove:

Theorem 4. (Intercalation Calculus to Sequent Calculus)
1. If $\Gamma \vdash C \uparrow$ *then* $\widehat{\Gamma} \vdash C$ *right.*

[1]Soundness for the deductive system S with respect to the deductive system S' is the property that every true proposition in S is also true in S'.[18]

[2]Completeness for the deductive system S with respect to the deductive system S' is the property that every true proposition in S' is also true in S.[18]

2. $If \Gamma \vdash A\downarrow$ *and* $\widehat{\Gamma}, A\ left \vdash C\ right\ then\ \widehat{\Gamma} \vdash C\ right$.[24]
The first option reflects the one to one correspondence between the Intercalation Calculus and the Sequent Calculus where A↓ is equivalent to A *left* and A↑ corresponds to A *right*. For the second sub theorem, if we had to translate the premise $\Gamma \vdash A \downarrow$ according to the one to one correspondence between calculi we would get $\widehat{\Gamma} \vdash A$ *left* which is an invalid sequent. To resolve the differences we apply the cut rule which requests the premises $\Gamma \vdash A \downarrow$ and $\widehat{\Gamma}, A\ left \vdash C\ right$ to conclude $\widehat{\Gamma} \vdash C\ right$

Proof (see [19] page 40). The proof for soundness is a proof by structural induction on the given derivation which is computed by proving the theorem for every logical connective defined. Here we consider the case for the rule $\supset_E$
Case:

$$\frac{\Gamma \vdash A \supset B \downarrow \qquad \Gamma \vdash A \uparrow}{\Gamma \vdash B \downarrow} \supset_E$$

1.	$\widehat{\Gamma}, B \Rightarrow C$	Assumption
2.	$\widehat{\Gamma}, A \supset B, B \Rightarrow C$	Weakening: If $\Gamma \vdash P$ *then* $\Gamma, Q \vdash P$.
3.	$\widehat{\Gamma} \vdash A$	*ih (1)* on $\Gamma \vdash A \uparrow$
4.	$\widehat{\Gamma}, A \supset B \Rightarrow A$	Weakening: If $\Gamma \vdash P$ *then* $\Gamma, Q \vdash P$.
5.	$\widehat{\Gamma}, A \supset B \Rightarrow C$	$\supset_E$ using lines 2 and 4
6.	$\widehat{\Gamma} \Rightarrow C$	*ih (2)* using $\Gamma \vdash A \supset B \downarrow$ and line 5

Our aim throughout this proof is to prove the rule $\supset_L$. Therefore it is required to prove the sequents $\widehat{\Gamma}, A \supset B \Rightarrow A$, $\widehat{\Gamma}, A \supset B, B \Rightarrow C$ and $\widehat{\Gamma} \Rightarrow C$. We prove the first two sequents (lines 2 and 4) by the application of *Weakening*. This rule states that whenever a sequent holds for a specific set of antecedents then it should hold for a sequent with the same set of antecedents and maybe additional antecedents. The other required sequent (line 6) is proved using the inductive hypothesis on the given sequent $\Gamma \vdash A \supset B \downarrow$ and line 5.

□

Before proving completeness for the Sequent Calculus it is better to introduce the substitutions for Intercalation Calculus.

Theorem 5. (Substitution for Intercalation) Assume $\Gamma \vdash A \downarrow .Then$
1. $If\ \Gamma, A \downarrow \vdash B \downarrow\ then\ \Gamma \vdash B \downarrow, and$
2. $If\ \Gamma, A \downarrow \vdash C \uparrow\ then\ \Gamma \vdash C \uparrow$.[24]

Proof. By mutual induction on the structure of the deduction of $\Gamma, A \downarrow \vdash B \downarrow$ and $\Gamma, A \downarrow \vdash C \uparrow$[24] □

To show that the Sequent Calculus is complete with respect to the Intercalation Calculus then we have to prove the relation how to interpret proposition from the Sequent Calculus into Intercalation Calculus.

Theorem 6. (Sequent Calculus to Intercalation Calculus)
If $\Gamma \vdash C$ *right then* $\widehat{\Gamma} \vdash C \uparrow$.[24]

Proof. The proof is by structural induction on the given derivation.[24] Due to the facts that: this proof is one of the fundamental points of this system, contains a certain level of complexity to compute and we will refer to it in the design section then it is better to provide the full proof. The full proof listing can be found in the Appendices (ref. to B).

□

5.1.3 Choice of antecedent

Another non deterministic factor which we discussed in Chapter 3 was that none of the calculi define specifically which antecedent should be used and at what stage. This non deterministic factor is more related to the Sequent Calculus since it can decompose antecedents.

Rather than trying each possible antecedent in in an iterative order we opt to solve this problem by trying to extract certain information before we apply an inference rule. For instance if we are trying to determine whether the sequent $A, B, P \wedge Q \Rightarrow Q$ holds it is useless to try to apply an inference rule on propositions A and B because it does not lead us to Q. A sensible choice is to apply an inference rule on the antecedent $P \wedge Q$ because by further decompositions we can establish the succedent. Therefore, our solution for this problem is to traverse the list of antecedents and choose the antecedents which contain any common propositions with the succedent.

That being said there can still be cases for which trying to extract information from antecedents fails. Hence in such cases an automated system has to traverse the antecedents' list and try to extract information which may lead to the succedent.

5.1.4 Focusing

Focusing is a method to restrict the space of possible proofs in the Sequent Calculus (without the cut rule) without affecting provability [16]. In comparison to both the Natural Deduction calculus and the Intercalation Calculus, non determinism is reduced in proof search if the proof tree is computed in the Sequent Calculus because it does not require any intuitions during proof search but it follows inference rules application. This does not mean that proof search in the Sequent Calculus eliminates the possibility of non determinism. During proof search one still has to choose whether to apply a "left" rule or a "right" rule. To minimise substantially the non determinism created when choosing whether to apply a left rule or a right rule we use focusing. Focusing is based on the negative fragment of the intuitionistic logic [14].

Definition 1. The negative fragment contains all the Sequent Calculus inference rules that are invertible. [10]

Definition 2. An invertible rule is a deterministic rule where the conclusion implies the precedent. [6]

Therefore, a focus is the sequence of applying the invertible rules until the succedent has been normalised to an atomic formula.[9] For our calculus it happens to be that the negative fragment is composed from the “right” rules for the implication, conjunction and truth logical connectives. Hence to apply focusing during proof search we opt to first apply "right" rules until the succedent is an atomic proposition and then continue proving using the "left" rules for decomposition.

5.2 Methodology

Now that we have analysed different ways how to reduce non determinism from proof search we describe how we plan to implement an automated theorem prover.

One of the aims of the system is to compute a proof search in Natural Deduction to conclude whether a given judgment holds. In order to minimise the non determinism from proof search in Natural Deduction we plan to implement the methodology as illustrated in figure 5.1. Thus, knowing that proof computation in the Sequent Calculus has less unpredictable factors, given a judgment in the Natural Deduction, the system applies Theorem 2 to translate the judgment from Natural Deduction to the Intercalation Calculus. Having obtained a judgment in the Intercalation Calculus it applies the modus ponens rule on Theorem 4 to obtain a valid sequent.

At this point the system computes the proof search in Sequent Calculus. To further minimise any non determinism in Sequent Calculus it applies focusing and extracts information to choose a suitable antecedent for decomposition. If the judgment does not hold then the proof search is terminated and an empty tree is outputted to the user. Otherwise, for a valid proof search in the Sequent Calculus the system applies the modus ponens rule on Thereom 6 to obtain an equally expressive proof tree in Intercalation Calculus. This proof tree is then translated to Natural Deduction following Theorem 3.

Finally, the last option offered by this system is an ATP for proof Terms. Recalling from section 3.4.4 that Proof Terms have the same non deterministic level of Natural Deduction and there exists a one-to-one correspondence between the two calculi, rather than computing the proof search using Proof Terms the system computes a proof tree using the methodology mentioned in this section (where the proof search is carried out in the Sequent Calculus) to obtain a proof search in Natural Deduction. It then translates the proof search to Proof Terms. This methodology is illustrated in figure 1.3.

5.2.1 System’s Functionality

The system has two functionalities. It can verify judgments and it can prove judgments. If the system is requested to verify a judgment the user is given whether the judgment holds or not. Thus the system does not compute the full transition between calculi. By the transition described above we know that there exists a *one to one* correspondence between the Sequent Calculus and Natural Deduction. Therefore for verification it is sufficient to use the system described above and terminate once judgment is proved in the Sequent Calculus. The user is

notified whether proof search succeeds or not and no proof trees are outputted at this time hence it is redundant to translate the proof tree to the Intercalation Calculus then to Natural Deduction.

Another functionality of the planned system is as an automated theorem prover. For comparison reasons the system contains of two automated theorem provers. Both of them accept a judgment in Natural Deduction as input and output the proof search in Natural Deduction. The ATPs are:

- Natural Deduction (ND -> IC -> ND): the proof search is actually computed in the Intercalation Calculus thus it applies theorem 2 to translate the input judgment to Natural Deduction and theorem 3 to translate the proof search from Intercalation Calculus to Natural Deduction
- Natural Deduction (ND -> IC -> SC -> IC -> ND): The judgment passed by the user is translated to the Intercalation Calculus using theorem 2 then to the Sequent Calculus using theorem 4. The proof search is actually computed in the Sequent Calculus and translated back to Natural Deduction using theorems 6 and 3 respectively.

5.2.2 Proof Search Transition Methodology

In example 5.2.1 we apply a full transition in a step-by-step mode to illustrate how by the application of the suggestions above the level of non determinism is actual reduced to prove a judgment in Natural Deduction.

Example 5.2.1. In Example 3.1.5 we already discussed that sometimes during proof search in Natural Deduction one requires to make certain intuitions to prove the succedent goal. In that specific example we had to make the intuition to apply the *Hypothesis* rule on proposition $(A \wedge B) \wedge C$ in order to prove proposition A. The application of the proposed system eliminates the need to make intuitions during proof search.

1. Translate a Judgment from Natural Deduction to Intercalation Calculus

   We start by applying Theorem 2 so that we translate the Natural Deduction judgment into its equivalent in the Intercalation Calculus. This is the first translation required in order to translate the inputted judgment to its equivalent in the Sequent Calculus.

   $$(A \wedge B) \wedge C \vdash A \qquad \textit{translated to} \qquad (A \wedge B) \wedge C \vdash A \uparrow \tag{5.1}$$

2. Translate a Judgment from Intercalation Calculus to Sequent Calculus

   We apply Theorem 3 part 2 on the judgment obtained by the previous step
   (ie. $(A \wedge B) \wedge C \vdash A \uparrow$). After translation we obtain a valid sequent which we can process

to determine whether it holds or not.

$$(A \wedge B) \wedge C \vdash A \uparrow \qquad \textit{translated to} \qquad (A \wedge B) \wedge C \Rightarrow A \tag{5.2}$$

3. Compute Proof Search in Sequent Calculus

$$\begin{array}{l} \dfrac{}{(A \wedge B) \wedge C, A \wedge B, A \Rightarrow A}{}^{init} \\ \dfrac{(A \wedge B) \wedge C, A \wedge B \Rightarrow A}{}^{\wedge_{L_1}} \\ \dfrac{(A \wedge B) \wedge C \Rightarrow A}{}^{\wedge_{L_1}} \end{array} \tag{5.3}$$

4. Proof Search Translation from Sequent Calculus to Intercalation Calculus using Theorems 6 and 5:

   Proof search translation from one calculus into another follows from Theorem 6. The procedure is to translate each line in the proof search in a bottom up technique. Furthermore, every line in the proof search translation has to be justified using any of the Intercalation Calculus inference rules.

   For instance, the sequent showed below is translated directly using Theorem 6 and justified by the rule $\wedge_I$(ref to $A.2$):

$$\begin{array}{l} \left|\dfrac{}{A, B \Rightarrow A}{}^{init}\right. \\ \dfrac{\left|\dfrac{}{A, B \Rightarrow B}{}^{init}\right.}{A, B \Rightarrow A \wedge B}{}^{\wedge_R} \end{array} \quad \longrightarrow \quad \begin{array}{l} \left|\dfrac{}{A, B \vdash A}{}^{hyp}\right. \\ \dfrac{\left|\dfrac{}{A, B \vdash B}{}^{hyp}\right.}{A, B \vdash A \wedge B}{}^{\wedge_I} \end{array}$$

   Natural Deduction and Intercalation Calculus differ from the Sequent Calculus because in the latter the antecedents on which the rule is acting are usually carried on to the sequence of antecedents in the conclusion (reading bottom up). This means that the set of antecedents or the set of "left" propositions, Γ, is not a set with a fixed length. However, the inference rules for Intercalation Calculus are applicable only if the set Γ is a fixed set. In the next translation if one of the subproofs

$$\begin{array}{l} \dfrac{(A \wedge B) \wedge C, (A \wedge B) \Rightarrow A}{}^{\wedge_{L_1}} \\ \dfrac{(A \wedge B) \wedge C \Rightarrow A}{}^{\wedge_{L_1}} \end{array} \quad \longrightarrow \quad \begin{array}{l} \dfrac{(A \wedge B) \wedge C, (A \wedge B) \vdash A \uparrow}{} \\ \dfrac{(A \wedge B) \wedge C \vdash A \uparrow}{} \end{array}$$

   is translated using Theorem 6 then it cannot be justified by any of the Intercalation Calculus rules. The set of "left" propositions, Γ, differs from the sequent at the bottom to the one at the top ($[(A \wedge B) \wedge C] \neq [(A \wedge B) \wedge C, (A \wedge B)]$). To solve this problem we have to refer to Theorem 5 and apply the substitution rule. This rule requires the proof of two

other sub proofs

$$(A \wedge B) \wedge C \vdash (A \wedge B) \downarrow \tag{5.4}$$

$$(A \wedge B) \wedge C, (A \wedge B) \vdash A \uparrow \tag{5.5}$$

(5.4) must be computed directly in the Intercalation Calculus as it cannot be derived or translated directly from the proof computed in the Sequent Calculus. If we had to compute this subproof in the Sequent Calculus and translate it using Theorem 6 then we would have obtained the equivalent of $(A \wedge B) \wedge C \vdash (A \wedge B) \uparrow$ where the succedent is a *verification*. The substitution theorem specifies that the first premise should be a *use*($\downarrow$). Although we have a rule ($\downarrow\uparrow$) to translate from a *use* to a *verification*($\uparrow$), we have no rules to translate from a *verification* to a *use* thus, the first premise to the substitution theorem should be computed using the Intercalation Calculus. (5.5) is translated directly from the proof derived using the Sequent Calculus using Theorems 6 and 5 respectively.

$$\dfrac{\dfrac{}{(A \wedge B) \wedge C \vdash (A \wedge B) \wedge C \downarrow}{hyp}}{(A \wedge B) \wedge C \vdash (A \wedge B) \downarrow}{\wedge_{E_L}}$$

$$\dfrac{\dfrac{\dfrac{}{(A \wedge B) \wedge C, (A \wedge B) \vdash (A \wedge B) \downarrow}{hyp}}{(A \wedge B) \wedge C, (A \wedge B) \vdash A \downarrow}{\wedge_{E_L}} \quad \dfrac{\dfrac{\dfrac{}{(A \wedge B) \wedge C, (A \wedge B), A \vdash A \uparrow}{hyp}}{(A \wedge B) \wedge C, (A \wedge B), A \vdash A \uparrow}{\downarrow\uparrow}}{(A \wedge B) \wedge C, (A \wedge B) \vdash A \uparrow}{subs.rule}$$

$$\dfrac{}{(A \wedge B) \wedge C \vdash A \uparrow}{subs.rule}$$

In general, the substitution rule is used whenever the inference rule applied to the specific judgment is a left inference rule ($\wedge_{L_1}$, $\wedge_{L_2}$, $\rightarrow_L$ and $\vee_L$ - ref. to $A.3$) because these are the only rules that persist the set of antecedents and additional propositions.

5. Proof Search Translation from Intercalation Calculus to Natural Deduction using Theorem 3:

   The proof search translation from the Intercalation Calculus to the Natural Deduction is quite straight forward as it follows Theorem 3. Since the inference rules for the Intercalation Calculus are derived directly from the rules for Natural Deduction then we do not have to check whether the sets of "left" propositions are equal because in every case both sets correspond to each other.

   During translation, the final proof search in Natural Deduction might end up with several duplicate judgments whenever the $\downarrow\uparrow$ rule is used in the Intercalation Calculus proof search.

As discussed earlier, the Intercalation Calculus classifies the propositions as either a *use* or a *verification* which is not the case in Natural Deduction. This also justifies why the Natural Deduction does not have an inference rule which corresponds to the rule $\downarrow\uparrow$ in the Intercalation Calculus. In this case the best solution is to ignore the conclusion of the rule $\downarrow\uparrow$ whenever it is applied and just translate the precedent. In other words:

$$\dfrac{\dfrac{}{\Gamma, A \vdash A \downarrow}^{hyp}}{\Gamma, A \vdash A \uparrow}^{\downarrow\uparrow} \qquad \longrightarrow \qquad \dfrac{}{\Gamma, A \vdash A}^{hyp} \tag{5.6}$$

The translation between these two calculi follows Theorem 3:

$$\dfrac{\dfrac{}{(A \wedge B) \wedge C \vdash (A \wedge B) \wedge C}^{hyp}}{(A \wedge B) \wedge C \vdash (A \wedge B)}^{\wedge_{E_L}}$$

$$\dfrac{\dfrac{\dfrac{}{(A \wedge B) \wedge C, (A \wedge B) \vdash (A \wedge B)}^{hyp}}{(A \wedge B) \wedge C, (A \wedge B) \vdash A}^{\wedge_{E_L}}}{(A \wedge B) \wedge C, (A \wedge B) \vdash A}^{subs.rule}$$

$$\dfrac{\dfrac{\dfrac{}{(A \wedge B) \wedge C, (A \wedge B), A \vdash A}^{hyp}}{(A \wedge B) \wedge C, (A \wedge B) \vdash A}^{subs.rule}}{(A \wedge B) \wedge C \vdash A}^{subs.rule}$$

6. Proof Search Translation from Natural Deduction to Proof Terms using the one-to-one correspondence between both calculi:

   The final computation is to compute the type for the proof search we just translated. As described in Section 3.4.4, due to the correspondence between the calculi from a proof search in Natural Deduction we can obtain a proof term in Lambda Calculus. Furthermore, if we have a valid proof term for a judgment then we can construct the proof search in Natural Deduction for that particular judgment.

   Since type computation is dependant on the premise's type for every decomposition we should proceed using a top down technique. We start by assigning a representative type to every antecedent. Then, type computation follows the decompositions applied in the proof tree.

   The type computation for the judgment $(A \wedge B) \wedge C \vdash A$ is:

$$\Big|\ \frac{}{r{:}\,(A \wedge B) \wedge C \vdash r{:}\,(A \wedge B) \wedge C}^{hyp}$$
$$\frac{}{r{:}\,(A \wedge B) \wedge C \vdash \mathit{fst}\ (r){:}\,(A \wedge B)}^{\wedge_{E_L}}$$

$$\Big|\ \Big|\ \frac{}{r{:}\,(A \wedge B) \wedge C, s{:}\,(A \wedge B) \vdash s{:}\,(A \wedge B)}^{hyp}$$
$$\Big|\ \frac{}{r{:}\,(A \wedge B) \wedge C, s{:}\,(A \wedge B) \vdash \mathit{fst}\ (s){:}A}^{\wedge_{E_L}}$$

$$\Big|\ \frac{}{r{:}\,(A \wedge B) \wedge C, s{:}\,(A \wedge B)\,, t{:}A \vdash t{:}A}^{hyp}$$
$$\frac{}{r{:}\,(A \wedge B) \wedge C, s{:}\,(A \wedge B) \vdash t{:}A}^{subs.rule}$$

$$\frac{}{r{:}\,(A \wedge B) \wedge C \vdash t{:}A}^{subs.rule}$$

The first sub proof is justified by the rule $\wedge_{E_L}$ for Natural Deduction. The rule $\wedge_{E_L}$ for Proof Terms is defined as *fst (a)*

$$\frac{\Gamma \vdash M : A \wedge B}{\Gamma \vdash \pi_1 M : A}\wedge_{E_L}$$

Therefore the type after decomposition is

$$\frac{}{a{:}\,(A \wedge B) \wedge C \vdash a{:}\,(A \wedge B) \wedge C}^{hyp}$$
$$\frac{}{a{:}\,(A \wedge B) \wedge C \vdash \mathit{fst}\ (a){:}\,(A \wedge B)}^{\wedge_{E_L}}$$

The above subproof is a premise for the substitution theorem. The type assigned to the final succedent of this rule follows from the second premise. Therefore for the judgments

$$\Gamma \vdash p : P$$
$$\Gamma, p : P \vdash q : Q$$

the type assigned is q - $\Gamma \vdash q : Q$.

With reference to our example for the premises

$$r{:}(A \wedge B) \wedge C \vdash \mathit{fst}\ (r){:}\,(A \wedge B)$$
$$r{:}(A \wedge B) \wedge C, s{:}\,(A \wedge B) \vdash t{:}A$$

we take the type of the second premise therefore $r{:}\,(A \wedge B) \wedge C \vdash t{:}A$

5.3 Conclusion

To conclude we discuss the benefits of using the methodology described in this chapter. Proof computation for one of the automated theorem provers is carried out in the Sequent Calculus then translated to Natural Deduction. The advantage of this approach is that although the output for the final proof search is in Natural Deduction, the level of non determinism is reduced as the actual proof search is carried out in the Sequent Calculus.

On the other hand, for verifiers performing a partial translation is sufficient. Our aim is to verify whether the judgment holds thus if we have a proof search to confirm our claim we have a

verification. By partial transition we mean that we intend to handle verification by translating the Natural Deduction judgment into its equivalent in the Intercalation Calculus then to the Sequent Calculus. By the one to one correspondence between calculi described in section 5.1.2 we know that if a judgment holds in one calculus it holds in the other. Hence once we compute the proof search in the Sequent Calculus we have verified whether the judgment holds. The advantage of this system is that although the input judgment is in Natural Deduction, we are verifying it using a different calculus which has a better performance in comparison to Natural Deduction(it contains less non determinism). Furthermore, we are minimising the amount of computation needed if we had to output the verification in Natural Deduction.

6. System Implementation

From the analysis carried out in Chapter 3 we concluded that proof search computed in Natural Deduction contain high level of non determinism which is an undesired feature for automated systems. In Chapter 5, we concluded that a feasible automated methodology that reduces non determinism during proof search is to compute the proof search in the Sequent Calculus, translate it to the Intercalation Calculus then to the Natural Deduction.

In this section we intend to give an extensive description of how the system was implemented and what are the features that the system offer. This chapter includes decisions taken during implementation and reasons to justify such decision. It also includes system's achievements, limitations, intricate parts handled during implementation and known errors.

6.1 Language Selection

After considering several languages and paradigms we opted to use Haskell for the following reasons:

- Haskell is a functional language and these languages tend to carefully follow mathematical concepts. For this reason any programs written in Haskell are very easy to comprehend without the need of knowing specific syntax.
- Haskell has a call-by-need semantics, that is a function is only evaluated once no matter how many times the function is called in the program. Furthermore, if any function arguments are never called then the referred arguments are never evaluated. This type of evaluation allows us to argue about lists of infinite length because we can define every list element in terms of its successors.

6.2 Implementation Overview

The main purpose of this system is to develop an ATP for Natural Deduction. The system offers to two features; a verifier and an automated theorem prover. In case of the verifier, after that the user inputs the judgment in Natural Deduction the system translates the judgment into Intercalation Calculus then to Sequent Calculus. It computes the proof search in the Sequent Calculus and if this was successful then the system outputs that the judgment holds. Otherwise, it outputs that proof search failed therefore the judgment does not hold.

Furthermore, the system offers two different methodologies for proof computation in Natural Deduction and another methodology for Proof Terms. These are:

- Method 1: Natural Deduction - (ND $\rightarrow$ IC $\rightarrow$ ND): The system translates the input from Natural Deduction to Intercalation Calculus, computes the proof search in the Intercalation Calculus and then translates it into Natural Deduction.

- Method 2: Natural Deduction - (ND → IC → SC → IC → ND): The application translates the input from Natural Deduction to Intercalation Calculus and then to Sequent Calculus. It then computes the proof search in the Sequent Calculus, translates it to Intercalation Calculus and then to the Natural Deduction.
- Proof Terms - (ND → IC → SC →IC → ND → PT): For this option the system follows the same procedure as method 2 and translates the proof search from Natural Deduction to Proof Terms.

Other than the actual theorem provers for the Intercalation Calculus and the Sequent Calculus, the system implemented consists of several translators which translate the proof tree computed in any of the theorem provers to either the Intercalation Calculus, Natural Deduction or Proof Terms. Figure 6.1 shows the general module implementation and interaction of the presented system. The system implemented outputs the computed proof tree in two different

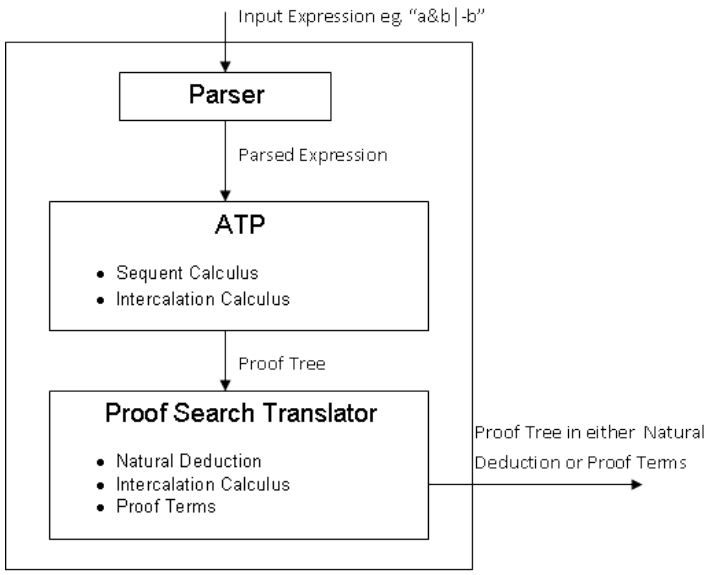


Figure 6.1: General Design: Module Interaction

ways. For every option it outputs the proof tree to the user through the console and it has the option to output the proof search to a file.

6.3 Data Structure

The specification for a proposition is:

```
data Proposition =   Var Char
                   | Truth
                   | Falsehood
                   | Conj Proposition Proposition
                   | Imply Proposition Proposition
```

```
| Disj Proposition Proposition
| Emp
```

For a valid proposition the data structure accepts the terminals; variable (eg. P), Truth, Falsehood and *Emp*. It also accepts the nonterminals *Conj a b* (a∧b), *Disj a b* (a∨b) and *Imply a b* (a→b) where both a and b are valid propositions.

One point worth mentioning at this point regards the terminal *Emp*. This proposition is assigned during computation only. There can be cases which are discussed later on in this document, when we have a list of propositions and we are constantly accessing elements starting from the middle of the list and proceeding circularly. Therefore we must have a terminal (in our case *Emp*) that indicates the position from where we start searching otherwise we end up in an infinite loop if the element we are searching for is not an element of the list.

The three infix operators (∧,∨,→) are considered to be right associative thus (A∧B∧C⊢C) is equivalent to (A∧(B∧C)⊢C). The recursive definition allows us to represent propositions as trees.

Example 6.3.1. For tree for the proposition (A∧(B∧C)) is

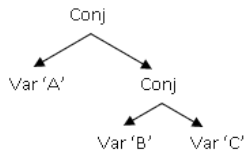


Figure 6.2: Proposition Tree for (A∧(B∧C))

6.4 Module Description

The modules in this system are classified into three areas; a parser, automated theorem provers and translators. In this section we explain the most intricate parts of each module.

6.4.1 Parser

The aim of the parser is to translate the input into a meaningful structure that the other modules can process. This module is divided into two section; the lexer and the parser itself.

The lexer's task is to convert the input stream into a stream of tokens. This is done by analysing the input stream and allocating tokens (ref to Table 6.1) for the appropriate characters.

It also filters out any space characters and detects errors in input. For instance, the system does not accept numbers as atomic propositions thus the expression "3&b|-b" outputs a lexical error. On the other hand, for a valid input like "a&b|-b" the lexer outputs the following list of tokens.

```
[VarName 'a', And, VarName 'b', Equates, VarName 'b']
```

Lexeme	Token	Lexeme	Token
(	OBracket	|	Or
)	CBracket	&	And
|-	Equates	!	F
,	Comma	.	T
"a"	VarName <Char>	->	Implies

Table 6.1: Lexeme and Token correspondence

There are several ways how to implement a parser. We could have implemented the parser using a parser generated. Since we are working with a Context Free Grammar (CFG) we could have easily translated our grammar for the parser generator. However we opted to implement our own parser for the following reasons:

- The coding produced automatically by the parser generator requires a certain technical level to comprehend its functionality. By writing our own parser we present elegant and maintainable coding.
- Parser generators tend to minimise the control on error handling and recovery. Writing our own parser gives us the full control of its functionality.

The parser was implemented using parser combinators. Actually when working with combinators one is implementing simple parsers to parse simple grammar expressions [5]. One should also implement the combinators to combine these individual parsers. The actual parser is just a combination of these straight forward parsers. Moreover, using this methodology the parser produced can be easily maintained by only changing either the combinators or the simple parsers themselves.

The parser implemented has three parser type definitions:

1. `type Parser a = [Token] -> Maybe (a, a)`
   This type of parser accepts a list of tokens and output a tuple of type a. It is used in the primitive stages when we want to parse the set of "left" propositions (antecedents) and the "right" proposition (succedent).

2. `type ParserList a = [Token] -> Maybe ([a])`
   This parser accepts a list of tokens and outputs a list of type a. Once the list of tokens has been parsed into the list of antecedents and the succedents, then the antecedent expression should be parsed into a list of lists of tokens, where every list of tokens represent a proposition.

3. `type ParserProp a = [Token] -> Maybe (a)`
   This is the general parser for all the logic connective parsers. For a parser of type Proposition it accepts a list of tokens and outputs a proposition.

Example 6.4.1. Parsing expression "a&b,c|-a"

After lexical analysis and tokenisation the list of tokens for the above expression is

```
[VarName 'a', And, VarName 'b',Comma, VarName 'c', Equates, VarName 'a']
```

The first parser type applied to the expression is Parser (the first one listed). After the list is parsed by parsers of this type the output is:

```
[VarName 'a', And, VarName 'b',Comma, VarName 'c'] --- antecedents
[VarName 'a'] --- succedent
```

The second parser type (ParserList) is then applied to the list of tokens which represent the antecedents.

```
[[VarName 'a', And, VarName 'b'], [VarName 'c']] --- antecedents
[VarName 'a'] --- succedent
```

The last parser type is ParserProp which includes every basic parser that parses expressions into structures of the type Proposition. For the example shown here, after the expression is parsed with a parser of this type the output is

```
[Conj (Var ;'a') (Var 'b'), Var 'c'] --- antecedents
[Var 'a'] --- succedent
```

Elementary Parsers

Each of the tokens has an elementary parser which parses a list of tokens into a proposition according to the particular token being processed. For instance one of the elementary parsers is `conj` which is defined as

```
1. conj :: ParserProp Proposition
2. conj = \x -> case (propAnd [] x) of
           3. (f,[]) -> Nothing
           4. (f,s) -> case (expr f) of
                       5. Nothing -> Nothing
                       6. Just prop -> case (expr s) of
                                       7. Nothing -> Nothing
                                       8. Just prop2 -> Just (Conj (prop) (prop2))
```

The elementary parser invokes the function *propAnd* which in turn if an *And* token exists it outputs a tuple of two list of tokens which represent the tokens on the LHS and tokens on the RHS of the *And* token. Thus for the input `[VarName 'a', And, VarName 'b']` the output expected is `[VarName 'a']` and `[VarName 'b']`. For this output, the parser then proceeds to parse the LHS list and the RHS list separately and output `Just (Conj (Var 'a') (Var 'b'))`.

Furthermore, we have to take into consideration the scope of the expression we are trying to parse. If the *And* token happens to be bound between bracket tokens like the case `[OBracket, VarName 'a', And, VarName 'b', CBracket, Implies, VarName 'c']` then when parsing for *And* the elementary parser outputs `Nothing`. Eventually, the expression within the brackets will be parsed but within the appropriate scope.

Parser Combinators

The system uses three parser combinators in total. It uses an *or* (denoted as `<|>`) parser combinator to combine two parses using the *or* operator. The other two are *and* parsers denoted as `<-+>` and `<+->`. The *and* parser denoted as `<-+>` is used when to combine the two parsers using the and operator but it returns the result of the second parser. On the other hand, the and parser `<+->` is used to combine two parsers using the and operator but it returns the result of the first parser. These two parsers are used in combinator to another parser to remove extra brackets from a proposition.

On the other hand, the *or*(`<|>`) combinator connects the elementary parsers to achieve a parser capable to parse a list of tokens into a proposition. The order in which the parsers are combined defines the precedence for the logical connectives as defined in Table 2.1. Our combined parser starts checking the logical connective with the least precedence so that logical connective with the highest precedence is grouped with the correct atomic proposition/s. For the combined parser

```
1.  expr :: ParserProp Proposition
2.  expr =  obracket <-+> expr <+-> cbracket
3.      <|> impl
4.      <|> disj
5.      <|> conj
6.      <|> var
7.      <|> truth
8.      <|> falsehood
```

For the input "a&b->c", the parser starts by checking whether there are any extra brackets (ref to line 2). It then checks whether the expression contains an implication by line 3. At this point the parser assigns the proposition `Impl` and calls the parser on the expressions “ab” and “c” to get the right proposition and left proposition respectively of the implication.

For the right proposition, the `expr` parser calls the var parser which returns the proposition `Var 'c'`.

For the left proposition, that is “a&b”, the parser starts checking for extra brackets, it checks for the implication and disjunction (lines 2 and 3) operators before it checks for conjunction (line 4). The parser assigns the proposition `Conj` and it calls recursively the parser `expr` on the expressions “a” and “b”. By the `var` parser the propositions are `Var 'a'` and `Var 'b'` returned. Therefore, the final parsed expression is `Impl (Conj (Var 'a') (Var 'b')) (Var 'c')`. The same execution flow can be seen in table 6.4.1. The table shows the list of tokens rather then the identifiers for propositions (for instance `VarName 'a'` is the token for `a`)

Token List	Elementay Parser Invoked	Proposition
[VarName 'a',And,VarName 'b', Implies,VarName 'c']	impl	Impl _ _
[VarName 'a',And,VarName 'b']	conj	Impl (Conj _ _) _
[VarName 'a']	var	Impl (Conj (Var 'a') _) _ _) _
[VarName 'b']	var	Impl (Conj (Var 'a') (Var 'b')) _
[VarName 'c']	var	Impl (Conj (Var 'a') (Var 'b')) (Var 'c')

Table 6.2: Step-by-Step Parsing

6.4.2 Automated Theorem Provers

Before we describe the implementation of the ATPs we better give an overview of how these work. The theorem provers are passed a list of antecedents and the final succedent. For each decomposition the system checks the succedent and starts traversing the list of antecedents.

The proposition *Emp* is added as the last element in the antecedents' list when this is passed by the main program. The system is constantly traversing and shifting the elements in the antecedents' list (including the *Emp* proposition) in order to search and extract information to apply a decomposition rule. The *Emp* proposition serves to show from which point traversing has started and indicates when list traversing should stop. Hence, it prevents the possibility of entering an infinite loop.

Example 6.4.2. RTP: $B \vee A, E, A \wedge B \vdash K$

To show the functionality of the proposition *Emp* in the system we prove the above judgment using the Emp proposition and another computation without the *Emp*. We start with the computation for the one using the *Emp* proposition.

Using the Emp Proposition
As already defined, when the system passes the data to this module the last proposition of the antecedents' list is *Emp*. Thus, we have the following expression. (NB this is just a representation of the actual parsed expression. For simplicity reasons we opt to present it using the easiest way possible)

$$B \vee A, E, A \wedge B, Emp \vdash K$$

The system starts traversing the list of antecedents. The first proposition is $B \vee A$ thus the

system try to apply the rule $\vee_E$. This rule requires the premises:

$$B \vee A, E, A \wedge B, Emp \vdash B \vee A \tag{6.1}$$

$$E, A \wedge B, B, Emp, B \vee A \vdash K \tag{6.2}$$

$$E, A \wedge B, A, Emp, B \vee A \vdash K \tag{6.3}$$

The premise (6.1) is justified by the *Hypothesis* rule. The system then starts proving judgment (6.2). At this point we notice the way the system passes the list of antecedents. It is important that at no point during decomposition the system deletes any of the antecedents. However, there may be cases when their application can lead to an infinite loop. Rather than passing the list of antecedents in the same sequence as it was inputted, the system places proposition $B \vee A$ after the *Emp* proposition. As described later, this prevents the system from the possibility of entering an infinite loop.

The system starts traversing the list of antecedents in order to determine which antecedent can establish the succedent and once it reads the *Emp* proposition it stops. Thus, for this case the system reads proposition E, knowing that without any other information this proposition does not lead to the succedent the system places the antecedent at the end of the list. After the first computation the judgment is of the form

$$A \wedge B, B, Emp, B \vee A, E \vdash K$$

Next the system checks for proposition $A \wedge B$, and places it at the end of the antecedents' list as it does not lead to the succedent. The same procedure applies for proposition B. Therefore, at this point of computation the system is processing the following judgment

$$Emp, B \vee A, E, A \wedge B, B \vdash K$$

Once it reads the *Emp* proposition the system stops traversing the list and terminates the proof search outputting to the user that the judgment does not hold.

Having said that, the reason for using *Emp* as a proposition is to let the system "hide" certain propositions that their application can lead the proof search into an infinite loop. The system places proposition $B \vee A$ after *Emp* once it concluded the premises for the rule $\vee_E$ so if the judgment does not hold, the system does not reapply the rule $\vee_E$. To get a better intuition of how this is prevented we now consider the computation if the system had to ignore proposition *Emp*.

Computation without the Emp Proposition

$$B \vee A, E, A \wedge B \vdash K$$

The system start traversing the antecedents' list and since the first proposition is $B \vee A$ it opts

to apply the rule $\vee_E$ for which it has to prove the following premises.

$$B \vee A, E, A \wedge B \vdash B \vee A \tag{6.4}$$

$$E, A \wedge B, B, B \vee A \vdash K \tag{6.5}$$

$$E, A \wedge B, A, B \vee A \vdash K \tag{6.6}$$

Again the system justifies judgment (6.4) by the *Hypothesis* rule. For judgment (6.5) we have the same situation as judgment (6.2) but without the *Emp* proposition as an antecedent. We also notice that the system passes the antecedent $B \vee A$ as the last element of the antecedents' list otherwise the system checks the first antecedent and reapplies the rule $\vee_E$.

The system traverses the list as the previous computation. Since propositions $E, A \wedge B$ and B do not lead to the succedent it places them at the end of the list. Thus the judgment is of the form

$$B \vee A, E, A \wedge B, B \vdash K$$

At this point the system is in a similar position to the starting point. It should notice that any other traversing is unnecessary because the judgment does not hold. But since this is an automated system and in this particular case no delimiter is used to show the starting point and the end point for traversing, the system traverses the next proposition which again is $B \vee A$. It reapplies the same reasoning explained above and reapply $\vee_E$. This leads the proof search into an infinite loop.

We conclude that the use of proposition *Emp* guides the system to identify the starting point and the end point of traversing. It also "hides" certain propositions to prevent the possibility of an infinite loop.

SC (Sequent Calculus) Module

This module implements an ATP for the Sequent Calculus. We already defined the general data structure for a proposition however a proof search is not composed from propositions only. It is actually a list of decompositions. For the Sequent Calculus each decomposition consists of a list of antecedents, the succedent, the rule (a Sequent Calculus rule eg. *init*) that justifies the decomposition and the propositions on which the decomposition was applied. Hence proof tree in the Sequent Calculus has the following type:

```
type LeftJudgSC = [Proposition]
type RightJudgSC = Proposition
type ElementSC = (LeftJudgSC,RightJudgSC,(SeqCalRule,[Proposition]))
type ProofTreeSC = [ElementSC]
```

The proof search implementation follows the inference rules for the Sequent Calculus and it proceeds bottom up. We discussed that although the Sequent Calculus has the least amount of non determinism one still has to choose whether to apply a right or a left rule. Thus during

implementation we applied a level of focusing (to minimise further the non determinism) that is for every decomposition the system first applies the right rule if applicable otherwise it opts for the left rule. Moreover, the system includes the implementation for printing the proof tree.

IC (Intercalation Calculus) Module

The IC module is a sub module of the ND (Natural Deduction) module. Similar to the SC module, we defined a type appropriate for proof trees in the Intercalation Calculus. Each element within the list of decompositions consists a list of tuples, each tuple is made up of an antecedent and its scope. Propositions, such as antecedents added by the implementation introduction rule are bounded by a scope and are only valid within that scope. Moreover, each element includes the succedent, the rule that justifies the decomposition and the propositions on which the decomposition was applied. Since we are using the Intercalation Calculus we have to include data structures to represent the *verifications* and *uses*. For this matter we use a boolean where a true represents a *verification* and false is a *use*. Hence a proof tree in this calculus is of the type:

```
type PropScope = (String,Proposition)
type LeftJudg = [PropScope]
type RightJudg = (Proposition,Bool)
type Element = (LeftJudg,RightJudg,(Rule,[Proposition]))
type ProofTree = [Element]
```

There exist two types of inference rules; the left rules and right rules. Similar to the SC module the system applies some focusing and the right rule if it is applicable first, if not it proceeds to apply a left rule. Although this decision minimises considerably the amount of non determinism sometimes we might end up with a longer proof tree.

6.4.3 Proof Search Translators

ND (Natural Deduction) Module

The aim of the system is to reduce the non determinism in Natural Deduction. The system does not compute any proof searches in Natural Deduction however this can be achieved in two different ways. One way to do so is to compute the proof search in Intercalation Calculus and then translate it to Natural Deduction which is the aim for this module. The module contains a sub module, IC, which is responsible for the computation of proof searches in the Intercalation Calculus.

Having computed the proof search in the Intercalation Calculus the task to translate it to the Natural Deduction is fairly simple. The differences between Intercalation Calculus and Natural Deduction are *verifications* and *uses*, and the rule $\downarrow\uparrow$ to change from a *use* to a *verification* reading top down. Hence to translate the proof search we only have to apply Theorem 3, that is we traverse and remove the *use* or *verification* from each element in the list. Since *verifications*

and *uses* are insignificant in Natural Deduction when we are translating proof search deduction the $\downarrow\uparrow$ rule is not translated otherwise we end up with same deductions in the proof search.

The output of this module is a proof tree in the following type

```
type PropScope = (String,Proposition)
type LeftJudg = [PropScope]
type ElementND = (LeftJudg,Proposition,(Rule,[Proposition]))
type ProofTreeND = [ElementND]
```

The data structure is similar to the data structure for the Intercalation Calculus proof tree but without the boolean that represents a *verification* or *use* as these are insignificant in Natural Deduction.

SeqCaltoIntCal (Sequent Calculus to Intercalation Calculus) Module

Proof search translation from the Sequent Calculus to the Intercalation Calculus follows Theorem 6. For this translation the system is expecting a proof tree of type

```
type LeftJudgSC = [Proposition]
type RightJudgSC = Proposition
type ElementSC = (LeftJudgSC,RightJudgSC,(SeqCalRule,[Proposition]))
type ProofTreeSC = [ElementSC]
```

and outputs a proof tree of type

```
type PropScope = (String,Proposition)
type LeftJudg = [PropScope]
type RightJudg = (Proposition,Bool)
type Element = (LeftJudg,RightJudg,(Rule,[Proposition]))
type ProofTree = [Element]
```

Decompositions which are justified by the Sequent Calculus right inference rules are translated from one calculus to another by Theorem 6. However, both calculi handle antecedents in a different manner for the left inference rules. The Intercalation Calculus' decomposition requires the list of antecedents (Γ) for the precedent and the conclusion of the inference rule to be equal. On the other hand, the left inference rules in the Sequent Calculus usually add propositions to the list of antecedents while applying decompositions. Hence we cannot translate directly from one calculus to another. As discussed in Section 4.2 the system applies the Substitution rule as defined by Theorem 5. The theorem requires two inputs ($\Gamma \vdash A \downarrow$ and $\Gamma, A \downarrow\vdash B \downarrow$) in order to conclude $\Gamma \vdash B \downarrow$. The first input ($\Gamma \vdash A \downarrow$) is proved directly in the Intercalation Calculus by invoking the IC ATP. We decided to directly compute a proof in the Intercalation Calculus rather than compute it in the Sequent Calculus because if we had to follow the latter option and translate it to Intercalation Calculus we would have increased the level of complexity of the proof tree. The second required input ($\Gamma, A \downarrow\vdash B \downarrow$) is translated directly from the proof search for the Sequent Calculus.

PT (Proof Terms) Module

Lambda_Calculus Module

Before we describe the implementation for Proof Terms, we shall give an overview of the Lambda Calculus module. Every decomposition (or judgment) in a Proof Terms' proof search requires a lambda term which is the type for the succedent. The lambda term `LTerm` is defined as:

```
data LTerm =  Inr Proposition LTerm                          <∨_{I_L}>
        | Inl Proposition LTerm                              <∨_{I_R}>
        | Variable String                                    <hyp>
        | App LTerm LTerm                                    <⊃_E>
        | Abs String Proposition LTerm                       <⊃_I>
        | Pair LTerm LTerm                                   <∧_I>
        | First LTerm                                        <∧_{E_L}>
        | Second LTerm                                       <∧_{E_R}>
        | OrInlInr LTerm String LTerm String LTerm           <∨_E>
        | Abort Proposition LTerm                            <⊥_E>
        | Empty                                              <⊤_I>
```

Moreover, this module implements the reduction methodology for the lambda calculus as described in Section 3.4.2. We opted to implement an applicative order reduction sequence which states that arguments should be reduced before passed to the function. This reduction methodology is the simplest to implement and it needs the least amount of overhead which is the reason why languages such as C and ML use this evaluation strategy. However, this method will not terminate if a non terminating function is passed as an argument.

Another functionality of this module is renaming of bound variables whenever a variable capture situation (the situation when free variables become bounded variables) arises. We discussed in Section 3.4 that care should be taken when substituting arguments in functions. A variable capture situation arises whenever the argument contains free variables which are bounded in the function. The system handles such situations by renaming the function's bounded variables with a new term. This check takes place before substitution. Hence, whenever substitution is required the system first checks that $BV(function) \cap FV(argument) = \phi$. If the set is not empty then the system checks which variables in the function are violating the check and rename them with another (not already used) variable name.

PT Module

Unlike the other translations, where the system follows a theorem and translate accordingly, this translation module has a different approach. This system takes a proof tree in Natural Deduction as an input (`type ProofTreeND`) and outputs a proof tree in Proof Terms terminology. The data structure contains a variable of type LTerm which represents the type for the succedent. Thus, the data structure for proof trees in Proof Term is

```
type PropScope = (String,Proposition)
type LeftJudg = [PropScope]
type PropLTerm = (LTerm,Proposition)
type ElementPT = (LeftJudg,PropLTerm,(Rule,[Proposition]))
type ProofTreePT = [ElementPT]
```

Moreover, for translation the system follows the one-to-one correspondence between the Natural Deduction and Proof Terms where it compares inference rules and compute types according to the decompositions in the proof tree for Natural Deduction. It gets the rule that justifies the decomposition for each element in the proof tree in Natural Deduction, compares it to the equivalent rule in the Proof Terms Calculus and adds the appropriate lambda term for that judgment (the correspondence between the lambda term and the inference rule is shown in the definition of `LTerm`). For instance for the judgment `A,B⊢A` the system translates it to `a:A,b:B⊢a:A` where the uppercase literal defines a proposition and its equivalent in lowercase defines the assigned type. The translation proceeds in a top down technique for the reason that the composition of the lambda term for a decomposition in the proof tree is dependent on the lambda term of the previous decomposition. As we already alluded to in Sec for every lambda term composition the system has to compute a lambda reduction when it is possible. Any lambda reduction is processed by the Lambda Calculus module.

6.5 Achievements

In Chapter 1 we set our aim to create a system that automates better proof search for Natural Deduction. From the design and the implementation of the system we conclude the following achievements:

- Since the proof search in the Sequent Calculus performs better than in Natural Deduction, the system uses the correspondence between the Natural Deduction and Sequent Calculus to compute the proof search in the Sequent Calculus. The achievement for this functionality is system performance as proof search in the Sequent Calculus performs better due to the less non deterministic factors in this calculus in comparison to Natural Deduction. This achievement is echoed in both functionalities of the system; the verifier and one of the automated theorem provers (method 2).

- Using proof terms the system outputs an encoding in the Lambda Calculus for the actual proof tree. As defined in Section 3.4 using the one to one correspondence between the Natural Deduction and Proof Terms one can compute the proof search from the type assigned to the judgment's succedent (ref to Section 3.4.3). Although we are translating a proof search from Natural Deduction to Proof Terms, thus, we are computing the types for judgments, the final judgment's type computation is actually a lambda encoding to the proof tree. This achievement is significant if we have a memory restriction where instead of saving the actual proof tree we could save the type then when we retrieve the type we can construct the proof tree which is a sequential process.

6.6 Errors and Limitations

6.6.1 Notation

Since in this section we illustrate examples as computed by the system, we give a brief explanation of the notation. Table 6.3 shows the equivalent representation to each logical connective.

Logical Connective	Representation	Logical Connective	Representation
$\wedge$	&	$\supset$	->
$\vee$	|	$\bot$	!
$\top$	.	$\vdash$	|-

Table 6.3: System's Logical Connective Representation

Furthermore, every decomposition in a proof search is justified by a rule. Other than the rule the proof search outputs the propositions which were the premises for the rule. Therefore in case of the following proof search (the proof search in the right) `ConjIntro - A,B` means that the decomposition is justified by the conjunction introduction rule ($\wedge_I$) rule on the proposition `A` and `B` which were the premises for the rule $\wedge_I$. The indentation in a proof search shows the sub proofs for a particular decomposition.

$$\Big|\overline{A, B \vdash A}^{hyp} \qquad \longrightarrow \qquad \Big|\overline{A, B \vdash B}^{hyp}$$
$$\overline{A, B \vdash A \wedge B}^{\wedge_I}$$

```
A,B |- A     Hypothesis - A

A,B |- B     Hypothesis - B
A,B|- A&B    ConjIntro - A, B
```

6.6.2 Limitations

In cases where the proof search requires to proof the proposition $\bot$ as the conclusive succedent, the system performs a redundant decomposition. This happens because the system checks to apply the inference rule $\supset_E$(`ImplElim`) before it checks to apply the $\bot_E$(`FalsehoodElim`) rule (refer to lines 4 and 5). Thus, for the case when the system has the antecedents $A\supset$ $\bot$(`A->!`) and A(`A`), the system first performs an $\supset_E$ and obtains $\bot$(`!`), then it applies the redundant decomposition $\bot_E$ rule. This case is clearly shown in example 6.6.1

Example 6.6.1. For the input `(a&b), (a->!)|- !` the system outputs the following proof search:

```
1. (a&b), (a->!) |- (a->!)          Hypothesis -- (a->!)

2. (a&b), (a->!) |- (a&b)          Hypothesis -- (a&b)
3. (a&b), (a->!) |- a          ConjElimLeft -- (a&b)
```

```
4. (a&b), (a->!) |- !          ImplElim -- (a->!), a
5. (a&b), (a->!) |- !          FalsehoodElim
```

In order to prove $\perp(!)$ it is sufficient to prove till step `4` however the system performs the redundant decomposition of step `5`. Nonetheless the proof search is still a valid proof search.

The other limitation is an aesthetic limitation rather than an implementation limitation. Proof searches for Proof Terms contain the propositions and lambda terms. For representation the system does not differ between propositions and terms if these happen to be represented by the same character. For instance lets consider the following example:

Example 6.6.2. For the input `a,b|-a&b` the system outputs the following:

```
1. a:a, b:b |- a: a          Hypothesis -- a

2. a:a, b:b |- b: b          Hypothesis -- b
3. a:a, b:b |- <a,b>: (a&b)          ConjIntro -- a, b
```

Care should be taken when reading the proof tree. The succedent for line `1` is `a:a` where the first `a` is a lambda term while the second `a` is an atomic proposition. Unfortunately, for cases like these the system does not assign a different representation for the lambda term which may require more effort from the reader to comprehend the proof.

6.6.3 Errors

From the testing carried out the known error of the system is based on how the system handles the list of antecedents. The propositions' order in which the user inputs the list of antecedents is crucial to the system. The system processes this list sequentially, it analyses propositions and try to gather enough information to apply an inference rule. For most of the rules the order of the list is insignificant however it is important if the list contains any disjunction propositions. Most of the time if the list contains a disjunction proposition then we have to apply the rule $\vee_E$ first. However, if the system finds another proposition which may lead to the conclusive succedent the system decomposes it. This approach may lead the system into an infinite loop or to no output. Consider the two inputs:

$$a \vee b, a \supset c, b \supset c \vdash c \tag{6.7}$$

$$a \supset c, a \vee b, b \supset c \vdash c \tag{6.8}$$

Before we start computation for the judgments above we recall the inference rules required.

$$\frac{\Gamma \vdash A \supset B \qquad \Gamma \vdash A}{\Gamma \vdash B} \supset E$$

$$\frac{\Gamma \vdash A \vee B \quad \Gamma, u : A \vdash C \quad \Gamma, v : B \vdash C}{\Gamma \vdash C} \vee E^{u,v}$$

Example 6.6.3. RTP: $a \vee b, a \supset c, b \supset c \vdash c$

The proof starts traversing the antecedents' list and since the first proposition is $A \vee B$ it applies the rule $\vee_E$. For the application of this rule the system is required to proof the premises

$$a \vee b, a \supset c, b \supset c \vdash a \vee b \tag{6.9}$$

$$a \vee b, a \supset c, b \supset c,\ a \vdash c \tag{6.10}$$

$$a \vee b, a \supset c, b \supset c,\ b \vdash c \tag{6.11}$$

Judgment (6.9) is justified by the *Hypothesis* rule. For judgment (6.10) the system applies the $\supset_E$ rule and obtain c:

$$\Big|\overline{a \supset c, b \supset c,\ a,\ a \vee b \vdash a \supset c}^{hyp}$$

$$\dfrac{\Big|\overline{a \supset c, b \supset c,\ a,\ a \vee b \vdash a}^{hyp}}{a \supset c, b \supset c,\ a,\ a \vee b \vdash c}^{\supset_E}$$

The proof search for judgment (6.11) is similar to the one above thus the final proof search is:

$$\dfrac{\begin{array}{l}\Big|\overline{a \vee b, a \supset c, b \supset c \vdash a \vee b}^{hyp} \\ \\ \dfrac{\begin{array}{l}\Big|\overline{a \vee b, a \supset c, b \supset c,\ a\ \vdash a \supset c}^{hyp} \\ \\ \Big|\overline{a \vee b, a \supset c, b \supset c,\ a\ \vdash a}^{hyp}\end{array}}{a \vee b, a \supset c, b \supset c,\ a\ \vdash c}^{\supset_E} \\ \\ \dfrac{\begin{array}{l}\Big|\overline{a \vee b, a \supset c, b \supset c,\ b\ \vdash b \supset c}^{hyp} \\ \\ \Big|\overline{a \vee b, a \supset c, b \supset c,\ b\ \vdash b}^{hyp}\end{array}}{a \vee b, a \supset c, b \supset c,\ b\ \vdash c}^{\supset_E}\end{array}}{a \vee b, a \supset c, b \supset c \vdash c}^{\vee_E}$$

Example 6.6.4. RTP: $a \supset c, a \vee b, b \supset c \vdash c$

The system starts traversing the antecedents' list where the first antecedent is $a \supset c$. Since it is an implication which contains proposition c the system applies the $\supset_E$ to obtain c. One of the premises for $\supset_E$ is to prove proposition a. At this point the system gets stuck cause there is no information that the system can extract to obtain a. Thus, it terminates outputting to the

user that the judgment does not hold.

$$\Big|\ \frac{}{a \supset c, b \supset c, a \vee b \vdash a \supset c}\,hyp$$

$$\frac{\Big|\ \begin{array}{l}\vdots\ \ stuck\ state\\ \hline a \supset c, b \supset c, a \vee b \vdash a\end{array}?}{a \supset c, b \supset c, a \vee b \vdash c}\,\supset_E$$

From examples 6.6.3 and 6.6.4 we notice that the order in which the antecedents are listed is important especially when the system has to apply the rule $\vee_E$.

7. Evaluation

7.1 Introduction

Human beings use reasoning all the time to analyse what happens if a particular situation had to arise (e.g. *If John is older than 16 years old than he is eligible to sit for the exam*). Natural Deduction is the most straight forward calculus usually used to compute proofs to verify a particular judgment. During the development of this system we studied what are the disadvantages of using such calculus for proof search computations. It turned out, that the definition of the inference rules for the Natural Deduction give several paths that the proof search can follow.

We studied the Intercalation Calculus which restricts introduction rules to be used only in a bottom up technique and elimination rules to be used only in a top down technique. Then we analysed the Sequent Calculus which restricts the proof search to follow a bottom up technique only. Lastly, we studied the relationship between these calculi and how a judgment can be translated from one calculus into another (section 5.1.2). In order to minimise the non determinism in the Natural Deduction we concluded that the proof search should be computed in the Sequent Calculus, translated to the Intercalation Calculus then to Natural Deduction.

The system presented offers the user two different ways (which both methodologies reduce non determinism in proof search) how to obtain a proof tree in Natural Deduction:

- Method 1: Proof search computed in the Intercalation Calculus then it is translated to the Natural Deduction.
- Method 2: Proof search computed in the Sequent Calculus, the system then translates it to Intercalation Calculus then it is translated to the Natural Deduction.

We made our effort to ensure that our system outputs correct proof searches which sometimes are intricate yet comprehensive. In this chapter we describe the output of the system and whether we have achieved our goals by the implementation of our system.

7.2 Testing Plan

As with all non trivial programs, we had to make sure that our testing plan shows the maximum amount of errors in the minimum time possible. Initially, we considered to carry out testing using a proper tool to exhaust the system's performance. We could have tested our system using a tool which can generate propositions automatically, however, while developing the system we decided to carry out testing using single test cases which were specifically aimed to show the system response to potential crucial problems. The system's concern is proof search correctness

rather than the use of machine's resources and performance speed. For this reason, we opted to test test cases sequentially and verify the output of each test case.

In fact, testing did brought some core implementation issues to light which were handled during development. Although all the logical connectives were tested, most of the test cases were based on the response of the system to the disjunction elimination logical connective. As already alluded to, this logical connective is the one which can result in the creation of most of the non determinism during proof tree construction. Lastly, after every single test case we analysed and verified that the proof search is correct and the right approach was carried out to mainly reduce non determinism during proof search.

7.3 Proof Correctness

In order to conclude whether a specific judgment holds, we have to ensure that the proof tree computation abides by the inference rules. Since a proof search in Natural Deduction can be computed in two different methodologies in the presented system, during implementation we had to ensure that the output given to the user is always correct irrelevant which system is used for proof tree computation.

In this section we intend to compare the resultant proof tree for both approaches and report where the implementation was successful and where it failed. Several proof searches are documented in this document. These are documented in the actual format outputted by the system. Table 7.1 shows the equivalent representation to the logical connectives.

Logical Connective	Representation	Logical Connective	Representation
$\wedge$	&	$\supset$	->
$\vee$	|	$\bot$	!
$\top$	.	$\vdash$	|-

Table 7.1: System's Logical Connective Representation

Thus the two proof searches listed below are equal. One is shown in the actual notation for Natural Deduction while the other (RHS) is shown as outputted by the system.

Example 7.3.1. RTP - A,B,C $\vdash (A \wedge B)$
System's Input : `A, B, C |- (A&B)`

$$\dfrac{\dfrac{\Big|}{A, B, C \vdash A}\,hyp \quad \dfrac{\Big|}{A, B, C \vdash B}\,hyp}{A, B, C \vdash (A \wedge B)}\,\wedge_I$$

Proof Search using correct notation

```
1. A, B, C |- A      Hypothesis – A
2. A, B, C |- B      Hypothesis – B
3. A, B, C |- (A&B)  ConjIntro – A, B
```

Proof Search computed by the system

7.3.1 Logical Connectives' Inference Rules Evaluation

To check whether the proof tree is correct we recall each rule from the Natural Deduction, compute proof searches in both methodologies implemented for Natural Deduction using the respective rule and then we evaluate the final proof search.

Conjunction

Inference rule:

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B} \wedge_I$$

This rule guides us to proof propositions A and B to conclude the proposition $A \wedge B$. Example 7.3.2 shows the computation for the judgment `A,B,C|-A&B` using both methodologies.

Example 7.3.2. RTP - A,B,C $\vdash (A \wedge B)$
System's Input : `A, B, C |- (A&B)`

Proof 7.3.2.1

`1. A, B, C |- A`	*Hypothesis* $- A$
`2. A, B, C |- B`	*Hypothesis* $- B$
`3. A, B, C |- (A&B)`	*ConjIntro* $- A, B$

Proof Search computed using Method 1

Proof 7.3.2.2

`1. A, B, C |- A`	*Hypothesis* $- A$
`2. A, B, C |- B`	*Hypothesis* $- B$
`3. A, B, C |- (A&B)`	*ConjIntro* $- A, B$

Proof Search computed using Method 2

The proof search starts off by proving propositions A and B. In this case since both are elements of the antecedent list their respective proof is justified by the *Hypothesis* rule. We then apply the $\wedge_I$ rule to obtain `A,B,C|-A&B`. Although both proof searches were computed using different calculi the correct resultant proof search is identical.

Inference Rule:

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A} \wedge_{E_L} \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B} \wedge_{E_R}$$

The rule defines that propositions A and B can be obtained if we know that proposition $A \wedge B$ *is true*. Before we prove correctness for this rule we recall how computation is carried out.

For Method 1 we are using the Intercalation Calculus and translate the whole proof search into Natural Deduction. The actual proof is shown in example 7.3.3 (proof 7.3.3.1) while in example 7.3.3 (proof 7.3.3.2) we are showing the proof search in Natural Deduction. Proof search 7.3.3.1 applies the Intercalation Calculus rule

$$\frac{\widehat{\Gamma} \vdash A \wedge B \downarrow}{\widehat{\Gamma} \vdash B \downarrow} \wedge_{E_R}$$

to obtain proposition $B\downarrow$.

Example 7.3.3. RTP - (A$\wedge B$) $\vdash B$
System's Input : `(A&B) |- B`

Proof 7.3.3.1

`1. (A&B) |- (A&B)` $\downarrow$ *Hypothesis* $-(A\&B)$
`2. (A&B) |- B` $\downarrow$ *ConjElimRight* $-(A\&B)$
`3. (A&B) |- B` $\uparrow$ *LeftRight* $- B$

Proof Search using Intercalation Calculus

Proof 7.3.3.2

`1. (A&B) |- (A&B)` *Hypothesis* $-(A\&B)$
`2. (A&B) |- B` *ConjElimRight* $-(A\&B)$

Proof Search using Natural Deduction

Theorem 3. (Intercalation Calculus to Natural Deduction)
If $\widehat{\Gamma} \vdash A \uparrow$ *then* $\Gamma \vdash A$ *and*
$If \widehat{\Gamma} \vdash A \downarrow$ *then* $\Gamma \vdash A$. [19]

Recalling Theorem 3, in example 7.3.3 we translate the proof search 7.3.3.1 to 7.3.3.2 . According to the translation, proposition B is justified by the rule $\wedge_{E_R}$ which is correct.

On the other hand, to evaluate correctness for proofs computed using the second methodology (Method 2) we have to once again recall the difference between the Natural Deduction calculus and the Sequent Calculus. In Section 3.3 we discussed that antecedents are persisted in the Sequent Calculus because proof search follows a bottom up technique and backtracking is not allowed. For this reason "left" rules add propositions to the list of antecedents after decomposition. One of the Sequent Calculus "left" rules is the $\wedge_{L_2}$ rule defined as:

$$\frac{\widetilde{\Gamma}, A \wedge B, B \Rightarrow C}{\widetilde{\Gamma}, A \wedge B \Rightarrow C} \wedge_{L_2}$$

In fact when comparing rules $\wedge_{L_2}$ and $\wedge_{E_R}$ we notice that set of antecedents is retained after decomposition when the rule $\wedge_{E_R}$ is applied in the Natural Deduction Calculus. However, this is not the case for the Sequent Calculus rule $\wedge_{L_2}$ which adds propositions to the antecedents' list during decomposition. This difference makes it impossible to translate proof search directly from the Sequent Calculus to the Natural Deduction. Hence we first translate the proof search from the Sequent Calculus to the Intercalation Calculus using Theorem 6 and then resort to Theorem 5 to handle these differences.

Theorem 6. (Sequent Calculus to Intercalation Calculus)
If $\widetilde{\Gamma} \vdash C$ *right then* $\widehat{\Gamma} \vdash C \uparrow$.[24]

Theorem 5. (Substitution for Intercalation) Assume $\widehat{\Gamma} \vdash A \downarrow$.*Then*
1. $If \widehat{\Gamma}, A \downarrow\!\!\vdash B \downarrow$ *then* $\widehat{\Gamma} \vdash B \downarrow$, *and*
2. $If \widehat{\Gamma}, A \downarrow\!\!\vdash C \uparrow$ *then* $\widehat{\Gamma} \vdash C \uparrow$.[24]

Finally we conclude our translation by applying Thereom 3 to obtain the proof search in Natural Deduction.

The reason why we gave the actual algorithm for computation using the second option is to give an intuition how we have to handle proof correctness for proofs computed using this methodology. Example 7.3.4 shows the proof for judgment computed by these two different methodologies. Proof 7.3.4.1 resembles an actual proof search computed in Natural Deduction. At first glance, the user may think that the proof search for proof 7.3.4.2 is wrong however following the description above the proof is correct.

Example 7.3.4. RTP - (A$\wedge B$) $\vdash B$
System's Input : `(A&B) |- B`

Proof 7.3.4.1

`1. (A&B) |- (A&B)` *Hypothesis* $-(A\&B)$
`2. (A&B) |- B` *ConjElimRight* $-(A\&B)$

Proof Search computed using Method 1

Proof 7.3.4.2

`1. (A&B) |- (A&B)` *Hypothesis* $-(A\&B)$
`2. (A&B) |- B` *ConjElimRight* $-(A\&B)$
`3. B,(A&B) |- B` *Hypothesis* $-B$
`4. (A&B) |- B` *Substitution*

Proof Search computed using Method 2

Implication

Rule:

$$\frac{\Gamma, u : A \vdash B}{\Gamma \vdash A \supset B} \supset I^u$$

We check correctness for rule $\supset I^u$, by checking that the list of antecedents and proposition A within the scope u are valid and lead to proposition B. The best way to evaluate whether decompositions using this rule are correct is to compare two examples worked out using Method 1 and Method 2.

Example 7.3.5. RTP - C,B $\vdash A \supset B$
System's Input : `C, B |- (A->B)`

Proof 7.3.5.1

`1. a:A, C, B |- B` *Hypothesis* $-B$
`2. C, B |- (A->B)` *ImplIntro* $-A, B$

Proof Search computed using Method 1

Proof 7.3.5.2

`1. a:A, C, B |- B` *Hypothesis* $-B$
`2. C, B |- (A->B)` *ImplIntro* $-A, B$

Proof Search computed using Method 2

Although both proofs in example 7.3.5 were computed in a different calculus and neither in

the Natural Deduction, in both cases the translated proof search is correct. Once again the translated proof search is identical for both methodologies.

Inference Rule:

$$\frac{\Gamma \vdash A \supset B \quad \Gamma \vdash A}{\Gamma \vdash B} \supset_E$$

Example 7.3.6. RTP - A, A$\supset B \vdash B$

System's Input : `A, (A->B) |- B`

Proof 7.3.6.1

```
1. A, (A->B) |- (A->B)      Hyp. – (A->B)

2. A, (A->B) |- A           Hyp. – A
3. A, (A->B) |- B           ImplElim – (A->B), A
```

Proof Search computed using Method 1

Proof 7.3.6.2

```
    1. A, (A->B) |- (A->B)      Hyp. – (A->B)

    2. A, (A->B) |- A           Hyp. – A
  3. A, (A->B) |- B             ImplElim – (A->B), A

    4. B, A, (A->B) |- B        Hyp. – B
5. A, (A->B) |- B               Substitution
```

Proof Search computed using Method 2

The rule $\supset_E$ defines that in order to obtain the succedent B of an implication $A \supset B$ we must prove proposition A. Showing correctness for this rule is trivial if the first methodology is used for proof computation. The Intercalation Calculus corresponding rule to $\supset_E$ is

$$\frac{\widehat{\Gamma} \vdash A \supset B \downarrow \qquad \widehat{\Gamma} \vdash A \uparrow}{\widehat{\Gamma} \vdash B \downarrow} \supset_E$$

Comparing $\supset_E$ for the Intercalation Calculus and $\supset_E$ for Natural Deduction they are clearly similar. The difference is the use of *verifications* and *uses* which during translation these are eliminated by the application of Theorem 3.

Both proofs for Proof 7.3.6 are identical for the reason that proposition B is implied by the implication. However, there can be cases when the eliminated proposition by the rule $\supset_E$ requires further processing to prove the succedent (for example $A, A \supset A \wedge B \vdash B$). Since the rule for the Sequent Calculus persists propositions then (in such cases) one has to use the *Substitution* rule. Although the equivalent Sequent Calculus rule $\supset_L$ (ref. to $A.3$) for the Natural Deduction rule $\supset_E$ requires two premises $\widetilde{\Gamma}, A \supset B \Rightarrow A$ and $\widetilde{\Gamma}, A \supset B, B \Rightarrow C$ to conclude $\widetilde{\Gamma}, A \supset B \Rightarrow C$, only one premise persists its propositions. During the translation between the Sequent Calculus and the Intercalation Calculus using theorem 6 the system applies theorem 5 to solve the proposition persistence for the premise $\widetilde{\Gamma}, A \supset B, B \Rightarrow C$.

Truth

Inference Rules:

$$\frac{}{\Gamma \vdash \top} \top I$$

Example 7.3.7. RTP - A $\vdash \top$
System's Input : `A |- .`

Proof 7.3.7.1	*Proof* 7.3.7.2
`1. A |- .` *TruthIntro*	`1. A |- .` *TruthIntro*
Proof Search computed using Method 1	*Proof Search computed using Method* 2

Remark: The notation . (*point*) stands for the character $\top$.

Truth is always introduced without any premises needed no matter which methodology the system uses for computation. Example 7.3.7 shows the two proof searches computed using methodology 1 and 2. As expected the judgment `A |- .` is justified by the rule $\top_I$ in both cases. Hence, the implementation for this inference rule is correct.

Falsehood

Inference Rules:

$$\frac{\Gamma \vdash \bot}{\Gamma \vdash C}\bot E$$

Example 7.3.8. RTP - $\bot \vdash A$
System's Input : `!  |- A`

Proof 7.3.8.1	*Proof* 7.3.8.2
`1. !  |- !` *Hypothesis–!* `1. !  |- A` *FalsehoodElim*	`1. !  |- !` *Hypothesis–!* `1. !  |- A` *FalsehoodElim*
Proof Search computed using Method 1	*Proof Search computed using Method* 2

Rule $\bot_E$ states that knowing that a proposition is *False* then we can conclude any other proposition. For example 7.3.7, the proof search is given the proposition $\bot$(denoted as *!*) as an antecedent. For both proofs the system concludes proposition A justifying the decomposition by the rule $\bot_E$. Thus, the application of rule $\bot_E$ during proof search is implemented correctly.

Disjunction

Rule:

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B}\vee_{I_L}$$

Similar to other introduction rules, the equivalent rules in the Sequent Calculus and the Intercalation Calculus to the rule are very similar as it can be noticed from the rules defined below.

$$\frac{\widehat{\Gamma} \vdash A\uparrow}{\widehat{\Gamma} \vdash A \vee B\uparrow}\vee_{I_L} \qquad \frac{\widetilde{\Gamma} \Rightarrow A}{\widetilde{\Gamma} \Rightarrow A \vee B}\vee_{R_1}$$

The similarity between rules makes it easier to check for proof search correctness in the Natural Deduction Calculus. All the rules reflect the idea that if we know that propositions *A is true* or *B is true* then we can conclude $A \vee B$ *is true*. In order to show correctness in the system implemented in example 7.3.9 we show the proof search computed in both methodologies for judgment `A|-(A|(B|C))`.

Example 7.3.9. RTP - $A \vdash (A \vee (B \vee C))$
System's Input : `A |- (A|(B|C))`

Proof 7.3.9.1

`1. A |- A` *Hypothesis* $-$ *A*
`2. A |- (A|(B|C))` *DisjIntroLeft* $-$ *A*

Proof Search computed using Method 1

Proof 7.3.9.2

`1. A |- A` *Hypothesis* $-$ *A*
`2. A |- (A|(B|C))` *DisjIntroLeft* $-$ *A*

Proof Search computed using Method 2

Rule:

$$\frac{\Gamma \vdash A \vee B \quad \Gamma, u : A \vdash C \quad \Gamma, v : B \vdash C}{\Gamma \vdash C} \vee E^{u,v}$$

Checking for proof correctness for this rule does not include any intuitions which require the application of the *Substitution* rule to successfully translate the proof search to Natural Deduction Calculus. Translation for this decomposition follows Theorem 3 for proof search computed in the Intercalation Calculus. Similarly, translation for proof search computed using the second methodology follows Theorem 6.

Example 7.3.10. RTP - $A \vee B, \bot \vdash C$
System's Input : `(A|B), ! |- C`

Proof 7.3.10.1

`1. (A|B), ! |- (A|B)` *Hypothesis* $-$ $(A|B)$

`2. (A|B), !, a:A |- !` *Hypothesis* $-$ *C*
`3. (A|B), !, a:A |- C` *FalsehoodElim*

`4. (A|B), !, b:B |- !` *Hypothesis* $-$ *C*
`5. (A|B), !), b:B |- C` *FalsehoodElim*
`6. (A|B), ! |- C` *DisjElim* $-$ $(A|B)$

Proof Search computed using Method 1

Proof 7.3.10.2

`1. (A|B), ! |- (A|B)` *Hypothesis* $-$ $(A|B)$

`2. (A|B), !, a:A |- !` *Hypothesis* $-$ *C*
`3. (A|B), !, a:A |- C` *FalsehoodElim*

`4. (A|B), !, b:B |- !` *Hypothesis* $-$ *C*
`5. (A|B), !, b:B |- C` *FalsehoodElim*
`6. (A|B), ! |- C` *DisjElim* $-$ $(A|B)$

Proof Search computed using Method 2

We evaluate correctness for this rule using example 7.3.10. Both proofs start by proving the first premise (line 1) which is justified by the *Hypothesis* rule. The two then follow to prove the premises `(A|B),!,a:A |- C` and `(A|B),!,b:B |- C` separately.

Each sub proof proves the Falsehood (`!`) proposition justified by the *Hypothesis* rule. Then, in each case the proposition C is decomposed from the succedent Falsehood (`!`) justified by the $\perp_E$ (falsehood elimination) rule.

Conclusion

After evaluating proof searches for each individual inference rule for the Natural Deduction Calculus we conclude that proof searches computed by either Method 1 or 2 are computed correctly. Examples show us (especially examples 7.3.4 and 7.3.6) that the resultant proof search may be longer if computed with method 2 instead of method 1, nonetheless, both methods compute the proof tree correctly.

7.3.2 Proof Terms

After evaluating whether the resultant proof search is correct or not, in this section we discuss proof searches computed in the Proof Terms Calculus. As discussed in Section 3.4, the proof search for Proof Terms is a translation for the proof search in the Natural Deduction. Although the system offers two methodologies for proof search computation in Natural Deduction, this part of the system uses Method 2 for its proof search computation in Natural Deduction then it translates it to Proof Terms. Hence, as we already evaluated proof correctness for the Natural Deduction we do not have to evaluate it for this module. On the other hand, we evaluate whether the transition from the Natural Deduction to Proof Terms is correct and whether the Lambda Calculus term generated is valid.

Having evaluated every rule for the Natural Deduction and proof searches composed from a combination of rules we opt to evaluate the correctness for proof searches using Proof Term by examples.

Example 7.3.11. RTP - B $\vdash ((A \supset (B \wedge A)) \vee C)$
System's Input : `B |- ((A->(B&A))|C)`

```
    1. b:B, a:A |- b:  B     Hypothesis − B

    2. b:B, a:A |- a:  A     Hypothesis − A
 3. b:B, a:A |- <b,a>:  (B&A)     ConjIntro − B, A
 4. b:B |- \a:A.<b,a>:  (A->(B&A))     ImplIntro − A, (B&A)
 5. b:B |- inl (C) (\a:A.<b,a>):  ((A->(B&A))|C)     DisjIntroLeft − (A->(B&A))
```

First we recall the rules $\wedge_I$, $\vee_{I_L}$ and $\supset_I$ for Proof Terms.

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash N : B}{\Gamma \vdash \langle M, N \rangle : A \wedge B} \wedge_I \qquad \frac{\Gamma \vdash M : A}{\Gamma \vdash \mathbf{inl}^B M : A \vee B} \vee_{I_L} \qquad \frac{\Gamma, u : A \vdash M : B}{\Gamma \vdash \lambda u : A.M : A \supset B} \supset_I$$

In example 7.3.11 lines 1 and 2 are two different sub proofs but both are justified by the *Hypoth-*

esis rule. Both subproofs are the premises for the rule $\wedge_I$ which is used to compose line 3. The term `<b,a>` is the pair which defines the type for the conjunction `B&A`. The system then applies the rule $\supset_I$ on line 3. As the rule $\supset_I$ assigns the abstraction term to the type of its premise, the type for the term `(A->(B&A))` is `(\a:A.<b,a>)`. Finally, the prove is concluded by the application of the $\vee_{I_L}$ rule. Hence the final type composed for the proposition `((A->(B&A))|C)` is `inl (C) (\a:A.<b,a>)`. The type `inl (C) (\a:A.<b,a>)` shows that the proposition C was introduction on the RHS of a proposition which has the assigned type `(\a:A.<b,a>)`.

Example 7.3.12. RTP - $((\mathrm{A} \vee B) \wedge C) \vdash (A \vee B)$
System's Input : `((A|B)&C) |- (A|B)`

`1. a:((A|B)&C) |- a: ((A|B)&C)` *Hypothesis* – $((A|B)\&C)$
`2. a:((A|B)&C) |- fst (a):(A|B)` *ConjElimLeft* – $((A|B)\&C)$

`3. a:((A|B)&C), d:(A|B) |- d:(A|B)` *Hypothesis* – $(A|B)$
`4. a:((A|B)&C) |- d:(A|B)` *Substitution*

Recall rule:

$$\frac{\Gamma \vdash M : A \wedge B}{\Gamma \vdash \pi_1 M : A} \wedge_{E_L}$$

At first glance the reader asks why the proof search is not composed from lines 1 and 2 only since they suffice the goal judgment. Once again we recall that the proof search is computed in the Sequent Calculus and translated to the Proof Terms (through the translation to Intercalation Calculus and Natural Deduction). We discussed that while the rules in the Sequent Calculus persists and add propositions to the antecedents' list, the other calculi persist the antecedents' list and no other propositions are added to the list of antecedents. To solve this difference we resort to the *Substitution* rule.

The application of the substitution rule unveils an undesired affect. Comparing lines 2 and 3 from example 7.3.12 the reader gets the impression that the types `d` and `fst(a)` are different yet these are the type for the proposition `(A|B)`. As discussed in Section 6.6 the system assigns a representative type for the proposition because the system is not given enough information to formulate the proposition's type. Hence both types `d` and `fst (a)` are just a representation to the actual type for the proposition `(A|B)`. We conclude that although type `d` is actually correct as it abides by the *Substitution* rule, type `fst (a)` gives a better meaning to the system.

Conclusion

Section 3.4.2 gives a detailed method how Lambda terms or proposition types are reduced. During system's evaluation we noted that it is very rare that a computed term gets reduced. In Section 3.4.2 we defined that the causes for reduction result from the nature of the Proof Terms calculus itself.

Proof search in this calculus has the same problems as defined for the Natural Deduction,

that is, proof search can be computed using both bottom up and top down technique, and when the proof search reaches a stuck state it has to backtrack. Hence, by the definition of this calculus the proof search in example 7.3.13 is possible.

Example 7.3.13. RTP - A $\vdash A$

```
1. a:A |- a:A      Hypothesis − A
```

```
2. a:A |- a:A      Hypothesis − A
3. a:A |- <a,a>:A      ConjIntro − A, A
4. a:A |- fst <a,a>:A      ConjElimLeft − (A&A)
5. a:A |- a:A      Reduced term : fst < x, y >= x
```

Due to the fact that the proof search for Proof Terms within our system is actually computed in the Sequent Calculus then translated to Proof Terms the mentioned problems are eliminated. The Sequent Calculus does not support proof tree construction in a top down technique but the tree is built in a bottom up technique. Furthermore, due to antecedents' persistence Sequent Calculus disallow any backtracking thus proof searches as illustrated in example 7.3.13 are not allowed. Therefore, we conclude that during computation the system does not reduce the lambda term as the proof search does not contain any redundant decompositions.

The only cases where the proof search can reduces terms is when the actual term for the proposition is passed by the user. This is illustrated in example 7.3.14. The first computation (proof 7.3.14.1) mirrors how the current system works where for the antecedent `(A&B)&C` it assigns a representative type. On the other hand, the second computation (proof 7.3.14.2) shows how the term is reduced if a meaningful term is assigned to the antecedent `(A&B)&C`.

Example 7.3.14. RTP - $((\text{A} \wedge B) \wedge C) \vdash C$
System's Input : `(A&B)&C |- C`

Proof 7.3.14.1

```
1. v:(A&B)&C |- v:(A&B)&C      Hypothesis − (A&B)&C
2. v:(A&B)&C |- snd (v):C      ConjElimLeft − (A&B)&C
```

Proof 7.3.14.2

```
1. «a,b>,c>:(A&B)&C |- «a,b>,c>:(A&B)&C      Hypothesis − (A&B)&C
2. «a,b>,c>:(A&B)&C |- snd «a,b>,c>:C      ConjElimLeft − (A&B)&C
3. «a,b>,c>:(A&B)&C |- c:C      Reduced term : snd < x, y >= y
```

We conclude that the translation from the Natural Deduction to Proof Terms was correctly implemented and it does output a valid proof search in Proof Terms.

7.3.3 Proof Search Evaluation (using combinations of logical connectives)

Proof searches are not normally composed of decompositions of the same type. In this section we evaluate for correctness in proof searches that use different rule variants for decomposition.

Example 7.3.15. RTP - B $\vdash ((A \supset (B \wedge A)) \vee C)$
System's Input : `B |- ((A->(B&A))|C)`

```
    1. a:A, B |- B      Hypothesis – B

    2. a:A, B |- A      Hypothesis – A
 3. a:A, B |- (B&A)     ConjIntro – B, A
 4. B |- (A->(B&A))     ImplIntro – A, (B&A)
 5. B |- ((A->(B&A))|C)     DisjIntroLeft – (A->(B&A))
```

We chose example 7.3.15 because for proof tree composition it applies most of the introduction rules for Natural Deduction. Example 7.3.15 applies the rules *Hypothesis*, $\wedge_I$, $\supset_I$ and $\vee_I$ (ref. to $A.1$) during decomposition. Reading top-down, the proof starts by proving the propositions `A` and `B` which in this case are justified by the *Hypothesis* rule. The two subproofs are then used as premises for the application of the $\wedge_I$ rule. Having the scope `a` for proposition `A` and the succedent proposition `B&A`, using the rule $\supset_I$ the system concludes `A->(B&A)`. Finally, it applies the rule $\vee_I$ on the proposition `A->(B&A)`.

Example 7.3.16. RTP - (A$\wedge(B \supset C)), B \vdash C$
System's Input : `(A&(B->C)), B |- C`

Proof 7.3.16.1 : *Proof Search computed using Method* 1

```
    1. (A&(B->C)), B |- (A&(B->C))      Hypothesis – (A&(B->C))
    2. (A&(B->C)), B |- (B->C)      ConjElimRight – (A&(B->C))

    3. (A&(B->C)), B |- B      Hypothesis – B
 4. (A&(B->C)), B |- C      ImplElim – (B->C), B
```

Proof 7.3.16.2 : *Proof Search computed using Method* 2

```
1. B, (A&(B->C)) |- (A&(B->C))      Hypothesis − (A&(B->C))

        2. B, (A&(B->C)), (B->C) |- (A&(B->C))      Hypothesis − (A&(B->C))
        3. B, (A&(B->C)), (B->C) |- (B->C)      ConjElimRight − (A&(B->C))

        4. B, (A&(B->C)), (B->C) |- B      Hypothesis − B
      5. B, (A&(B->C)), (B->C) |- C      ImplElim − (B->C), B

        6. C, B, (A&(B->C)), (B->C) |- C      Hypothesis − C
   7. B, (A&(B->C)), (B->C) |- C      Substitution
8. B, (A&(B->C)) |- C      Substitution
```

Example 7.3.16 was chosen to show the correct application of the rules $\wedge_E$ and $\supset_E$ (ref. to $A.1$). Reading top down, proof search 7.3.16.1 starts by proving the proposition `(A&(B->C))` by the *Hypothesis* rule. Similarly for line 3. Line 2 is obtained by the application of rule $\wedge_{E_R}$ on line 1. Both sub proofs lines 1 to 2 and 3 respectively are then used as premises for the rule $\supset_E$.

On the other hand, the proof search computed using method 2 (proof 7.3.16.2) is similar to proof 7.3.16.1 but it applies the rule of *Substitution* at the end. This proof search uses the sub proofs defined in lines 2-5 and line 6 as premises to the *Substitution* rule. Similarly, subproofs in line 1 and lines 2-7 are the premises for the *Substitution* rule used to conclude line 8. As described in examples 7.3.4.2 and 7.3.6.2 the application of the substitution rule is required to handle any propositions persisted by the Sequent Calculus rules.

Both examples 7.3.15 and 7.3.16 show that the system computes correct proof terms even when more than one inference rule is required.

7.3.4 Invalid Judgments

Finally, the last aspect to evaluate in terms of proof search correctness is when the passed judgment does not hold hence the system should inform the user when this happens. For these cases we evaluated the system with several test cases for which the system outputs *Judgment does not hold!*

7.4 Proof Search Quality

Other than outputting the correct proof tree we were careful that any computation which resulted in backtracking was eliminated from the final proof search. For instance in example 7.4.1 the system first checks whether an $\wedge_{E_L}$(ref. to $A.1$) is applicable however if this is applied the proof search is derived into a stuck state (line 2) therefore we have to backtrack.

Example 7.4.1. RTP - (A$\wedge B$) $\vdash B$
System's Input : `(A&B)|- B`

Proof 7.4.1.1

`1. (A&B) |- (A&B)` *Hypothesis* $-(A\&B)$
`2. (A&B) |- A` *ConjElimLeft* $-(A\&B)$
`3. (A&B) |- B` *ConjElimRight* $-(A\&B)$

Proof Search showing backtracking

Proof 7.4.1.2

`1. (A&B) |- (A&B)` *Hypothesis* $-(A\&B)$
`2. (A&B) |- B` *ConjElimRight* $-(A\&B)$

Proof Search as shown by the system

We could have opted to leave the decomposition within the proof search because as shown in example 7.4.1 (proof 7.4.1.1) the proof search is still correct. However, in order to keep the proof search concise and readable we opted not to output any backtracking required during proof tree computation (proof 7.4.1.2).

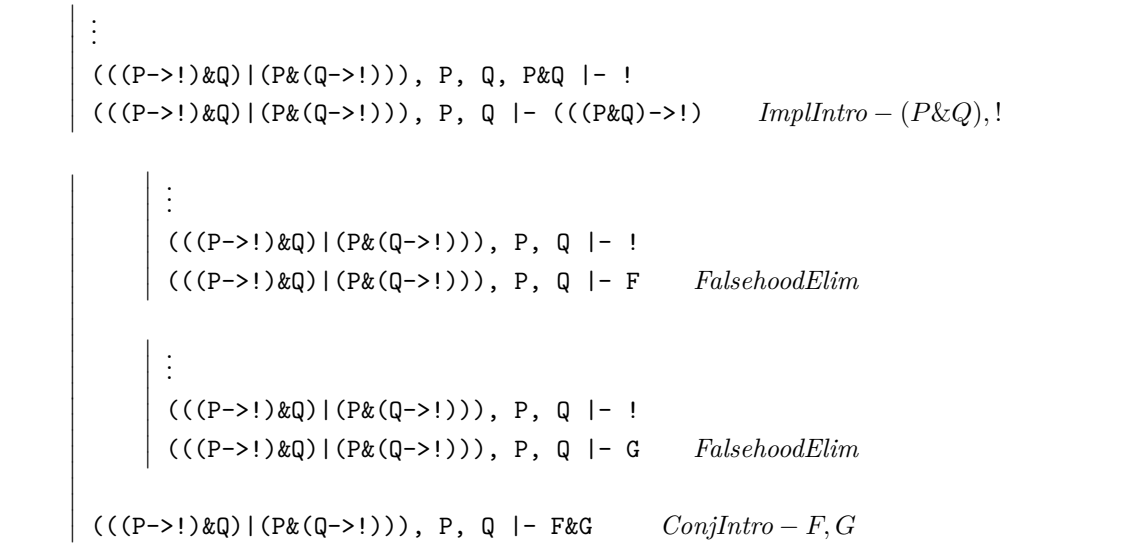
7.5 Stress Testing

In the previous section, we evaluated whether the proof searches outputted by the system are correct or not. We also gave simple proof search examples to prove trivial judgments to sustain the correct computation carried out by the implemented system. In this section we intend to evaluate the system for more complicate and intricate proofs. Our aim is to determine how the system responds to judgments that require demanding computation. For this reason we carefully created provable judgments.

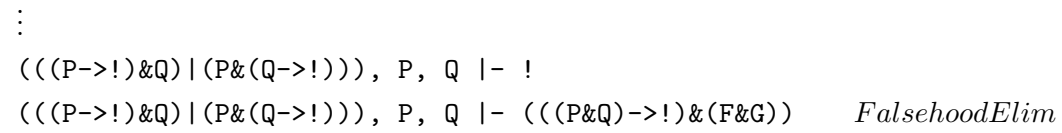
Test Case: (((P$\supset \bot$)$\wedge$Q)$\vee$(P$\wedge$(Q$\supset \bot$))), P, Q $\vdash$ (((P$\wedge$Q)$\supset \bot$)(F$\wedge$G))
System's Input: `(((P->!)&Q)|(P&(Q->!))), P, Q |- (((P&Q)->!)&(F&G))`
Evaluation: exhaustive inference rule application

This test case was specifically chosen because to prove that the judgment holds the system has to apply almost all the inference rules for the Natural Deduction Calculus. The full proof search is listed in the appendices (ref. to *C*) but to show the intricate reasoning behind this proof search we give an brief description how the proof search was computed.

Starting bottom up the search first applies the $\wedge_I$ rule over the propositions `((P&Q)->!)` and `(F&G)`. For both subproofs the system opts to prove `!` ($\bot$) using the $\vee_E$ over the antecedent `(((p->!)&q)|(p&(q->!)))` in both cases. According to which sub proof it is computing the system then concludes the propositions `((p&q)->!)` and `(F&G)` by the rule $\bot_E$.



Although the test shows that the system does output a correct output for judgments that require demanding computation, the system has room for improvement. For instance, in this particular example the system is proving the proposition `!` for three times and for every proof the system is performing the same decomposition. A more efficient way to handle this particular proof was to first proof the proposition `!` then by the rule $\perp_E$ the system concludes `(((P&Q)->!)&(F&G))` as in 7.1.



Test Case: (a∧(a⊃ ⊥))⊃c, a∧(a|c), b⊃c, (((a⊃ ⊥)∧b)⊃(a∧(b⊃ ⊥))), ((b⊃c)⊃a) ⊢ a∧c
System's Input: `(a&(a->!))->c, a&(a|c), b->c, (((a->!)&b)->(a&(b->!))), ((b->c)->a)|- a&c`
Evaluation: Backtracking during proof tree computation

From the analysis carried out on the Intercalation Calculus and the Sequent Calculus we concluded that proof search computed in the Sequent Calculus at no point during runtime it needs to backtrack since propositions are persisted. However, if a proof search computed in the Intercalation Calculus is in a stuck state then the system has to backtrack.

In order to output a concise and yet precise proof search any backtracking performed by the system in case of the Intercalation Calculus theorem prover is not outputted to the user. The output to the user consists only of the correct proof tree that leads to a proof. Hence to check the computation that the system has to perform due to backtracking using the proof search

is not possible. Thus we resort to a tool offered by the WinHugs interpreter (an interpreter for Haskell). We compare the number of reductions per computation. The interpreter replaces variables in a function with other functions until the expression is just a simple expression that can be interpreted [17]. This helps us identify the amount of reductions that were performed during computation. Furthermore, if the system has to backtrack then the amount of reductions will increase. Therefore, we compare the number of reductions computed for the proof search computed in the Intercalation Calculus and the Sequent Calculus to get an idea of the backtracking effect in the system.

Judgment `(a&(a->!))->c, a&(a|c), b->c, (((a->!)&b)->(a&(b->!))), ((b->c)->a)|-a&c` does not hold however the system still has to verify that the judgment does not hold. Before outputting the judgment's verification (it holds or it does not hold) the system tries every possible path. This particular judgment has five antecedents that all have to be checked during runtime. The number of reductions when the proof search is carried out using the Sequent Calculus is *6309* while when it uses the Intercalation Calculus is *8301*. This shows that for the Intercalation Calculus proof search the system has to perform more reductions due to backtracking.

7.6 Reduced Non Determinism from Proof Search

During the analysis carried out in chapter 3 we gave an extensive description of the causes for non determinism in Natural Deduction during proof computation. We concluded that the main causes are:

- During proof search sometimes one needs to make intuitions in order to establish how to prove the succedent.
- Once the system hits a stuck state it has to decide whether to backtrack or else terminate the proof search.

In the same chapter we analysed the Sequent Calculus and concluded that the definition of the inference rules for this calculus guides us how to conduct the proof search without the need to make intuitions during proof computation. Since proof search in the Sequent Calculus is computed using a bottom up technique only, then during proof search some propositions and the list of antecedents are usually retained. This eliminates the possibility of backtracking thus once the proof search hits a stuck state it terminates.

Therefore, a feasible approach to reduce the non determinism in Natural Deduction is to compute the proof in the Sequent Calculus and translate it to Natural Deduction (refer to figure 5.1).

Knowing before we started implementing the system that the Sequent Calculus contains less non determinism in comparison to the Natural Deduction and the Intercalation Calculus, then in this section we only have to evaluate whether the process to translate from one calculus to another introduces any form of non determinism.

7.6.1 Translating from the Intercalation Calculus

When using method 1 to compute a proof search in the Natural Deduction, the system is actually computing the proof tree in the Intercalation Calculus. Due to the similarities between the two calculi, the translation from one calculus to another is fairly simple. Translation follows theorem 3.

Example 7.6.1. RTP - (A$\wedge B) \wedge C \vdash A$
System's Input : `(A&B)|- B`

Proof 7.6.1.1 : *Proof Search computed using the Intercalation Calculus*

1. `((A&B)&C) |- ((A&B)&C)`$\downarrow$ *Hypothesis* $-((A\&B)\&C)$
2. `((A&B)&C) |- (A&B)`$\downarrow$ *ConjElimLeft* $-((A\&B)\&C)$
3. `((A&B)&C) |- A`$\downarrow$ *ConjElimLeft* $-(A\&B)$
4. `((A&B)&C) |- A`$\uparrow$ $\downarrow\uparrow$ - (A&B)

Proof 7.6.1.2 : *Proof Search after translated by Theorem* 2

1. `((A&B)&C) |- ((A&B)&C)` *Hypothesis* $-((A\&B)\&C)$
2. `((A&B)&C) |- (A&B)` *ConjElimLeft* $-((A\&B)\&C)$
3. `((A&B)&C) |- A` *ConjElimLeft* $-(A\&B)$

Hence, for every decomposition in the proof tree, the translation removes the identification of *verifications* and *uses*. Translation is a sequential process thus other to the level of non determinism for the actual proof search computation in the Intercalation Calculus, the translation does not introduce any additional non determinism.

7.6.2 Translating from the Sequent Calculus to the Intercalation Calculus

When using method 2 to compute a proof search in the Natural Deduction, the system is actually computing the proof tree in the Sequent Calculus. The actual proof search is then translated to the Intercalation Calculus then to the Natural Deduction. The latter translation follows Theorem 3. On the other hand, the translation between the Sequent Calculus and the Intercalation Calculus follows Theorem 6.

"Right" Rules

At this point we should distinguish decomposition justified by a "left" rule and others justified by a "right" rule. We already alluded to the fact that the "right" rules persist the same antecedents' list during decomposition therefore when translated by theorem 6 the same justification applies for that calculus. We recall the rule for $\wedge_I$ (or $\wedge_R$) for both the Intercalation Calculus and the Sequent Calculus.

$$\frac{\widehat{\Gamma} \vdash A\uparrow \quad \widehat{\Gamma} \vdash B\uparrow}{\widehat{\Gamma} \vdash A \wedge B\uparrow}\wedge_I \qquad \frac{\Gamma \Rightarrow A \quad \Gamma \Rightarrow B}{\Gamma \Rightarrow A \wedge B}\wedge_R$$

From definitions we notice that after decomposing a judgment using the rule $\wedge_R$ in Sequent Calculus the list of antecedents is retained without any changes. The same can be said if the rule is applied in a proof search computed in the Intercalation Calculus (therefore $\widehat{\Gamma} \equiv \widetilde{\Gamma}$). Hence, when translating from the Sequent Calculus to the Intercalation Calculus using theorem 6, the decomposition can be justified by its equivalent rule in the Intercalation Calculus. This means that the translation is just a sequential process that goes through the proof tree and apply theorem 6 without introducing any non determinism. Example 7.6.2 shows the proof search as computed in the Sequent Calculus and after it is translated to the Intercalation Calculus.

Example 7.6.2. RTP - $\mathrm{B} \vdash ((A \supset (B \wedge A)) \vee C)$
System's Input : `B |- ((A->(B&A))|C)`

Proof 7.6.2.1 : *Proof Search computed using the Sequent Calculus*

`1. A, B ⇒ B` $\quad Init - B$

`2. a:A, B ⇒ A` $\quad Init - A$
`3. a:A, B ⇒ (B&A)` $\quad ConjRight - B, A$
`4. B ⇒ (A->(B&A))` $\quad ImplRight - A, (B\&A)$
`5. B ⇒ ((A->(B&A))|C)` $\quad DisjRight1 - (A\text{->}(B\&A))$

Proof 7.6.2.2 : *Proof Search after translated by Theorem* 6

`1. a:A, B |- B ↓` $\quad Hypothesis - B$
`2. a:A, B |- B ↑` $\quad Hypothesis - B$

`3. a:A, B |- A ↓` $\quad Hypothesis - A$
`4. a:A, B |- A` $\quad Hypothesis - A$
`5. a:A, B |- (B&A)↑` $\quad ConjIntro - B, A$
`6. B |- (A->(B&A))↑` $\quad ImplIntro - A, (B\&A)$
`7. B |- ((A->(B&A))|C)↑` $\quad DisjIntroLeft - (A\text{->}(B\&A))$

"Left" Rules

When decompositions are justified by "left" rules, using theorem 6 for translation would not suffice the equivalent elimination rule in the Intercalation Calculus. We recall the rules

$$\frac{\widetilde{\Gamma}, A \wedge B, A \Rightarrow C}{\widetilde{\Gamma}, A \wedge B \Rightarrow C} \wedge_{L_1} \qquad \frac{\widetilde{\Gamma}, A \wedge B, B \Rightarrow C}{\widetilde{\Gamma}, A \wedge B \Rightarrow C} \wedge_{L_2}$$

$$\frac{\widehat{\Gamma} \vdash A \wedge B \downarrow}{\widehat{\Gamma} \vdash A \downarrow} \wedge_{E_L} \qquad \frac{\widehat{\Gamma} \vdash A \wedge B \downarrow}{\widehat{\Gamma} \vdash B \downarrow} \wedge_{E_R}$$

It is clearly visible that while the application of the rule $\wedge_L$ persists the antecedents' list and add a new proposition (reading bottom up), the equivalent rule in the Intercalation Calculus $\wedge_E$ persists the antecedents' list only ($\widehat{\Gamma} \equiv \widetilde{\Gamma}, A \wedge B$ but $\widehat{\Gamma} \equiv \widetilde{\Gamma}, A \wedge B, \underline{B}$). Therefore, after translating using theorem 6 the judgments cannot be justified by the rule $\wedge_E$. Example 7.6.3 shows the proof search computation in the Sequent Calculus and its translation using theorem 6.

Example 7.6.3. RTP - (A$\wedge B$) $\vdash A$
System's Input : `A&B|- A`

Proof 7.6.3.1 : *Proof Search computed using the Sequent Calculus*

`1. A, (A&B)` $\Rightarrow$ `A` $\quad Init - A$
`2. (A&B)`$\Rightarrow$ `A` $\quad ConjLeft1 - (A\&B)$

Proof 7.6.3.2 : *Proof Search after translated by Theorem* 6

`1. A, (A&B) |- A`$\uparrow$?
`2. (A&B) |- A`$\uparrow$?

To solve this problem we apply theorem 5; the Substitution Theorem. Therefore after the application of theorem 6 we apply theorem 5 for these decompositions. Example 7.6.4 shows the correct translation for example 7.6.3.

Example 7.6.4. RTP - (A$\wedge B$) $\vdash A$
System's Input : `A&B|- A`

Proof 7.6.4.1 : *Proof Search after translated by Theorems* 6 *and* 5

`1. (A&B) |- (A&B)`$\downarrow$ $\quad Hypothesis - (A\&B)$
`2. (A&B) |- A`$\downarrow$ $\quad ConjElimLeft - (A\&B)$

`3. A, (A&B) |- A`$\downarrow$ $\quad Hypothesis - A$
`4. A, (A&B) |- A`$\uparrow$ $\quad \downarrow\uparrow$
`5. (A&B) |- A`$\uparrow$ $\quad Substitution$

We should be careful when evaluating proofs that resort to the Substitution Theorem. The premise $\widehat{\Gamma} \vdash A \downarrow$ for the Substitution Theorem has to be computed directly in the Intercalation Calculus because if we had to compute it in the Sequent Calculus and translate it using theorem 6 we would have obtained $\widehat{\Gamma} \vdash A \uparrow$ and there are no rules that change from a *verification* $\downarrow$ to a *use* $\uparrow$ (reading top down). The first impression is that the system will introduce non determinism during translation however it is not the case. We evaluate this using example 7.6.5.

Example 7.6.5. RTP - $((\mathrm{A} \wedge B) \wedge C) \vdash A$
System's Input : `((A&B)&C) |- A)`

Proof 7.6.5.1 : *Proof Search computed using the Sequent Calculus*

`1. ((A&B)&C), (A&B), A |- A` *Init*	$\dfrac{}{((A \wedge B) \wedge C), (A \wedge B), A \vdash A}^{Init}$
`2. ((A&B)&C), (A&B) |- A` *ConjLeft1*	$\dfrac{((A \wedge B) \wedge C), (A \wedge B) \vdash A}{((A \wedge B) \wedge C), (A \wedge B) \vdash A}^{\wedge_L}$
`3. ((A&B)&C) |- A` *ConjLeft1*	$\dfrac{((A \wedge B) \wedge C) \vdash A}{((A \wedge B) \wedge C) \vdash A}^{\wedge_L}$

Proof 7.6.5.2 : *Proof Search after translated by Theorems* 6 *and* 5

`1. ((A&B)&C) |- ((A&B)&C)↓` *Hyp*	$\dfrac{}{((A \wedge B) \wedge C) \vdash ((A \wedge B) \wedge C)\downarrow}^{hyp}$
`2. ((A&B)&C) |- (A&B)↓` *ConjElimLeft*	$\dfrac{((A \wedge B) \wedge C) \vdash ((A \wedge B) \wedge C)\downarrow}{((A \wedge B) \wedge C) \vdash (A \wedge B)\downarrow}^{\wedge_E}$
`3. ((A&B)&C),(A&B) |- (A&B)↓` *Hyp*	$\dfrac{}{((A \wedge B) \wedge C), (A \wedge B) \vdash (A \wedge B)\downarrow}^{hyp}$
`4. ((A&B)&C),(A&B) |- A↓` *ConjElimLeft*	$\dfrac{((A \wedge B) \wedge C), (A \wedge B) \vdash (A \wedge B)\downarrow}{((A \wedge B) \wedge C), (A \wedge B) \vdash A\downarrow}^{\wedge_E}$
`5. ((A&B)&C),(A&B),A |- A↓` *Hyp*	$\dfrac{}{((A \wedge B) \wedge C), (A \wedge B), A \vdash A\downarrow}^{hyp}$
`6. ((A&B)&C),(A&B),A |- A↑` ↓↑	$\dfrac{((A \wedge B) \wedge C), (A \wedge B), A \vdash A\downarrow}{((A \wedge B) \wedge C), (A \wedge B), A \vdash A\uparrow}^{\downarrow\uparrow}$
`7. ((A&B)&C),(A&B) |- A↑` *Subs*	$\dfrac{((A \wedge B) \wedge C), (A \wedge B), A \vdash A\uparrow}{((A \wedge B) \wedge C), (A \wedge B) \vdash A\uparrow}^{Subs}$
`8. ((A&B)&C) |- A↑` *Subs*	$\dfrac{((A \wedge B) \wedge C), (A \wedge B) \vdash A\uparrow}{((A \wedge B) \wedge C) \vdash A\uparrow}^{Subs}$

Example 7.6.6. RTP - $((\mathrm{A} \wedge B) \wedge C) \vdash A$
System's Input : `((A&B)&C) |- A)`

Proof 7.6.6.1 : *Proof Search computed using the Intercalation Calculus*

`1. ((A&B)&C) |- ((A&B)&C)↓` *Hypothesis*	$\dfrac{}{((A \wedge B) \wedge C) \vdash ((A \wedge B) \wedge C)\downarrow}^{hyp}$
`2. ((A&B)&C) |- (A&B)↓` *ConjElimLeft*	$\dfrac{((A \wedge B) \wedge C) \vdash ((A \wedge B) \wedge C)\downarrow}{((A \wedge B) \wedge C) \vdash (A \wedge B)\downarrow}^{\wedge_E}$
`3. ((A&B)&C) |- A↓` *ConjElimLeft*	$\dfrac{((A \wedge B) \wedge C) \vdash (A \wedge B)\downarrow}{((A \wedge B) \wedge C) \vdash A\downarrow}^{\wedge_E}$
`4. ((A&B)&C) |- A↑` ↓↑	$\dfrac{((A \wedge B) \wedge C) \vdash A\downarrow}{((A \wedge B) \wedge C) \vdash A\uparrow}^{\downarrow\uparrow}$

Proof 7.6.5.1 shows the proof search in the Sequent Calculus. Proof 7.6.5.2 shows proof 7.6.5.1 after it was translated to the Intercalation Calculus while proof 7.6.6.1 shows the computation in the Intercalation Calculus. We start our evaluation by the analysis for proof 7.6.6.1. We already defined that one of the non deterministic factors for Natural Deduction and Intercalation Calculus is the need to make intuitions during proof tree composition. Referring to proof 7.6.6.1 lines 2 and 3, the system could not apply any rule on line 3 (reading bottom up) to justify the proposition A knowing only *((A&B)&C)*. Therefore, the system had to make the intuition that in order to prove A it must first prove *(A&B)* then prove *((A&B)&C)* which is an antecedent.

We now evaluate proof 7.6.5.1 reading bottom up. The decomposition between line 3 and 2 for this rule is justified by the $\wedge_L$ (denoted as *ConjElim1*). After the application of theorem 6 the resultant decomposition (full translation shown in proof 7.6.5.2) cannot be justified by the rule $\wedge_E$ since the list of antecedents for the premise and the succedent differ (shown in pink). Thus we apply theorem 5 as shown in line 8 in proof 7.6.5.2. The Substitution Theorem requires the premises `((A&B)&C) |- (A&B)`$\downarrow$ and `((A&B)&C), (A&B) |- A`$\uparrow$ which subproofs shown in lines 1-2 and lines 3-6 in proof 7.6.5.1.

As defined by the Substitution Theorem the subproof in lines 1-2 should be computed in the Intercalation Calculus. Our concern is whether the computation for lines 1-2 will introduce any non determinism. Although the computation is carried out in the Intercalation Calculus, it does not introduce any non determinism. We know that the rules' definition for the Sequent Calculus guides us during proof search hence eliminating the need to make intuitions during computation. This fact is reflected in the translation. Thus for the sub proof rather than proving the judgment `((A&B)&C) |- A`, the system guides us to first prove the judgment `((A&B)&C), (A&B) |- A`$\uparrow$ which was the intuition needed in proof 7.6.5.1.

The sub proof for lines 3-6 is translated directly from the proof computed in Sequent Calculus using theorem 6. This does not mean that during the translation it does not resort to the Substitution Theorem to resolve any differences. This is the case for the sub proof in lines 3-7. Similarly, to the lines 1-2 the system proves the premise `((A&B)&C), (A&B) |- A`$\downarrow$ (lines 3-4) for the substitution theorem using the Intercalation Calculus, while the second premise (`A, ((A&B)&C), (A&B) |- A`$\uparrow$ - lines 5-6) for the Substitution Theorem is translated directly from the proof search computed in the Sequent Calculus that is proof 7.6.5.1 line 1.

Therefore we conclude that although we are applying the Substitution Theorem and resorting to the Intercalation Calculus for some computation, the proof 7.6.5.2 still includes less non determinism in comparison to prove 7.6.6.1 (which was computed in the Intercalation Calculus) because at no point during proof search the system has to make any intuitions. Finally, the information how the Substitution Theorem is applied and how non determinism is reduced in this manner applies for all the other "left" rules.

7.7 Performance

From the initial steps of this project our main concept was to provide a system that reduces non determinism for proof search computed in the Natural Deduction. Neither during design phase nor during the implementation phase we analysed how the system can have a better performance in terms of efficiency and computation speed. Having said that we knew from the calculi analysis carried out in Section 3.5 that the system would compute proof search faster if computed in the Sequent Calculus rather than in the Intercalation Calculus due to a higher level of non determinism in the Intercalation Calculus. In this section we intend to compare the output produced by the system using methods 1 and 2. We compare on the basis of proof search length and the amount of computations.

Proof Search Length

The length of the proof search for the same judgment computed by method 1 and 2 differs according to which rules are used for decomposition. For instance, the Sequent Calculus' "right" rules (ref to fig $A.3$) are very similar to the Intercalation Calculus' introduction rules which in turn are very similar to the Natural Deductions' introduction rules. (7.1) shows the "right" rule and the introduction rules for the conjunction logic connective.

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B} \wedge_I \qquad \frac{\widehat{\Gamma} \vdash A\uparrow \quad \widehat{\Gamma} \vdash B\uparrow}{\widehat{\Gamma} \vdash A \wedge B\uparrow} \wedge_I \qquad \frac{\widetilde{\Gamma} \Rightarrow A \quad \widetilde{\Gamma} \Rightarrow B}{\widetilde{\Gamma} \Rightarrow A \wedge B} \wedge_R \tag{7.1}$$

We can clearly notice that other than the symbols which define the calculus used ($\Rightarrow$ = Sequent Calculus, $\uparrow\downarrow$ = Intercalation Calculus) the rules are identical. Hence when the system translates from one calculus to another using theorems 3 (Intercalation Calculus to Natural Deduction) and 6 (Sequent Calculus to Intercalation Calculus), the proof search does not change, other than the corresponding rule. Example 7.7.1 shows the proof computation in the Intercalation Calculus and translated to the Natural Deduction while example 7.7.2 show the proof computation in the Sequent Calculus translated to the Intercalation Calculus then to Natural Deduction.

Example 7.7.1. RTP - A, B $\vdash (A \wedge B)$
System's Input : `A, B |- A&B`

Proof 7.7.1.1

`1. A, B |- A↓` *Hypothesis* $- A$
`2. A, B |- A↑` *LeftRight* $- A$

`3. A, B |- B↓` *Hypothesis* $- B$
`4. A, B |- B↑` *LeftRight* $- B$
`5. A, B |- (A&B)↑` *ConjIntro* $- A, B$

Proof search – Intercalation Calculus

Proof 7.7.1.2

`1. A, B |- A` *Hypothesis* $- A$

`2. A, B |- B` *Hypothesis* $- B$
`3. A, B |- (A&B)` *Conjintro* $- A, B$

Proof search translated to the Natural Deduction

Example 7.7.2. RTP - A, B $\vdash (A \wedge B)$
System's Input : `A, B |- A&B`

Proof 7.7.2.1

```
1. A, B => A      Init − A

2. A, B => B      Init − B
3. A, B => (A&B)      ConjRight − A, B
```

Proof search computed in the Sequent Calculus

Proof 7.7.2.2

```
1. A, B |- A      Hypothesis − A

2. A, B |- B      Hypothesis − B
3. A, B |- (A&B)      Conjintro − A, B
```

Proof search translated to the Natural Deduction

On the other hand, if during computation the elimination rules (for the Intercalation Calculus) and "left" rules (for the Sequent Calculus) are used then the length of the final proof searches in Natural Deduction will differ. (7.2) and (7.3) shows the rule to extract propositions from a conjunction and an implication respectively.

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A} \wedge_{E_L} \qquad \frac{\widehat{\Gamma} \vdash A \wedge B \downarrow}{\widehat{\Gamma} \vdash A \downarrow} \wedge_{E_L} \qquad \frac{\widetilde{\Gamma}, A \wedge B, A \Rightarrow C}{\widetilde{\Gamma}, A \wedge B \Rightarrow C} \wedge_{L_1} \tag{7.2}$$

$$\frac{\Gamma \vdash A \supset B \quad \Gamma \vdash A}{\Gamma \vdash B} \supset_E \qquad \frac{\widehat{\Gamma} \vdash A \supset B \downarrow \quad \widehat{\Gamma} \vdash A \uparrow}{\widehat{\Gamma} \vdash B \downarrow} \supset_E \qquad \frac{\widetilde{\Gamma}, A \supset B \Rightarrow A \quad \widetilde{\Gamma}, A \supset B, B \Rightarrow C}{\widetilde{\Gamma}, A \supset B \Rightarrow C} \supset_L \tag{7.3}$$

We can clearly notice that the rules for the Sequent Calculus retain propositions during decomposition. Therefore, translating the proof search using Thereom 6 would not suffice the proof search to justify the decomposition with equivalent rules for the Intercalation Calculus. Thus, we need to first translate the proof search to the Intercalation Calculus then apply theorem 5 to resolve any differences. Automatically, this lengthens the proof search for method 2. Example 7.7.3 shows the proof search computation in the Sequent Calculus and its equivalent translated to the Natural Deduction as outputted by the system.

Example 7.7.3. RTP - $\vdash A \wedge (B \wedge C) \supset C$
System's Input : `|- A&B&C -> C`

Proof 7.7.3.1

```
1. (A&(B&C)), (B&C), C => C      Init − C
2. (A&(B&C)), (B&C) => C      ConjLeft2 − (B&C)
3. (A&(B&C)) => C      ConjLeft2 − (A&(B&C))
```

Proof search computed in the Sequent Calculus

Proof 7.7.3.2

```
1. (A&(B&C)) |- (A&(B&C))      Hypothesis − (A&(B&C))

  2. (A&(B&C)), (B&C) |- (B&C)      Hypothesis − B&C
  3. (A&(B&C)), (B&C) |- C      ConjElimRight − (A&(B&C))

  4. (A&(B&C)), (B&C), C |- C      Hypothesis − C
5. (A&(B&C)), (B&C) |- C      Substitution
6. (A&(B&C)) |- C      Substitution
```

Proof search translated to the Natural Deduction

Furthermore although the rules $\vee_L$ and $\bot_L$ does not persist any additional propositions other than $\widetilde{\Gamma}$ during decomposition, translating from the Sequent Calculus to the Intercalation Calculus using theorem 6 would not suffice. Both rules require another premise to have a successful translation therefore during translation the required premise is proved and combined to the proof search. Examples 7.7.4 and 7.7.5 show the proof search for the judgment `(A|B), ! |- C` both in method 1 and 2 and their respective proof search in the actual calculus. We notice for example 7.7.5 that proof translated to Natural Deduction contains addition judgments in comparison to the actual proof search in the Sequent Calculus. This shows what we have argued above, that is, sometimes during translation additional judgments are proved to produce a correct proof in the other calculus (in this case the Natural Deduction).

Example 7.7.4. RTP - A$\vee B, \bot \vdash C$
System's Input : `(A|B), ! |- C`

Proof 7.7.4.1

```
1. (A|B), ! |- (A|B)↓      Hypothesis − (A|B)

  2. (A|B), !, a:A |- !↓      Hypothesis − C
  3. (A|B), !, a:A |- C↑      FalsehoodElim

  4. (A|B), !, b:B |- !↓      Hypothesis − C
  5. (A|B), !, b:B |- C↑      FalsehoodElim
6. (A|B), ! |- C↑      DisjElim − (A|B)
```

Proof search computed in Int. Cal.

Proof 7.7.4.2

```
1. (A|B), ! |- (A|B)      Hypothesis − (A|B)

  2. (A|B), !, a:A |- !      Hypothesis − C
  3. (A|B), !, a:A |- C      FalsehoodElim

  4. (A|B), !, b:B |- !      Hypothesis − C
  5. (A|B), !, b:B |- C      FalsehoodElim
6. (A|B), ! |- C      DisjElim − (A|B)
```

Proof search translated to Nat. Ded.

Example 7.7.5. RTP - A$\vee B, \bot \vdash C$
System's Input : `(A|B), !  |- C`

Proof 7.7.5.1

```
1. (A|B), !, A => C     FalsehoodLeft
2. (A|B), !, B => C     FalsehoodLeft
3. (A|B), !  => C       DisjElim − (A|B)
```

Proof search computed in the Seq. Cal.

Proof 7.7.5.2

```
1. (A|B), !  |- (A|B)      Hypothesis − (A|B)
2. (A|B), !, a:A |- !      Hypothesis − C
3. (A|B), !, a:A |- C      FalsehoodElim
4. (A|B), !, b:B |- !      Hypothesis − C
5. (A|B), !, b:B |- C      FalsehoodElim
6. (A|B), !  |- C          DisjElim − (A|B)
```

Proof search translated to Nat. Ded.

Finally, we conclude that the length of a proof search depends on which method (method 1 or 2) is used for the proof search in the Natural Deduction. Although the proof search computed in the Sequent Calculus then translated to the Natural Deduction tends to be longer to its equivalent in the Intercalation Calculus, during computation the proof search requires less non deterministic choices and it still produces a correct proof tree.

Computation Time

When we started evaluating the system in terms of computation time we opted to study the system's performance in terms of reductions performed by the system at runtime. The number of reductions imply the number of times the system had to replace a function with another simpler function until the expression does not require anymore reductions. Furthermore, inference rules are implemented in terms of functions therefore the number of reductions guides us through performance evaluation for different proving methodologies implemented for this system. To compare different methodologies we chose a set of test cases, tested each test case using different calculi and studied the computation time of the set of test cases in one calculus in relation to other calculi.

From the analysis carried out in Chapter 3 we concluded that proof search computed in the Sequent Calculus contains less non deterministic choices in comparison to another computed in the Intercalation Calculus. The presented system contains the implementation of the Intercalation Calculus and the Sequent Calculus theorem provers. Hence, we start by evaluating the performance for the automated theorem provers. Figure 7.7 shows the results for the same set of test cases first tested using the Intercalation Calculus theorem prover then the Sequent Calculus theorem prover. It clearly shows the computation time difference between the two calculi. The set of test cases tested confirms that due to less non determinism in the Sequent Calculus proof

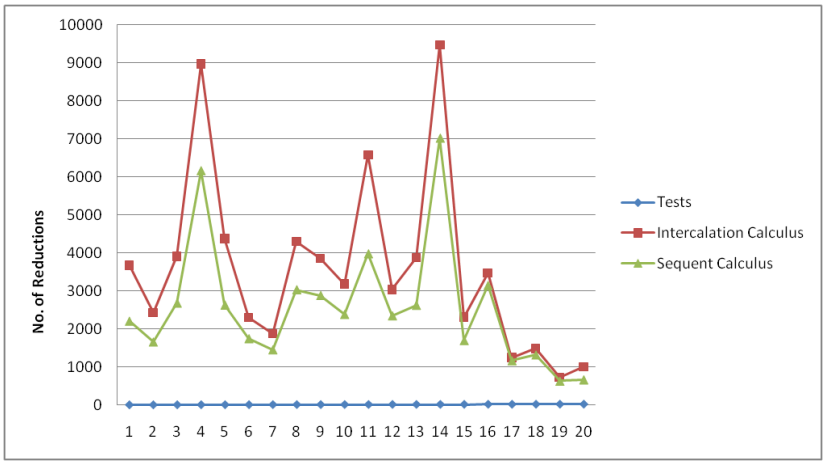


Figure 7.1: Graph of Number of Reductions against Test Cases for the Intercalation Calculus and Sequent Calculus

search system's performance is better. On the other hand, the non deterministic nature of the Intercalation Calculus, which is usually handled by backtracking the proof search if the correct choice is not chosen at runtime, results in a less efficient performance.

After evaluating the theorem provers' performance we then opted to evaluate the system's performance for the Natural Deduction proof searches in relation to methods 1 and 2. We recall that in order to minimise non determinism during proof search for the Natural Deduction calculus we can either compute the proof search in the Intercalation Calculus then translate it to the Natural Deduction (Method 1) or else compute the proof search in the Sequent Calculus translate it to the Intercalation Calculus then to Natural Deduction (Method 2). Figure 7.7 shows the

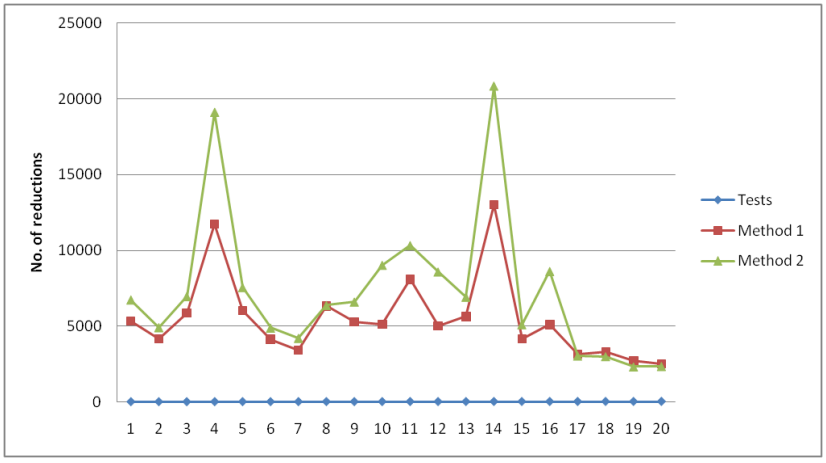


Figure 7.2: Graph of Number of Reductions against Test Cases for methods 1 and 2

results for the same set of test cases first tested using Methods 1 and 2. The plotted graph show that in general although method 2 reduced the non determinism during proof tree computation, it takes more time to output the proof search in Natural Deduction in comparison to method 1. We have to keep in mind, that other than computing the proof search the system has to perform two translations for method 2. Moreover, decompositions might persist their propositions during

proof search in the Sequent Calculus. Thus, to obtain a valid translation from the Sequent Calculus to the Natural Deduction the system might apply the Substitution rule which by definition it requires further computation. On the other hand, for proof searches computed using method 1 the system has to perform a translation only once, from the Intercalation Calculus to the Sequent Calculus. That being said, one have to keep in mind, that the translation process is sequential therefore if the computation contains alot of non determinism for method 1 in comparison to method 2 then the amount of backtracking required will outweigh the sequential translation process for method 2. Thus, in such cases method 2 will perform better.

One point worth mentioning at this time is the results for test cases 17-20. Figure 7.7 shows that for these cases method 2 performs better than method 1. Each input judgment for these tests does not hold for which the automated theorem prover outputs an empty proof tree. Hence the number of translations required to achieve the proof tree in Natural Deduction does not matter as empty proof trees are translated to other empty proof trees. This strengthens the argument that proof search computed and outputted in the Sequent Calculus performs better than proof search computed and outputted in the Intercalation Calculus. It also points out that due to the number of translations required, for some cases, the proof search computed using method 2 is not as efficient as its equivalent computed using method 1.

7.8 Conclusion

Although the system contains limitations and bugs which have already been explained and listed in this document, overall it does produce proof searches which contain less non determinism. The proofs computed can sometimes be quite lengthy nonetheless precise. The facility to output proofs in more than one calculus made it easier for us to compare and contrast the calculi together. Moreover, the production of a lambda term for a specific proof tree gave an opportunity to study the relation between proofs and programs where the latter are just a syntax representation of the former.

8. Conclusion

To conclude, we give a brief description of what this project was about, what it set to achieve and whether this was successful. The system presented consists of a verifier and an automated theorem prover for Natural Deduction. For the verifier the user enter the judgment to be verified and the system outputs whether it holds or not. On the other hand, for proof search automation the system offers three options:

- Method 1: Natural Deduction - Proof search computed in the Intercalation Calculus and translated to the Natural Deduction,
- Method 2: Natural Deduction - Proof search computed in the Sequent Calculus, translated to the Intercalation Calculus then to the Natural Deduction.
- Method 3: Proof Terms - A Natural Deduction proof search translated to Proof Terms.

In Chapter 1 we set our aims to:

- Provide a system which better automates better the proof search in the Natural Deduction
- Provide correct proof searches

To conclude we analyse analyse whether our aims have been achieved.

8.1 Reduce Non Determinism from Proof Search

One of the main aims of this system was to reduce the non determinism during proof search in Natural Deduction. In Chapter 3, particularly Section 3.1 we analysed the problems that causes non determinism in this calculus for which we concluded that the definition of the inference rules for the Natural Deduction causes most of the non determinism during proof tree computation. We highlighted the fact that due to the simplistic way the inference rules for this calculus are defined, during proof searching one has to make some intuitions in order to derive the conclusion.

In the same chapter but in Section 3.3 we analysed the Sequent Calculus where we concluded that by the definition of its inference rules and the way proof search proceeds (proceeding from the conclusion to the premises - bottom up technique) non determinism is reduced substantially. Furthermore, using the latter methodology, we pointed out that few are the cases when the system has to make an intuition rather than decompose using an inference rule.

Thus we concluded that a feasible way to reduce the non determinism from proof searching in the Natural Deduction is to compute it in the Sequent Calculus then translate it to the Natural Deduction. This was analysed in chapter 5 section 5.2 where we concluded that for a Sequent Calculus proof search to be translated to the Natural Deduction it should first be translated to the Intercalation Calculus then to the Natural Deduction as defined in figure 8.1.

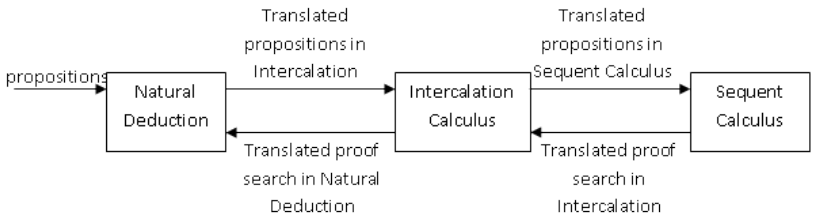


Figure 8.1: Graphical Representation of the system to reduce non determinism during proof search in Natural Deduction.

Subsequently, we evaluated whether this approach was correct. During evaluation we concluded that knowing before hand that the Sequent Calculus contains less non determinism then we only had to evaluate whether the process to translate from one calculus to another introduces any form of non determinism. But since translation is based on the actual proof search and every judgment is translated according to the rules that justifies the judgment then the translation is just simple computation. At this point, we pointed out the difference between translating judgments justified by the "left" and "right" rules in Sequent Calculus. We concluded that for judgments justified by the "left" rules the translation is purely change of notation with no addition computation required. However, although judgments justified by "right" rules may require further computation once translated to the Intercalation Calculus for a successful translation, evaluation shows us that in comparison to a proof search computed in the Intercalation Calculus then translated to Natural Deduction calculus the system presented still contains less non determinism during proof search.

Moreover, performance wise when we compared the two variants of proof search computation methods (method 1 and 2) presented in this system we concluded that the first methodology performs better for test cases that require few computations. Testing results showed us that since a proof search computed in the Sequent Calculus has to be translated twice in order to achieve its equivalent in the Natural Deduction then it requires more reductions in comparison to a proof search computed in the Intercalation Calculus and translated to Natural Deduction. However, the translation process is a sequential process. Therefore it affects case which require few computation steps. Once computations get larger in scale, the non determinism and the required backtracking for method 1 outweigh the sequential process for translation. In such cases, method 2 performs better. Nonetheless, this does not affect the level of non determinism reduced during proof search. Thus, we conclude that the approach to compute the proof search in the Sequent Calculus then translate it to the Natural Deduction did reduce non determinism during proof search.

8.2 Proof Search Correctness

Another important aim for this project was to output correct proof searches. Actually the system consists of two automated theorem provers while the others are translators. For our system, it is sufficient to prove judgments using the Sequent Calculus and the Intercalation Calculus. We

had to ensure that both automated theorem provers produce a correct proof.

Section 7.3 shows the evaluation for proof correctness. We handled correctness in terms of Natural Deduction for which we considered every rule in the Natural Deduction and compared the output by methods 1 and 2 to check whether the produced proof search is correct. We could have checked that the actual proof search computed in the respective calculus is correct rather than checking the proof search after translation for the simple reason that there exist a correspondence between calculi which does not alter the reasoning of the proof search. Thus, having a correct proof in either the Intercalation Calculus or the Sequent Calculus then we are guaranteed that the proof search in the Natural Deduction is correct.

We opted to proof check after translations are computed to be able to compare the output from both method 1 and 2. In fact we conclude that the proof search computed using method 2 may be longer in comparison to that computed using method 1. Still evaluation shows proof correctness for all the methodologies.

8.3 Future Work

There are several ideas that due to time constraints we were not capable to add to this system. Some of these as discussed briefly in this section.

8.3.1 Loop Checking module

There can be cases when the system proves the same succedent twice. Howe [11] suggests that to ensure a theorem prover does not prove the same proposition twice one must implement a history mechanism. This ideas is based on the addition of another data structure (in addition to the proof tree) which keeps record of all the succedents occurred so far. Before the application of a decomposition the system checks whether there was already an attempt to proof that succedent, if so then the system backtracks otherwise it applies an appropriate inference rule [11].

8.3.2 More restrictive Sequent Calculus

Since antecedents are persisted in Sequent Calculus the list of antecedents extends with every decomposition during proof search. The disadvantage is that rules can be applied to the same antecedents during proof search without making any progress. Although the system takes this into consideration and applies appropriate handling, a future implementation can be to restrict the Sequent Calculus. Pfenning[20] defines a new calculus (an extension of the Sequent Calculus) which inverts inference rules for the same connective into one inference rule [20]. For instance the rules $\wedge_{L_1}$ and $\wedge_{L_2}$ (ref to figure) are inverted into one rule as

$$\frac{\Gamma, A, B \Rightarrow C}{\Gamma, A \wedge B \Rightarrow C} \wedge_L$$

The inversion of these rules shows us that the antecedent is not persisted. Although the antecedent list was extended the problem of applying the same rule to the same antecedent is

eliminated. Furthermore, in order to eliminate the choice of whether to apply a left or a right rule Pfenning[20] introduces invertible rules for which he defines a new set of "left" and "right" inference rules which define when we should apply a "right" or "left" rule. The implementation of this restriction decreases the amount of storage required to store the antecedents list and minimises loop checking.

8.3.3 Extending the system to include Predicate Logic Proof Searching

Our main goal was to investigate proof searches in Intuitionistic Propositional Logic. However the system lacks the means to proof judgments about quantifiers. The introduction of $\forall$ (*for all*) and $\exists$ (*there exists*) will enable the system to proof judgments about quantified propositions. It will also give a new dimension to the system; rather than proving judgments on facts, with the introduction of predicate logic the system will be capable to prove judgments on structures [13]. For instance the system accepts propositions of the type

"A dog has four legs"

but with the quantifiers introduction the system will allow propositions of the type

"All animals of type dog have four legs"

8.3.4 Interface Improvement

Due to time constraints we did not dedicate alot of time to the development of the user interface. The introduction of a graphical user interface will help the user to navigate better through the options offered by the system. Moreover, the implementation of GUI will be a better tool for us developers to use to represent the scopes and proof search in a better structural way.

A. Inference Rules

A.1 Natural Deduction

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B}\wedge_I \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A}\wedge_{E_L} \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}\wedge_{E_R}$$

$$\frac{\Gamma, u : A \vdash B}{\Gamma \vdash A \supset B}\supset^{I^u} \qquad \frac{\Gamma \vdash A \supset B \quad \Gamma \vdash A}{\Gamma \vdash B}\supset_E$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B}\vee_{I_L} \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}\vee_{I_R} \qquad \frac{\Gamma \vdash A \vee B \quad \Gamma, u : A \vdash C \quad \Gamma, v : B \vdash C}{\Gamma \vdash C}\vee_{E^{u,v}}$$

$$\frac{}{\Gamma \vdash \top}\top_I \qquad \frac{\Gamma \vdash \bot}{\Gamma \vdash C}\bot_E \qquad \frac{}{\Gamma, A \vdash A}hyp$$

A.2 Intercalation Calculus

$$\frac{\Gamma \vdash A\uparrow \quad \Gamma \vdash B\uparrow}{\Gamma \vdash A \wedge B\uparrow}\wedge_I \qquad \frac{\Gamma \vdash A \wedge B\downarrow}{\Gamma \vdash A\downarrow}\wedge_{E_L} \qquad \frac{\Gamma \vdash A \wedge B\downarrow}{\Gamma \vdash B\downarrow}\wedge_{E_R}$$

$$\frac{\Gamma, u : A\downarrow \vdash B\uparrow}{\Gamma \vdash A \supset B\uparrow}\supset^{I^u} \qquad \frac{\Gamma \vdash A \supset B\downarrow \quad \Gamma \vdash A\uparrow}{\Gamma \vdash B\downarrow}\supset_E$$

$$\frac{\Gamma \vdash A\uparrow}{\Gamma \vdash A \vee B\uparrow}\vee_{I_L} \qquad \frac{\Gamma \vdash B\uparrow}{\Gamma \vdash A \vee B\uparrow}\vee_{I_R} \qquad \frac{\Gamma \vdash A \vee B\downarrow \quad \Gamma, u : A\downarrow \vdash C\uparrow \quad \Gamma, v : B\downarrow \vdash C\uparrow}{\Gamma \vdash C\uparrow}\vee_{E^{u,v}}$$

$$\frac{}{\Gamma \vdash \top\uparrow}\top_I \qquad \frac{\Gamma \vdash \bot\downarrow}{\Gamma \vdash C\uparrow}\bot_E \qquad \frac{}{\Gamma, A \vdash A\downarrow}hyp$$

A.3 Sequent Calculus

$$\frac{\Gamma, A \wedge B, A \Rightarrow C}{\Gamma, A \wedge B \Rightarrow C}\wedge_{L_1} \qquad \frac{\Gamma, A \wedge B, B \Rightarrow C}{\Gamma, A \wedge B \Rightarrow C}\wedge_{L_2} \qquad \frac{\Gamma \Rightarrow A \quad \Gamma \Rightarrow B}{\Gamma \Rightarrow A \wedge B}\wedge_R$$

$$\frac{\Gamma, A \supset B \Rightarrow A \quad \Gamma, A \supset B, B \Rightarrow C}{\Gamma, A \supset B \Rightarrow C}\supset_L \qquad \frac{\Gamma, A \Rightarrow B}{\Gamma \Rightarrow A \supset B}\supset_R$$

$$\frac{\Gamma \Rightarrow A}{\Gamma \Rightarrow A \vee B}\vee_{R_1} \qquad \frac{\Gamma \Rightarrow B}{\Gamma \Rightarrow A \vee B}\vee_{R_2} \qquad \frac{\Gamma, A \vee B, A \Rightarrow C \quad \Gamma, A \vee B, B \Rightarrow C}{\Gamma, A \vee B \Rightarrow C}\vee_L$$

$$\frac{}{\Gamma \Rightarrow \top}\top_R \qquad \frac{}{\Gamma, \bot \Rightarrow C}\bot_L \qquad \frac{}{\Gamma, A \Rightarrow A}init$$

A.4 Proof Terms

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash N : B}{\Gamma \vdash \langle M, N \rangle : A \wedge B}\wedge_I \qquad \frac{\Gamma \vdash M : A \wedge B}{\Gamma \vdash \pi_1 M : A}\wedge_{E_L} \quad \frac{\Gamma \vdash M : A \wedge B}{\Gamma \vdash \pi_2 M : B}\wedge_{E_R}$$

$$\frac{\Gamma, u : A \vdash M : B}{\Gamma \vdash \lambda u : A.M : A \supset B}\supset_I \qquad \frac{\Gamma \vdash M : A \supset B \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B}\supset_E$$

$$\frac{\Gamma \vdash M : A}{\Gamma \vdash \mathbf{inl}^B M : A \vee B}\vee_{I_L} \quad \frac{\Gamma \vdash N : B}{\Gamma \vdash \mathbf{inr}^A N : A \vee B}\vee_{I_R}$$

$$\frac{\Gamma \vdash M : A \vee B \qquad \Gamma, u : A \vdash N : C \qquad \Gamma, v : B \vdash O : C}{\Gamma \vdash \mathbf{case}\ M\ \mathbf{of\ inl}\ u \supset N | \mathbf{inr}\ v \supset O : C}\vee_{E^{u,v}}$$

$$\frac{}{\Gamma \vdash \langle\rangle : \top}\top_I \qquad \frac{\Gamma \vdash M : \bot}{\Gamma \vdash \mathbf{abort}^C\ M : C}\bot_E \qquad \frac{}{\Gamma_1, x : A, \Gamma_2 \vdash x : A}hyp$$

B. Proof for Theorem 6

Theorem 6. (Sequent Calculus to Intercalation Calculus)
If $\Gamma \vdash C\ right\ then\ \widehat{\Gamma} \vdash C \uparrow$.[24]

Proof. The proof is by structural induction on the given derivation.[24]

Base Cases:
Case:

$$\frac{}{\Gamma, A\ left\ \vdash A\ right}\ init$$

RTP: $\widehat{\Gamma}, A \downarrow \vdash A \uparrow$
Proof:

1. $\widehat{\Gamma}, A \downarrow \vdash A \downarrow$ by hypothesis rule
2. $\widehat{\Gamma}, A \downarrow \vdash A \uparrow$ by $\downarrow\uparrow$

Case:

$$\frac{}{\Gamma, \bot\ left\ \vdash C\ right}\ \bot_L$$

RTP: $\widehat{\Gamma}, \bot \downarrow \vdash C \uparrow$
Proof:

1. $\widehat{\Gamma}, \bot \downarrow \vdash \bot \downarrow$ by hypothesis rule
2. $\widehat{\Gamma}, \bot \downarrow \vdash C \uparrow$ by $\bot_E$

Case:

$$\frac{}{\Gamma \vdash \top\ right}\ \top_R$$

RTP: $\widehat{\Gamma} \vdash \top \uparrow$
Proof:

1. $\widehat{\Gamma} \vdash \top \uparrow$ by $\top_I$

Inductive Hypothesis: If $\Gamma \vdash C\ right\ then\ \widehat{\Gamma} \vdash C \uparrow$.

Inductive Cases:
Case:

$$\frac{\Gamma \vdash A\ right \quad \Gamma \vdash B\ right}{\Gamma \vdash A \wedge B\ right}\ \wedge_R$$

Given: $\Gamma \vdash A\ right$ and $\Gamma \vdash B\ right$

RTP: $\widehat{\Gamma} \vdash A \wedge B \uparrow$

Proof:

1.	$\widehat{\Gamma} \vdash A \uparrow$	*ih* on $\Gamma \vdash A\ right$
2.	$\widehat{\Gamma} \vdash B \uparrow$	*ih* on $\Gamma \vdash B\ right$
3.	$\widehat{\Gamma} \vdash A \wedge B \uparrow$	$\wedge_I$ using lines 1 and 2

Case:

$$\frac{\Gamma, A \wedge B\ left, A\ left \vdash C\ right}{\Gamma, A \wedge B\ left \vdash C\ right} \wedge_{L_1}$$

Given: $\Gamma, A \wedge B\ left, A\ left \vdash C\ right$

RTP: $\widehat{\Gamma}, A \wedge B \downarrow\vdash C \uparrow$

Proof:

1.	$\widehat{\Gamma}, A \wedge B \downarrow, A \downarrow\vdash C \uparrow$	*ih* on $\Gamma, A \wedge B\ left, A\ left \vdash C\ right$
2.	$\widehat{\Gamma}, A \downarrow\vdash A \downarrow$	by hypothesis rule
3.	$\widehat{\Gamma}, A \wedge B \downarrow\vdash C \uparrow$	by substitution rule on lines 1 and 2

Case:

$$\frac{\Gamma, A \wedge B\ left, B\ left \vdash C\ right}{\Gamma, A \wedge B\ left \vdash C\ right} \wedge_{L_2}$$

Given: $\Gamma, A \wedge B\ left, B\ left \vdash C\ right$

RTP: $\widehat{\Gamma}, A \wedge B \downarrow\vdash C \uparrow$

Proof:

1.	$\widehat{\Gamma}, A \wedge B \downarrow, B \downarrow\vdash C \uparrow$	*ih* on $\Gamma, A \wedge B\ left, B\ left \vdash C\ right$
2.	$\widehat{\Gamma}, B \downarrow\vdash B \downarrow$	by hypothesis rule
3.	$\widehat{\Gamma}, A \wedge B \downarrow\vdash C \uparrow$	by substitution rule on lines 1 and 2

Case:

$$\frac{\Gamma, A\ left \vdash B\ right}{\Gamma \vdash A \supset B\ right} \supset_R$$

Given: $\Gamma, A\ left \vdash B\ right$

RTP: $\widehat{\Gamma} \vdash A \supset B \uparrow$

Proof:

1.	$\widehat{\Gamma}, A \downarrow\vdash B \uparrow$	*ih* on $\Gamma, A\ left \vdash B\ right$
2.	$\widehat{\Gamma} \vdash A \supset B \uparrow$	by $\supset_E$

Case:

$$\frac{\Gamma, A \supset B\ left \vdash A\ right \quad \Gamma, A \supset B\ left, B\ left \vdash C\ right}{\Gamma, A \supset B\ left \vdash C\ right} \supset_L$$

Given: $\Gamma, A \supset B\ left \vdash A\ right$ and $\Gamma, A \supset B\ left, B\ left \vdash C\ right$

RTP: $\widehat{\Gamma}, A \supset B \downarrow\vdash C \uparrow$

Proof:

1. $\widehat{\Gamma}, A \supset B \downarrow\vdash A \uparrow$ — *ih* on $\Gamma, A \supset B\ left \vdash A\ right$
2. $\widehat{\Gamma}, A \supset B \downarrow\vdash A \supset B \downarrow$ — by hypothesis rule
3. $\widehat{\Gamma}, A \supset B \downarrow\vdash B \downarrow$ — by $\supset_E$ *online*1*and*2
4. $\widehat{\Gamma}, A \supset B \downarrow, B \downarrow\vdash C \uparrow$ — *ih* on $\Gamma, A \supset B\ left, B\ left \vdash C\ right$
5. $\widehat{\Gamma}, A \supset B \downarrow\vdash C \uparrow$ — by substitution rule on lines 3 and 4

Case:

$$\frac{\Gamma \vdash A\ right}{\Gamma \vdash A \vee B\ right} \vee_{R_1}$$

Given: $\Gamma \vdash A\ right$

RTP: $\widehat{\Gamma} \vdash A \vee B \uparrow$

Proof:

1. $\widehat{\Gamma} \vdash A \uparrow$ — *ih* on $\Gamma \vdash A\ right$
2. $\widehat{\Gamma} \vdash A \vee B \uparrow$ — by $\vee_I$ on line 1

Case:

$$\frac{\Gamma \vdash B\ right}{\Gamma \vdash A \vee B\ right} \vee_{R_2}$$

Given: $\Gamma \vdash B\ right$

RTP: $\widehat{\Gamma} \vdash A \vee B \uparrow$

Proof:

1. $\widehat{\Gamma} \vdash B \uparrow$ — *ih* on $\Gamma \vdash B\ right$
2. $\widehat{\Gamma} \vdash A \vee B \uparrow$ — by $\vee_I$ on line 1

Case:

$$\frac{\Gamma, A \vee B\ left, A\ left \vdash C\ right \quad \Gamma, A \vee B\ left, B\ left \vdash C\ right}{\Gamma, A \vee B\ left \vdash C\ right} \vee_L$$

Given: $\Gamma, A \vee B\ left, A\ left \vdash C\ right$ and $\Gamma, A \vee B\ left, B\ left \vdash C\ right$

RTP: $\widehat{\Gamma}, A \vee B \downarrow\vdash C \uparrow$

Proof:

1. $\widehat{\Gamma}, A \vee B \downarrow, A \downarrow\vdash C \uparrow$ — *ih* on $\Gamma, A \vee B\ left, A\ left \vdash C\ right$
2. $\widehat{\Gamma}, A \vee B \downarrow, B \downarrow\vdash C \uparrow$ — *ih* on $\Gamma, A \vee B\ left, B\ left \vdash C\ right$
3. $\widehat{\Gamma}, A \vee B \downarrow\vdash A \vee B \downarrow$ — by hypothesis rule
4. $\widehat{\Gamma}, A \vee B \downarrow\vdash C \uparrow$ — by $\vee_E$ on lines 1,2 and 3

□

C. Proof Search Referred to in Section 7.4

RTP: (((P⊃ ⊥)∧Q)∨(P∧(Q⊃ ⊥))), P, Q ⊢ (((P∧Q)⊃ ⊥)(F∧G))
System's Input: `(((P->!)&Q)|(P&(Q->!))), P, Q |- (((P&Q)->!)&(F&G))`

```
1. (((P->!)&Q)|(P&(Q->!))), a:(P&Q), P, Q |- (((P->!)&Q)|(P&(Q->!)))          Hypothesis -- (((P->!)&Q)|(P&(Q->!)))


      2. b:((P->!)&Q), a:(P&Q), P, Q, (((P->!)&Q)|(P&(Q->!))) |- ((P->!)&Q)          Hypothesis -- ((P->!)&Q)
      3. b:((P->!)&Q), a:(P&Q), P, Q, (((P->!)&Q)|(P&(Q->!))) |- (P->!)          ConjElimLeft -- ((P->!)&Q)

      4. a:(P&Q), P, Q, (((P->!)&Q)|(P&(Q->!))), b:((P->!)&Q) |- P          Hypothesis -- P
5. b:((P->!)&Q), a:(P&Q), P, Q, (((P->!)&Q)|(P&(Q->!))) |- !          ImplElim -- (P->!), P


      6. c:(P&(Q->!)), a:(P&Q), P, Q, (((P->!)&Q)|(P&(Q->!))) |- (P&(Q->!))          Hypothesis -- (P&(Q->!))
      7. c:(P&(Q->!)), a:(P&Q), P, Q, (((P->!)&Q)|(P&(Q->!))) |- (Q->!)          ConjElimRight -- (P&(Q->!))

      8. a:(P&Q), P, Q, (((P->!)&Q)|(P&(Q->!))), c:(P&(Q->!)) |- Q          Hypothesis -- Q
   9. c:(P&(Q->!)), a:(P&Q), P, Q, (((P->!)&Q)|(P&(Q->!))) |- !          ImplElim -- (Q->!), Q
10. (((P->!)&Q)|(P&(Q->!))), a:(P&Q), P, Q |- !          DisjElim -- (((P->!)&Q)|(P&(Q->!)))
11. (((P->!)&Q)|(P&(Q->!))), P, Q |- ((P&Q)->!)          ImplIntro -- (P&Q), !


     12. (((P->!)&Q)|(P&(Q->!))), P, Q |- (((P->!)&Q)|(P&(Q->!)))          Hypothesis -- (((P->!)&Q)|(P&(Q->!)))
```

```
      13. a:((P->!)&Q), P, Q, (((P->!)&Q)|(P&(Q->!))) |- ((P->!)&Q)          Hypothesis -- ((P->!)&Q)
      14. a:((P->!)&Q), P, Q, (((P->!)&Q)|(P&(Q->!))) |- (P->!)          ConjElimLeft -- ((P->!)&Q)

      15. P, Q, (((P->!)&Q)|(P&(Q->!))), a:((P->!)&Q) |- P          Hypothesis -- P
   16. a:((P->!)&Q), P, Q, (((P->!)&Q)|(P&(Q->!))) |- !          ImplElim -- (P->!), P
   17. a:((P->!)&Q), P, Q, (((P->!)&Q)|(P&(Q->!))) |- F          FalsehoodElim


      18. b:(P&(Q->!)), P, Q, (((P->!)&Q)|(P&(Q->!))) |- (P&(Q->!))          Hypothesis -- (P&(Q->!))
      19. b:(P&(Q->!)), P, Q, (((P->!)&Q)|(P&(Q->!))) |- (Q->!)          ConjElimRight -- (P&(Q->!))

      20. P, Q, (((P->!)&Q)|(P&(Q->!))), b:(P&(Q->!)) |- Q          Hypothesis -- Q
   21. b:(P&(Q->!)), P, Q, (((P->!)&Q)|(P&(Q->!))) |- !          ImplElim -- (Q->!), Q
   22. b:(P&(Q->!)), P, Q, (((P->!)&Q)|(P&(Q->!))) |- F          FalsehoodElim
23. (((P->!)&Q)|(P&(Q->!))), P, Q |- F          DisjElim -- (((P->!)&Q)|(P&(Q->!)))


   24. (((P->!)&Q)|(P&(Q->!))), P, Q |- (((P->!)&Q)|(P&(Q->!)))          Hypothesis -- (((P->!)&Q)|(P&(Q->!)))


      25. a:((P->!)&Q), P, Q, (((P->!)&Q)|(P&(Q->!))) |- ((P->!)&Q)          Hypothesis -- ((P->!)&Q)
      26. a:((P->!)&Q), P, Q, (((P->!)&Q)|(P&(Q->!))) |- (P->!)          ConjElimLeft -- ((P->!)&Q)

      27. P, Q, (((P->!)&Q)|(P&(Q->!))), a:((P->!)&Q) |- P          Hypothesis -- P
   28. a:((P->!)&Q), P, Q, (((P->!)&Q)|(P&(Q->!))) |- !          ImplElim -- (P->!), P
   29. a:((P->!)&Q), P, Q, (((P->!)&Q)|(P&(Q->!))) |- G          FalsehoodElim


      30. b:(P&(Q->!)), P, Q, (((P->!)&Q)|(P&(Q->!))) |- (P&(Q->!))          Hypothesis -- (P&(Q->!))
      31. b:(P&(Q->!)), P, Q, (((P->!)&Q)|(P&(Q->!))) |- (Q->!)          ConjElimRight -- (P&(Q->!))
```

```
                    32. P, Q, (((P->!)&Q)|(P&(Q->!))), b:(P&(Q->!)) |- Q          Hypothesis -- Q
                  33. b:(P&(Q->!)), P, Q, (((P->!)&Q)|(P&(Q->!))) |- !          ImplElim -- (Q->!), Q
                  34. b:(P&(Q->!)), P, Q, (((P->!)&Q)|(P&(Q->!))) |- G          FalsehoodElim
              35. (((P->!)&Q)|(P&(Q->!))), P, Q |- G          DisjElim -- (((P->!)&Q)|(P&(Q->!)))
          36. (((P->!)&Q)|(P&(Q->!))), P, Q |- (F&G)          ConjIntro -- F, G
    37. (((P->!)&Q)|(P&(Q->!))), P, Q |- (((P&Q)->!)&(F&G))          ConjIntro -- ((P&Q)->!), (F&G)
```

D. CD Contents

The CD contains a softcopy of this document. It also contains the source code for the system implemented. To use the system:

- Make sure you have got a Haskell interpreter installed on your system
- Open the *src* directory from the disc
- Double Click on ATP.hs
- Follow the instructions given. Make sure that when you are asked to enter the judgment you wish to prove or verify you enter it in the following format <antecedents separated by a ,(comma)>|-<succedent>

Bibliography

[1] Arun-Kumar S., *Introduction to Logic for Computer Science*, 2002, http://www.cse.iitd.ernet.in/∼sak/courses/ilcs/logic.pdf, last accessed: 20-05-2011

[2] Cittadini Saverio, *Intercalation Calculus for Propositional Intuitionistic Logic*, Department of Philosophy, Paper 251, 1992, http://repository.cmu.edu/philosophy/251

[3] D'Agostino Marcello, *Classical Natural Deduction*,We Will Show Them! (1), 429-468, 2005 http://web.unife.it/utenti/marcello.dagostino/papers/DAgostino.pdf, last accessed: 15-05-2011

[4] Detlovs Vilnis and Podnieks Karlis, *Introduction to Mathematical Logic*, 2000, http://www.ltn.lv/ podnieks/mlog/ml.htm, last accessed: 26-05-2011

[5] Doaitse Swierstra S., *Combinator Parsing: A Short Tutorial*, 2007, http://www.cs.tufts.edu/ nr/cs257/archive/doaitse-swierstra/combinator-parsing-tutorial.pdf, last accessed: 26-05-2011

[6] Dyckhoff Roy, Leslie Neil, Peillon Tom, Rapley Brenda, Pinto Luis, Andrew Adams, Christian Urban, Howe Jacob and others, *MacLogic 3 documentation*,Section 4.3: Sequent Caculi, 1999, http://www.cs.st-andrews.ac.uk/∼rd/logic/mac/docs/, last accessed: 26-05-2011

[7] Gallier Jean H. , *Logic For Computer Science Foundations of Automatic Theorem Proving*,John Wiley & Sons Inc,1986

[8] Galmiche D. and Perrier G., 1994, *Foundations of proof search strategies design in linear logic*,In Symposium on Logical Foundations of Computer Science, Volume 813/1994, 101-113, 1994

[9] Gentzen Gerhard , *Investigation in Logic Deductions*,American Philosophy Quarterly Journal, Volume 1,288-306,1964

[10] Howard W. A., *The formulae-as-types notion of construction*, In J. P. Seldin and J. R. Hindley, editors, To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism,Academic Press, 479-490, 1980

[11] Howe, J. M., *Proof Search Issues in Some Non-Classical Logics*, PhD thesis, University of St Andrews (Research Report CS/99/1), 1998

[12] Huth Michael R. A. and Ryan Mark D, *Logic in Computer Science; Modelling and Reasoning about Systems*,Cambridge University Press,2000

[13] Johan Van Benthem, Hans Van Ditmarsch, Jan van Eijck and Jan Jaspars, *Project: Logic In Action*, 2011, http://www.logicinaction.org/docs/lia.pdf, last accessed: 26-05-2011

[14] Liang Chuck and Miller Dale, *Focusing and polarization in intuitionistic logic*, CSL 2007: Computer Science Logic, volume 4646 of LNCS, 451-465, Springer-Verlag, 2007

[15] MacFarlane John Gordon, *What does it mean to say that logic is formal?*, PhD thesis, University of Pittsburgh, 2000

[16] McLaughlin Sean and Pfenning Frank, *Imogen: Focusing the Polarized Inverse Method for Intuitionistic Propositional Logic*, Department of Computer Science, Carnegie Mellon University, 2008, http://www.cs.cmu.edu/∼fp/papers/lpar08.pdf, last accessed: 26-05-2011

[17] Medak Damir and Navrati Gerhard, *Haskell Tutorial*,2003, ftp://ftp.geoinfo.tuwien.ac.at/navratil/HaskellTutorial.pdf, last accessed: 24-05-2011

[18] Pace Gordon, *Mathematics of Discrete Structures from a Computing Perspective - Draft Version 1*,2003

[19] Pfenning Frank, *Automated Theorem Proving*, Draft 2004, http://www.cs.cmu.edu/∼fp/courses/atp/handouts/atp.pdf, last accessed: 23-05-2011

[20] Pfenning Frank , *Lecture Notes on Inversion*, 2009, http://www.cs.cmu.edu/∼fp/courses/15317-f09/lectures/11-inversion.pdf, last accessed: 26-05-2011

[21] Pfenning Frank, *Lecture Notes on Judgments and Propositions*, last accessed: 26-05-2011 http://www.cs.cmu.edu/∼fp/courses/15816-s10/lectures/01-judgments.pdf, 2010

[22] Pfenning Frank, *Lecture Notes on Proofs as Programs*, 2010, http://www-cgi.cs.cmu.edu./∼fp/courses/15816-s10/lectures/02-pap.pdf, last accessed: 20-05-2011

[23] Pfenning Frank, *Lecture Notes on The Curry-Howard Isomorphism*, 2004, http://www.cs.cmu.edu/∼fp/courses/15312-f04/handouts/23-curryhoward.pdf, last accessed: 20-05-2011

[24] Pfenning Frank, *Lecture Notes on Sequent Calculus*,2010, http://www.cs.cmu.edu/∼fp/courses/15816-s10/lectures/08-seqcalc.pdf, last accessed: 20-05-2011

[25] *Propositional Logic*, http://www.cis.upenn.edu/∼cis510/tcl/chap3.pdf, last accessed: 20-05-2011

[26] Reeves Steve and Clarke Mike, *Logic for Computer Science*, Pearson Education Limited, 1990

[27] *Using NAND or NOR gates to make up Circuits*, http://www.antonine-education.co.uk/Electronics_AS/Electronics_Mod2/topic_2_1/using_nand___or_nor_gates_to_make.htm, last accessed: 26-05-2011

[28] Van Emden M. H. and Kowalski R. A., *The Semantics of Predicate Logic as a Programming Language*, Journal of the ACM, 23, 596-574, 1976

[29] Van Oosten Jaap , *Intuitionism*, 1996, http://www.staff.science.uu.nl/∼ooste110/syllabi/intuitionism.pdf, last accessed: 12-04-2011

[30] Wadler Philip, *Proofs are Programs: 19th Century Logic and 21st Century Computing*, 2000, http://homepages.inf.ed.ac.uk/wadler/papers/frege/frege.pdf, last accessed: 26-05-2011

[31] Weiss William and D'Mello Cherie, *Fundamentals of Model Theory*, University of Toronto, 1997