Coastline Final Report

Koissi Adjorlolo
Consultant: Prof. Branden Fitelson

November 2019

Abstract

During the Spring 2019 semester I completed a Human-Computer Interaction directed study with Professor David Sprague [1]. The goal of the directed study was to produce a prototype for Shoreline, a macOS application that allows high school algebra students to solve simple math problems on their computer. Shoreline is driven by a Term Rewrite System (TRS), which is defined as a set of terms representing simple algebraic expressions, e.g. $1+2=3$, $y=mx+b$, etc., paired with a list of rules that determine how to transform one term into another. Shoreline's TRS was limited to terms defined using the unary operator $-$, binary operators / and $=$, and $+$ and $*$ as n-ary operators. I have since generalized the definition of terms in Shoreline and added the ability to add rules to the TRS arbitrarily.

After completing the directed study I set a goal for myself this semester in Math 4020 to come up with a way of manipulating and combining more than one algebraic expression at a time. I also decided to call any extensions to Shoreline I made throughout this semester Coastline, since a Coastline is like a Shoreline but covers more ground. Over the course of this project, guided by Professor Branden Fitelson, I learned about proof systems and compared Shoreline to existing work in Mathematical Logic, focusing on Gerhard Gentzen's Natural Deduction systems, Condensed Detachment, and the Unification Theorem. Learning Natural Deduction has inspired me to implement aspects of the system within Coastline, as it provides a natural way of combining logical truths to infer more truths, an act central to all of mathematics. Finally, I combined the definitions and extensions I made to Shoreline's TRS with what I learned about Natural Deduction, resulting in an alternative description of some common rules of inference in Natural Deduction systems.

1 Introduction

I started thinking about Shoreline/Coastline in 2017 as an online high school math and science tutor. The lack of face-to-face interaction I had with students often required me to write out equations on an online whiteboard in order to walk through simple algebra problems. A major downside of these online whiteboards is their static nature. If I wrote down one line of an algebraic manipulation, I'd have to write each modified version of the original line many times before completing the problem. When it came time for a student to take over and try the same problem on the online whiteboard, they were held back by the tedium required to write out each line by hand, or most often by their lack of proper input devices, such as an external writing tablet. Determined to overcome this problem, I looked for a computer application that allowed for the simple manipulation of algebraic expressions. I didn't find anything that came close to suiting my needs, so I began brainstorming possible solutions to this problem.

Shoreline was my first attempt at such a solution. I developed it from a human-computer interaction point of view, and therefore was much more concerned about basic interface details

than I was with the underlying theory. After the current semester began, I set out to learn about work others have done on systems that resembled Shoreline's central functions. Professor Branden Fitelson, who was vaguely familiar with my project, offered to assist me in learning about some of Shoreline's underlying theory. He pointed out similarities between what I had done and Term Rewrite Systems, as well as the resemblance between operations I defined for Shoreline which I called *rewrite* and $findBinding$.

The *rewrite* operation closely resembles a rule of inference called Condensed Detachment, while the $findBinding$ operation closely resembles the Unification algorithm [6]. Definitions central to Shoreline, as well as descriptions of the *rewrite* and $findBinding$ operators are laid out in section 2. I spent much of this semester learning about the subtle differences between the operators I developed for Shoreline and Condensed Detachment and Unification. I spent the rest of the semester learning about Natural Deduction by working through chapter 4 of [3] and implementing a proof system resembling Natural Deduction using the definitions from section 2. Section 3 introduces some of the connections between Shoreline and logic, section 4 describes some of what I learned about Natural Deduction based proof systems, and section 5 describes a basic proof system using Shoreline's basic definitions.

2 Shoreline

Shoreline started as a prototype macOS application written in Swift, and has therefore already been "formalized" in a sense. Unfortunately, code written in Swift can be difficult for someone unfamiliar with the syntax to understand, especially if this person has limited experience programming. This, paired with some poor development decisions on my part, make for a formalization that is difficult to reason about.[1] With this in mind I decided to formalize Shoreline in terms that are clearer to someone who is more familiar with standard mathematical notation than they are with Swift.

2.1 Abstract Reduction Systems

Shoreline is based on a Term Rewrite System (TRS), which is a special type of Abstract Reduction System (ARS).

An ARS is defined as a pair (A, R), where A is a set of objects and R is a binary relation on A. Each pair in R is called a rule.

2.2 Term Rewrite Systems

Informally, a **term** is a representation of information organized hierarchically, so that terms can be contained within other terms. Algebraic expressions are most naturally represented using terms, as the expression $2 * (3 + 4)$ is made up of the two terms 2 and $(3 + 4)$, which in turn is made up of the terms 3 and 4. I will be considering the $+$ operator as a term, too, as doing so will allow me to represent more than just algebraic expressions.

2.2.1 Definition of Terms

A term can be one of the following:

[1] Some of these poor development decisions include a lack of documentation, sparse testing of crucial components, and my decision to use Swift instead of a language I'm more familiar with, such as Java. Northeastern's course *Fundamentals of Computer Science I* warns students not to do these things. I would have avoided some of these decisions if I had more time.

- An n-ary function: a term representing a collection of n terms, where n is a natural number
- A variable: a term that can be used in place of any other term

This is a recursive definition, in that some kinds of terms are defined using other terms.

2.2.2 TRSs as a special kind of ARS

A TRS is an ARS (T, R), where $T = \Sigma \cup X$ is the set of all terms expressible within the system, Σ is a set of function terms, X is a set of variables, and $\Sigma \cap X = \emptyset$. For every TRS, the set X contains all possible variables, while Σ may or may not contain all possible function terms.

The **arity** of a function term refers to the number of terms within the function term. If f is a function term, f is of arity n and $t_1, t_2, ..., t_n$ are all terms, then $f(t_1, t_2, ..., t_n)$ is also a function term. [2]

A set of function terms Σ can be defined as the disjoint union of all n-ary function terms in T, where n is a natural number. For example, $\Sigma^{(k)}$ is the set of all k-ary functions in T, where k is a natural number.

$$\Sigma = \bigcup_{i \in \mathbb{N}} \Sigma^{(i)}$$

If l and r are terms and the pair (l, r) is in R, we can "rewrite" every occurrence of l with the term r. I will be referring to rules within TRS's as **rewrite rules**. What it means to rewrite a term using a rewrite rule is described in section 2.3.

2.2.3 Notation used in this document

Specific 0-ary terms will be referred to as underlined alphanumeric strings, e.g. cat to refer to the term made up of the characters 'c', 'a', and 't'. There is one special kind of 0-ary term that will be represented using the empty parenthetical ().

Function terms of arity n for all $1 \leq n$ will be referred to as a list of terms separated by spaces enclosed in parenthesis, e.g. (t_1 t_2 ... t_n), where t_1, t_1, ..., t_n are all in terms. Similarly, (cat rat dog) refers to the 3-ary term made up of the terms cat, rat, and dog. Due to the recursive definition of terms, function terms can contain other function terms, and can be nested arbitrarily. For example, (1 * (2 * 3)) is a valid term representing the algebraic expression $1 * (2 * 3)$.

Variable terms will be represented using a period followed by an alphanumeric string, e.g. .x, .y, .z.

A rewrite rule (l, r) will be referred to using the notation $l \to r$, where l and r are terms. For example, a rule that rewrites the term cat into the term dog will be written as follows.

$$\text{cat} \to \text{dog}$$

A rule that rewrites the term (cat rat dog) into the term .y will be written as follows.

$$(\text{cat rat dog}) \to .y$$

2.3 Operators on Terms and Rules

Shoreline relies on four operators for rewriting terms using rewrite rules. The domains and codomains of these operators will be defined using T_U, representing the set all terms, R_U, representing the set of all rewrite rules, and B, representing the set of all "bindings", or functions from a set of variables to terms. This section will also use $\Sigma_U^{(n)}$ to denote the set of all n-ary function terms, and X_U to denote the set of all variable terms.

The operator *match* relates two terms and can be used to determine if a rewrite rule $l \to r$ can be used to rewrite a term t.

$$match : T_U \times T_U \to \{true, false\}$$

Terms can be rewritten into other terms using the *rewrite* operator. If we know that a term t belongs to a given TRS and a rule $l \to r$ belongs to the same TRS, the result of a call to $rewrite(l \to r, t)$ is also a term in the TRS.

$$rewrite : R_U \times T_U \to T_U$$

The operators $findBinding$ and *substitute* are required to define *rewrite*, but can be used separately.

$$findBinding : T_U \times T_U \to B$$

$$substitute : T_U \times B \to T_U$$

We will first look at an example problem in Shoreline that requires the use of these operators, and then define each operator more precisely.

2.3.1 Example problem: From $b = a$ to $a = b$

Let (T, R) be a TRS, where $(\underline{\text{b}} \equiv \underline{\text{a}})$ is a term in T, and $R = \{(.\text{x} \equiv .\text{y}) \to (.\text{y} \equiv .\text{x})\}$. We want to show that $(\underline{\text{a}} \equiv \underline{\text{b}})$ is also a term in T, and can do so with the following call to *rewrite*.

$$rewrite((.\text{x} \equiv .\text{y}) \to (.\text{y} \equiv .\text{x}), (\underline{\text{b}} \equiv \underline{\text{a}})) = (\underline{\text{a}} \equiv \underline{\text{b}})$$

Therefore, $(\underline{\text{a}} \equiv \underline{\text{b}})$ must also be an element of T.

This works by finding a function b from the set $\{.\text{x}, .\text{y}\}$ to the set $\{\underline{\text{b}}, \underline{\text{a}}\}$ such that $(b(.\text{x}) \equiv b(.\text{y}))$ is equal to $(\underline{\text{b}} \equiv \underline{\text{a}})$. The function b is called a **binding**, since it "binds" the variables in its domain to terms in its codomain. The *rewrite* operator can only be called with the given term and rule if such a binding exists. The operator $findBinding$ is what finds this binding.

$$b : \{.\text{x}, .\text{y}\} \to \{\underline{\text{b}}, \underline{\text{a}}\}$$

$$b = findBinding((.\text{x} \equiv .\text{y}), (\underline{\text{b}} \equiv \underline{\text{a}}))$$

We cannot call *rewrite* with the given terms if $findBinding$ is undefined for its input.

We then apply b to the variables in the right hand side of the rule given to *rewrite* to obtain our desired result, $(b(.\text{y}) \equiv b(.\text{x}))$, which is equal to $(\underline{\text{a}} \equiv \underline{\text{b}})$. The *substitute* operator is used to apply a binding b to the variables inside a given term.

$$substitute((.\text{y} \equiv .\text{x})), b) = (b(.\text{y}) \equiv b(.\text{x})) = (\underline{\text{a}} \equiv \underline{\text{b}})$$

2.3.2 Defining $findBinding$

Let Y be a subset of all variables, and S be a subset of all terms. A "binding" b in the set of bindings B is any function with the following signature.

$$b : Y \to S$$

The function $findBinding$ takes in two terms and returns a binding.

$$findBinding : T_U \times T_U \to B$$

For all variables x and all terms t, if $b = findBinding(x, t)$, then $Y = \{x\}$, $S = \{t\}$, and $b(x) = t$.

$$b : \{x\} \to \{t\}$$

$$\forall x \in X_U, \forall t \in T_U, b = findBinding(x, t) \Rightarrow b(x) = t$$

For all 0-ary functions f and all terms t, if $b = findBinding(f, t)$, then $Y = S = \emptyset$.

$$b : \emptyset \to \emptyset$$

For all n-ary functions $t = (t_1\ t_2\ ...\ t_n)$ and $s = (s_1\ s_2\ ...\ s_n)$, $findBinding(t, s)$ returns the union of $findBinding(t_i, s_i)$ for all $1 \leq i \leq n$.

$$\forall t, s \in \Sigma_U^{(n)}, t = (t_1\ t_2\ ...\ t_n),\ s = (s_1\ s_2\ ...\ s_n),$$

$$findBinding(t, s) = \bigcup_{1 \leq i \leq n} findBinding(t_i, s_i)$$

2.3.3 The *match* operator

The operator *match* returns *true* if a call to $findBinding$ is defined for its given input, and false otherwise. I defined *match* separately from $findBinding$ before I noticed their relationship to each other, but still use *match* in places where a more efficient operator is needed, such as when searching for all rules that can be used to rewrite a given term.

An equivalent definition for *match* says that $match(x, t)$ returns true if and only if there exists a binding b such that $substitute(x, b) = t$.

2.3.4 Defining *substitute*

The function *substitute* has the following signature, where B is the set of all bindings.

$$substitute : T_U \times B \to T_U$$

Let b be a binding with the signature $b : Y \to S$, where Y is a subset of all variables and S is a subset of all terms, and let y be a variable in Y. Then $substitute(y, b) = b(y)$.

$$\forall y \in Y, b \in B, b : Y \to S, substitute(y, b) = b(y)$$

If t is a term such that $t \notin Y$, then substitute(t, b) = t.

$$\forall t \notin Y, b \in B, b : Y \to S, substitute(t, b) = t$$

Let t be an n-ary function such that $t = (t_1\ t_2\ ...\ t_n)$. Then $substitute(t, b)$ is equal to the function term formed by the results of calling $substitute(t_i, b)$ for all $0 \leq i \leq n$.

$$\forall y \in Y, b \in B, b : Y \to S$$

$$substitute(t, b) = (substitute(t_1, b)\ substitute(t_2, b)\ ...\ substitute(t_n, b))$$

2.3.5 Defining *rewrite*

The function *rewrite* has the following signature.

$$rewrite : R_U \times T_U \to T_U$$

For all terms t in T and all rules $l \to r$ in R, $rewrite(t, l \to r)$ is equal to the binding found by $findBinding(l, t)$ applied to every term in the right side r of the given rule.

$$\forall t \in T, \forall (l \to r) \in R_U$$

$$rewrite(l \to r, t) = substitute(r, findBinding(l, t))$$

3 From Shoreline to Coastline

I called my directed study project "Shoreline" because I was spending too much time trying to come up with a clever name for it, and Shoreline is the town I was raised in. During the following summer, the Math Department's Undergraduate Research Committee was kind enough to fund my trip to the Applications of Computer Algebra Conference in Montréal to present a poster on Shoreline. While presenting my poster, a retired professor remarked that Shoreline was a fitting name for my project, since a shoreline sits at the boundary between land and water, just as my prototype sits at the boundary between computer science and mathematics. Inspired by this professor's retrofitted justification for the name, I decided to call any future theoretical work I did on my project "Coastline", as a coastline is like a shoreline, but covers more ground.

My overarching goal of developing software that enables users to solve a variety of symbolic math problems on a computer requires that any such software is capable of expressing fundamental concepts in mathematics. This project is possible thanks to questions posed and work done in the early 20th century on the fundamentals of mathematics. Many of these fundamental questions were about formal logic, since all mathematical proofs require some kind of logical reasoning. Therefore, any software that claims to capture aspects of mathematical reasoning has roots in formal logic, and Coastline is no different. With this in mind, much of the work I have done this semester is from the point of view of formal logic.

3.1 Using *rewrite* in place of Modus Ponens

The *rewrite* operator is similar to a procedure used in automated reasoning called **Condensed Detachment**[6], which in turn resembles a common tool in the logician's toolbox called **Modus Ponens**. Modus Ponens says that if we know the propositional formula $P \Rightarrow Q$ to be true (the major premise), and know the proposition P to be true (the minor premise), we can conclude that the proposition Q is also true.

In Coastline, the rule $l \to r$ can be thought of as the implicational statement meaning "if the term l is in T, then the term r is in T". This rule resembles a major premise used in the application of Modus Ponens. When the same rule is plugged into the *rewrite* operator along with some term t, as in $rewrite(t, l \to r)$, the meaning of the rule becomes "if a term that matches l is in T, then some term matching r is also in T". Here, t is the term matching l, and the result of $rewrite(t, l \to r)$ is the term matching r. Based on the definition of *rewrite*, the term that has been "discovered" depends on the result of $findBinding(l, t)$. In this way, *rewrite* can be thought of as a more general version of Modus Ponens, as it can do everything Modus Ponens can do and much more.

3.1.1 Simple Example of *rewrite* as Modus Ponens

To illustrate how *rewrite* is similar to Modus Ponens, let's assume the 0-ary terms $\underline{\text{A}}$, $\underline{\text{B}}$, and $\underline{\text{C}}$ are in T, and the rules $\underline{\text{A}} \to \underline{\text{B}}$ and $\underline{\text{B}} \to \underline{\text{C}}$ are in R. Then we can rewrite $\underline{\text{A}}$ to conclude that $\underline{\text{C}}$ is also a term in T using the following sequence of rewrites.

$$rewrite(\underline{\text{A}} \to \underline{\text{B}}, \underline{\text{A}}) = \underline{\text{B}}$$

$$rewrite(\underline{\text{B}} \to \underline{\text{C}}, \underline{\text{B}}) = \underline{\text{C}}$$

We can represent the evaluation of the above rewrites as follows.

$$rewrite(\underline{\text{B}} \to \underline{\text{C}}, rewrite(\underline{\text{A}} \to \underline{\text{B}}, \underline{\text{A}})) = rewrite(\underline{\text{B}} \to \underline{\text{C}}, \underline{\text{B}}) = \underline{\text{C}}$$

3.1.2 Example of *rewrite* as a General Modus Ponens

The term ($\underline{1} \underline{*} \underline{2}$) matches the term (.a $\underline{*}$ .b), and can therefore be rewritten using the rule (.a $\underline{*}$ .b) $\rightarrow$ (.b $\underline{*}$ .a).

$$rewrite((.a \underline{*} .b) \rightarrow (.b \underline{*} .a), (\underline{1} \underline{*} \underline{2})) = (\underline{2} \underline{*} \underline{1})$$

3.1.3 Rules as Terms

Making a TRSs set of rules R a subset of its set of terms T allows us to combine rules to reason about the existence of new rules. For example, if we know that the rules $\underline{A} \rightarrow \underline{B}$ and $(\underline{A} \rightarrow \underline{B}) \rightarrow (\underline{C} \rightarrow \underline{D})$ are in R, then we can conclude that $\underline{C} \rightarrow \underline{D}$ is also a rule in R using the following rewrite.

$$rewrite((\underline{A} \rightarrow \underline{B}) \rightarrow (\underline{C} \rightarrow \underline{D}), \underline{A} \rightarrow \underline{B}) = \underline{C} \rightarrow \underline{D}$$

Combining rules as if they were implicational formulas can lead to the discovery of new ways to manipulate terms, but Coastline in this extended form still lacks many of the tools mathematicians rely on to reason. I intend to extend Coastline further in order to capture this more complicated kind of reasoning. Gerhard Gentzen's Natural Deduction captures this kind of reasoning.

4 Natural Deduction[3]

Natural Deduction encompasses many different systems used to prove formal statements. It was originally developed by the German mathematician Gerhard Gentzen in attempt to capture the "natural" way that mathematicians reason. This makes Natural Deduction a good starting point for Coastline, since it must feel natural to mathematicians who use it.

Proofs using Natural Deduction resemble upside-down trees, where the leaves are the axioms and assumptions used in the proofs, and the root of the tree is the proof's conclusion. Section 4.2 walks through the construction of such a proof.

I focused on learning Gentzen's system NK, which is used to study classical, first-order logic.

4.1 Rules of Inference

Rules of Inference define the ways in which formulae that are known to be valid within a system are combined to infer other valid formulae. They share a common form, where the **premises**, or valid formulae that are to be combined, are listed above a horizontal line, and the **conclusion**, the formula inferred from the premises, is listed below the line. The name of the rule of inference being used is displayed to the right of the horizontal line.

$$\frac{\text{Premise 1} \qquad \text{Premise 2} \qquad \ldots \qquad \text{Premise } n}{\text{Conclusion}} \text{ Rule of Inference}$$

4.1.1 Modus Ponens as a Rule of Inference

Modus ponens (written as $\Rightarrow$E) is one such rule of inference, as it takes the premises P and $P \Rightarrow Q$ and with them concludes Q, where P and Q are propositions.

$$\frac{P \Rightarrow Q \qquad P}{Q} \Rightarrow \text{E}$$

4.1.2 The *rewrite* operator as a Rule of Inference

The *rewrite* operator can also be used as a rule of inference. This will allow us to construct a proof that $(\underline{2} \underline{*} \underline{1})$ is a term in T from the terms $(\underline{1} \underline{*} \underline{2})$ and $(.a \underline{*} .b) \rightarrow (.b \underline{*} .a)$ as premises.

$$\frac{(.a \underline{*} .b) \rightarrow (.b \underline{*} .a) \qquad (\underline{1} \underline{*} \underline{2})}{(\underline{2} \underline{*} \underline{1})}\ rewrite$$

4.1.3 Assuming propositions

When proving statements of the form $A \Rightarrow B$, where A and B are formulae, mathematicians first assume A and use this assumption to derive B. Natural Deduction has a rule of inference for this kind of reasoning, simply called **Assume**. It is the only rule of inference (that I know of) that doesn't rely on any premises, since it is possible to assume the truth of any statement as long as this assumption is accounted for later in the proof. This is the only rule of inference I will be referring to without a label.

$$\overline{P}$$

In the above example, P can be any formula.

4.1.4 Introducing and Eliminating $\Rightarrow$

Mathematicians need more rules of inference beyond Assume in order to prove statements of the form $A \Rightarrow B$. Modus Ponens ($\Rightarrow$E) and Implication Introduction ($\Rightarrow$I) are two rules of inference which, when paired with Assume, make this kind of reasoning possible. We've already seen how $\Rightarrow$E is used in section 4.1.1. Another name for $\Rightarrow$E is **Implication Elimination**, since it takes a formula that contains the $\Rightarrow$ connective and eliminates it in the conclusion.

Implication Introduction ($\Rightarrow$I) works opposite to $\Rightarrow$E, as it takes a set of premises without the $\Rightarrow$ connective and introduces it in the conclusion. In section 4.1.3 I mentioned that assumptions must be accounted for later on in a proof. This accounting is handled by the "discharging" of assumptions, and $\Rightarrow$I is one rule of inference that can do this.

$$\frac{\begin{array}{c}\overline{[P]^x} \\ \vdots \\ Q\end{array}}{P \Rightarrow Q}\ \Rightarrow\mathrm{I}^x$$

The $[P]^x$ at the top of the proof refers to an assumption P that has been discharged. The superscript x is a label referenced by $\Rightarrow\mathrm{I}^x$, used to refer to the assumption being discharged by $\Rightarrow$I. This label is necessary if there is more than one assumption being discharged in a proof. The vertical ... represents a proof with the assumption P as its leaf, which then derives Q. The conclusion of this rule of inference is $P \Rightarrow Q$. An assumption cannot be used after it has been discharged unless it is assumed again later in the proof.

A more concrete example of $\Rightarrow$I is illustrated in the following proof of $P \Rightarrow R$, where $P \Rightarrow Q$ and $Q \Rightarrow R$ are known to be true.

$$\dfrac{\dfrac{Q \Rightarrow R \qquad \dfrac{P \Rightarrow Q \qquad \overline{[P]^1}}{\mathrm{Q}} \Rightarrow\mathrm{E}}{R} \Rightarrow\mathrm{E}}{P \Rightarrow R} \Rightarrow\mathrm{I}^1$$

This example also illustrates how the conclusion of one rule of inference can become a premise in another rule of inference. All Natural Deduction proofs are constructed in this way. A more complicated example will appear in section 4.2.

Most popular rules of inference that I have come across come in pairs, where one of the rules introduces a given connective and the other eliminates it.

4.1.5 Introducing and Eliminating &

And Introduction (&I) and **Elimination** (&E) introduces the & connective in the conclusion and eliminates the & connective from the premises, respectively. The rule of inference &E comes in two flavors. The first of which eliminates & to conclude the first operand, and is labelled as $\&\mathrm{E}_1$.

$$\frac{P \ \& \ Q}{P} \ \&\mathrm{E}_1$$

The second kind of &E eliminates the second operand, and is labelled as $\&\mathrm{E}_2$.

$$\frac{P \ \& \ Q}{Q} \ \&\mathrm{E}_2$$

The & connective can be introduced from two premises using &I.

$$\frac{P \qquad Q}{P \ \& \ Q} \ \&\mathrm{I}$$

4.2 Example Natural Deduction Proof

An example proof is illustrated below. We'll start with what we're trying to prove.

$$(P \Rightarrow (Q \Rightarrow R)) \Rightarrow ((P \ \& \ Q) \Rightarrow R)$$

We will assume that $P \ \& \ Q$ and $P \Rightarrow (Q \Rightarrow R)$ are true, then discharge these assumptions before the proof is finished.

$$\overline{P \ \& \ Q}$$

$$\overline{P \Rightarrow (Q \Rightarrow R)}$$

We made the above assumptions so that we can infer P, Q, and $Q \Rightarrow R$ to conclude R from the application of $\Rightarrow$E.

First, we use the assumption $P \ \& \ Q$ to conclude Q.

$$\dfrac{\dfrac{\overline{P \;\&\; Q}}{Q}\,\&\mathrm{E}_2}{}$$

We then use the assumption $P \;\&\; Q$ to conclude P, and combine P with the assumption $P \Rightarrow (Q \Rightarrow R)$ using $\Rightarrow$E to conclude $Q \Rightarrow R$.

$$\dfrac{\dfrac{\overline{P \Rightarrow (Q \Rightarrow R)} \qquad \dfrac{\overline{P \;\&\; Q}}{P}\,\&\mathrm{E}_1}{Q \Rightarrow R}\Rightarrow\mathrm{E}}{}$$

Combining these last two proofs with another application of $\Rightarrow$E will then allow us to conclude R.

$$\dfrac{\dfrac{\dfrac{\overline{P \Rightarrow (Q \Rightarrow R)} \qquad \dfrac{\overline{P \;\&\; Q}}{P}\,\&\mathrm{E}_1}{Q \Rightarrow R}\Rightarrow\mathrm{E} \qquad \dfrac{\overline{P \;\&\; Q}}{Q}\,\&\mathrm{E}_2}{R}\Rightarrow\mathrm{E}}{}$$

The final two steps require that we discharge each of our assumptions. We've only seen one rule of inference so far that can do this, $\Rightarrow$I. The final proof with all our assumptions discharged appears below.

$$\dfrac{\dfrac{\dfrac{\dfrac{\overline{[P \Rightarrow (Q \Rightarrow R)]^2} \qquad \dfrac{\overline{[P \;\&\; Q]^1}}{P}\,\&\mathrm{E}_1}{Q \Rightarrow R}\Rightarrow\mathrm{E} \qquad \dfrac{\overline{[P \;\&\; Q]^1}}{Q}\,\&\mathrm{E}_2}{R}\Rightarrow\mathrm{E}}{(P \;\&\; Q) \Rightarrow R}\Rightarrow\mathrm{I}^1}{(P \Rightarrow (Q \Rightarrow R)) \Rightarrow ((P \;\&\; Q) \Rightarrow R)}\Rightarrow\mathrm{I}^2$$

Now that we have obtained the formula we set out to prove and discharged all irrelevant assumptions, our proof is complete.

5 Natural Deduction using Terms

I decided to begin designing a computer program that is capable of representing, checking, and building Natural Deduction-based proofs as the final part of my capstone. Whatever proof checker/builder I end up implementing must allow the simple addition and removal of any rule of inference that I may come across or dream of on my own. I found that terms are a simple way of specifying such rules of inference in a flexible way.

Unfortunately, Natural Deduction as I have presented it thus far requires keeping track of assumptions every time a rule of inference is used, so that some rules of inference are capable of changing state not specified by the its premises. The rules of infererence Assume and $\Rightarrow$I are two such rules of inference, as the former adds a new proposition to a proof's assumptions and the latter discharges an assumption. Getting around this requires that a rule of inference's premises carry every assumption they rely on with them, and that a rule of inference's conclusion preserves these assumptions unless they are being discharged. I was delighted to find that [3] defines a kind of object that keeps track of a proposition's assumptions.

5.1 Sequents

A **Sequent** in Natural Deduction takes the form

$$a_1, a_2, ..., a_n \vdash_{\mathcal{L}} e$$

where $a_1, a_2, ..., a_n$ is a list of assumptions, e is a statement following from $a_1, a_2, ..., a_n$, and $\mathcal{L}$ is the logical system in which e follows from $a_1, a_2, ..., a_n$.

Sequents can be used to declare theorems that rely on certain assumptions. The statement we prove in 4.1.4 is "$P \Rightarrow R$, where $P \Rightarrow Q$ and $Q \Rightarrow R$ are known to be true". This is a cumbersome way of stating what we need to prove, and can instead be stated using the following sequent.

$$P \Rightarrow Q, Q \Rightarrow R \vdash_{NK} P \Rightarrow R$$

Normally one would write $\vdash_{NK}$ when working with sequents within the proof system NK, but I won't be referring to other systems. However, every proof I present in this document is from within NK, so for the rest of this document the NK in $\vdash_{NK}$ will be dropped so that it looks like $\vdash$.

Using sequents removes the need for the bracket/label notation used to represent discharged assumptions, e.g. $[P]^x$ in 4.1.4, since we now know that a proposition is being assumed if it appears on the left hand side of any $\vdash$. For example, the general form of $\Rightarrow$I now looks as follows if we remove the need for brackets and labels.

$$\frac{\dfrac{}{P \vdash P} \qquad \Gamma, P \vdash Q}{\Gamma \vdash P \Rightarrow Q} \Rightarrow\text{I}$$

Here, Γ is a set of zero or more assumptions appearing in the proof of $P \Rightarrow Q$.

The above example also illustrates why its okay to assume any proposition out of thin air. The rule of inference corresponding to assuming a new proposition simply asserts that the sequent $P \vdash P$ is always true for any proposition P.

5.1.1 Example Proof using Sequents

Below is a proof of the sequent $R \Rightarrow (S \Rightarrow T), S \vdash R \Rightarrow T$ using Natural Deduction and the rules of inference defined in the previous section.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{R \Rightarrow (S \Rightarrow T) \vdash R \Rightarrow (S \Rightarrow T)} \quad \dfrac{}{R \vdash R}}{R \Rightarrow (S \Rightarrow T), R \vdash S \Rightarrow T} \Rightarrow\text{E} \quad \dfrac{}{S \vdash S}}{R \Rightarrow (S \Rightarrow T), R, S \vdash T} \Rightarrow\text{E}}{R \Rightarrow (S \Rightarrow T), S \vdash R \Rightarrow T} \Rightarrow\text{I}$$

Proofs using sequents are less readable than proofs that only rely on propositions, but it is easier to automatically check that sequent-based proofs are correct.

With the help of sequents, every rule of inference that I have presented so far can almost entirely be described as terms using the operators I described in section 2.2. However, I've had to make some extensions to the definition I give for terms and the $findBinding$ operator in order to more flexibly match two sequents.

5.1.2 Representing Sequents as Terms

For the rest of this document I will refer to a second kind of variable term which can be used to represent a list of other terms. This addition was made to specify terms that match with terms representing sequents that contain more than one assumption, such as the following sequent.

$$\Gamma, P \vdash Q \Rightarrow P$$

As mentioned earlier, Γ represents a set of zero or more assumptions, so that the above sequent can represent the sequents $P \vdash Q \Rightarrow P$, $R, S, P \vdash Q \Rightarrow P$, or even $R, P, S \vdash Q \Rightarrow P$, since the order in which assumptions occur in a sequent is irrelevant. There are a countably infinite number of sequents that the above sequent represents, but these three examples are all that are required to motivate the introduction of list variables.

Using list variables, the sequent $\Gamma, P \vdash Q \Rightarrow P$ can be represented as the following term.

$$((...\Gamma_1\ \underline{\mathrm{P}}\ ...\Gamma_2) \vdash (\underline{\mathrm{Q}} \Rightarrow \underline{\mathrm{P}}))$$

Here, $...\Gamma_1$ is a list variable associated with all the terms to the left of $\underline{\mathrm{P}}$, and $...\Gamma_2$ is a list variable associated with all the terms to the right of $\underline{\mathrm{P}}$. This term would then need to match the following terms in order to preserve the original sequent's meaning.

$t_1 = ((\underline{\mathrm{P}}) \vdash (\underline{\mathrm{Q}} \Rightarrow \underline{\mathrm{P}}))$	representing the sequent $P \vdash Q \Rightarrow P$
$t_2 = ((\underline{\mathrm{R}}\ \underline{\mathrm{S}}\ \underline{\mathrm{P}}) \vdash (\underline{\mathrm{Q}} \Rightarrow \underline{\mathrm{P}}))$	representing the sequent $R, S, P \vdash Q \Rightarrow P$
$t_3 = ((\underline{\mathrm{R}}\ \underline{\mathrm{P}}\ \underline{\mathrm{S}}) \vdash (\underline{\mathrm{Q}} \Rightarrow \underline{\mathrm{P}}))$	representing the sequent $R, P, S \vdash Q \Rightarrow P$

Referring back to the definition in section 2.3.3, the term $x = ((...\Gamma_1\ \underline{\mathrm{P}}\ ...\Gamma_2) \vdash (\underline{\mathrm{Q}} \Rightarrow \underline{\mathrm{P}}))$ only matches the above three terms t_i if for each we can find a binding b_i such that $substitute(x, b_i) = t_i$. Those bindings are given as follows, where ε represents a list of zero terms.

Let v be a variable occurring in $((...\Gamma_1\ \underline{\mathrm{P}}\ ...\Gamma_2) \vdash (\underline{\mathrm{Q}} \Rightarrow \underline{\mathrm{P}}))$. A binding b_1 from these variables to a set of terms can be defined as follows.

$$b_1(v) = \begin{cases} \varepsilon & \text{if } v = ...\Gamma_1 \\ \varepsilon & \text{if } v = ...\Gamma_2 \end{cases}$$

We can verify that the existence of this binding implies that $((...\Gamma_1\ \underline{\mathrm{P}}\ ...\Gamma_2) \vdash (\underline{\mathrm{Q}} \Rightarrow \underline{\mathrm{P}}))$ matches $((\underline{\mathrm{P}}) \vdash (\underline{\mathrm{Q}} \Rightarrow \underline{\mathrm{P}}))$ as follows. The list variables $...\Gamma_1$ and $...\Gamma_2$ are to be forgotten if they are associated with the empty list of terms, ε.

$$\begin{aligned} substitute(x, b_1) &= ((b_1(...\Gamma_1)\ \underline{\mathrm{P}}\ b_1(...\Gamma_2)) \vdash (\underline{\mathrm{Q}} \Rightarrow \underline{\mathrm{P}})) \\ &= ((\varepsilon\ \underline{\mathrm{P}}\ \varepsilon) \vdash (\underline{\mathrm{Q}} \Rightarrow \underline{\mathrm{P}})) \\ &= ((\underline{\mathrm{P}}) \vdash (\underline{\mathrm{Q}} \Rightarrow \underline{\mathrm{P}})) \end{aligned}$$

Bindings whose existence imply that $((...\Gamma_1\ \underline{\mathrm{P}}\ ...\Gamma_2) \vdash (\underline{\mathrm{Q}} \Rightarrow \underline{\mathrm{P}}))$ matches t_2 and t_3 are given below.

$$b_2(v) = \begin{cases} \underline{\mathrm{R}}\ \underline{\mathrm{S}} & \text{if } v = ...\Gamma_1 \\ \varepsilon & \text{if } v = ...\Gamma_2 \end{cases}$$

Verifying that $substitute(x, b_2) = t_2$.

$$\begin{aligned} substitute(x, b_2) &= ((b_2(...\Gamma_1)\ \underline{\mathrm{P}}\ b_2(...\Gamma_2)) \vdash (\underline{\mathrm{Q}} \Rightarrow \underline{\mathrm{P}})) \\ &= ((\underline{\mathrm{R}}\ \underline{\mathrm{S}}\ \underline{\mathrm{P}}\ \varepsilon) \vdash (\underline{\mathrm{Q}} \Rightarrow \underline{\mathrm{P}})) \\ &= ((\underline{\mathrm{R}}\ \underline{\mathrm{S}}\ \underline{\mathrm{P}}) \vdash (\underline{\mathrm{Q}} \Rightarrow \underline{\mathrm{P}})) \end{aligned}$$

A binding b_3.

$$b_3(v) = \begin{cases} \underline{\text{R}} & \text{if } v = ...\Gamma_1 \\ \underline{\text{S}} & \text{if } v = ...\Gamma_2 \end{cases}$$

Verifying that $substitute(x, b_3) = t_3$.

$$\begin{aligned} substitute(x, b_3) &= ((b_3(...\Gamma_1)\ \underline{\text{P}}\ b_3(...\Gamma_2)) \vdash (\underline{\text{Q}} \Rrightarrow \underline{\text{P}})) \\ &= ((\underline{\text{R}}\ \underline{\text{P}}\ \underline{\text{S}}) \vdash (\underline{\text{Q}} \Rrightarrow \underline{\text{P}})) \end{aligned}$$

The addition of list variables also requires extensions to the definition given for the $findBinding$ operator.

5.1.3 Extending the definition of $findBinding$ to Support List Variables

The $findBinding$ operator's definition needs to updated to support list variables if it is to be used in the following examples. Unfortunately, bindings are no longer guaranteed to be unique for a given pair of terms containing list variables, and there are likely other implications of their addition I have not fully thought through. However, the examples I present in this paper only requires bindings that work for each example to exist. I supply such bindings when needed, so there is no need to use the $findBinding$ operators, and therefore no need to extend its definition to support list variables until I've had more time to think it through.

5.2 Defining Rules of Inference as Rewrite Rules

With all of these extensions in place, we can now define rules of inference as special kinds of rewrite rules. The rewrite rules presented in this section can be used to either infer a sequent from a set of premises, or to check that a given sequent follows from a set of premises for a given rule of inference. The rewrite rule for the rule of inference is different than the rules of inference for the other rewrite rules as it can only be used in the latter way. We will start with the rewrite rule for Assume to highlight this difference.

5.2.1 Assume as a Rewrite Rule

Assume is the rule of inference that allows one to infer a sequent from nothing

$$\frac{}{P \vdash P}$$

Here, P can be any proposition at all, since Assume doesn't rely on any inputs and $P \vdash P$ is a tautology for all propositions P.

The rewrite rule for Assume is as follows.

$$(\underline{\text{Assume}}) \rightarrow ((.\text{P}) \vdash .\text{P})$$

If we were to only use the operator $rewrite$ on ($\underline{\text{Assume}}$), we would only be able to infer the term $((.\text{P}) \vdash .\text{P})$, which isn't very helpful on its own.

$$\begin{aligned} rewrite((\underline{\text{Assume}}) \rightarrow ((.\text{P}) \vdash .\text{P}), (\underline{\text{Assume}})) \\ = ((.\text{P}) \vdash .\text{P}) \end{aligned}$$

This inference is only helpful if we substitute something in for the variable .P from the rewritten form of ($\underline{\text{Assume}}$) by evaluating $substitute(((.\text{P}) \vdash .\text{P}), b)$, where b is a binding.

Here, the only restriction on b is that the variable .P is in its domain, since .P is the only variable present in the result of rewriting (Assume). Beyond this restriction, b(.P) can be anything we want, which reflects our ability to assume any sequent of the form $P \vdash P$.

Let v be a variable term.

$$b(v) = \begin{cases} (\underline{\text{S}} \ \underline{\&} \ \underline{\text{T}}) & \text{if } v = .\text{P} \end{cases}$$

We can then plug this binding into the *substitute* operator along with the term resulting from $rewrite((\underline{\text{Assume}}) \rightarrow ((.\text{P}) \ \underline{\vdash} \ .\text{P}), (\underline{\text{Assume}}))$ to infer the term $(((\underline{\text{S}} \ \underline{\&} \ \underline{\text{T}})) \ \underline{\vdash} \ (\underline{\text{S}} \ \underline{\&} \ \underline{\text{T}}))$.

$$substitute(((.\text{P}) \ \underline{\vdash} \ .\text{P}), b) = (((\underline{\text{S}} \ \underline{\&} \ \underline{\text{T}})) \ \underline{\vdash} \ (\underline{\text{S}} \ \underline{\&} \ \underline{\text{T}}))$$

This corresponds to the following proof using sequents.

$$\frac{}{S \ \& \ T \vdash S \ \& \ T}$$

Our ability to choose any binding b that has the variable .P in its domain when calling $substitute(((.\text{P}) \ \underline{\vdash} \ .\text{P}), b)$ shows that any term matching $((.\text{P}) \ \underline{\vdash} \ .\text{P})$ can also be inferred from (Assume). Using substitute to infer terms beyond what is returned by the rewrite rules for the other rules of inference presented in this paper is redundant, as none of the other rewrite rules $l \rightarrow r$ contain variables in l not present in r.

5.2.2 ⇒I as a Rewrite Rule

The rule of inference ⇒I allows us to infer a sequent whose conclusion is of the form $P \Rightarrow Q$, where P has been assumed earlier in the proof and Q has been concluded earlier in the proof.

$$\frac{\dfrac{}{P \vdash P} \qquad \Gamma, P \vdash Q}{\Gamma \vdash P \Rightarrow Q} \Rightarrow\text{I}$$

The rewrite rule representing ⇒I needs to be able to discharge the assumed premise from the resulting sequent. Below, the list variables $...\Gamma_1$ and $...\Gamma_2$ allow us to do this by bookending the variable .P, which is associated with the term being assumed. The rewrite rule's resulting term "forgets" about the assumed term associated with .P on the left hand side of its $\underline{\vdash}$ by reconstructing the list of assumptions as $(...\Gamma_1 \ ...\Gamma_2)$ instead of $(...\Gamma_1 \ .\text{P} \ ...\Gamma_2)$.

$$(\underline{\Rightarrow}\text{I} \ ((.\text{P}) \ \underline{\vdash} \ .\text{P}) \ ((... \ \Gamma_1 \ .\text{P} \ ...\Gamma_2) \ \underline{\vdash} \ .\text{Q})) \rightarrow ((...\Gamma_1 \ ...\Gamma_2) \ \underline{\vdash} \ (.\text{P} \ \underline{\Rightarrow} \ .\text{Q}))$$

Just like in 5.2.1, we can use *rewrite* to infer that the term representing the sequent $R, S \vdash P \Rightarrow T$ from terms representing the sequents $P \vdash P$ and $P, R, S \vdash T$.

For example, if we wanted to prove that the sequent $R, S \vdash Q \Rightarrow T$ is true using ⇒I, we would start by representing our desired inference as a proof tree.

$$\frac{\dfrac{}{Q \vdash Q} \qquad \dfrac{}{Q, R, S \vdash T}}{R, S \vdash Q \Rightarrow T} \Rightarrow\text{I}$$

We then want to convert the involved sequents into terms so that we can later plug them into our *rewrite* operator.

$$Q \vdash Q \text{ becomes } ((\underline{\text{Q}}) \ \underline{\vdash} \ \underline{\text{Q}}),$$
$$Q, R, S \vdash T \text{ becomes } ((\underline{\text{Q}} \ \underline{\text{R}} \ \underline{\text{S}}) \ \underline{\vdash} \ \underline{\text{T}}),$$
$$R, S \vdash Q \Rightarrow T \text{ becomes } ((\underline{\text{R}} \ \underline{\text{S}}) \ \underline{\vdash} \ (\underline{\text{Q}} \ \underline{\Rightarrow} \ \underline{\text{T}}))$$

Our goal now is to use the terms $((\underline{\mathrm{Q}}) \vdash \underline{\mathrm{Q}})$ and $((\underline{\mathrm{Q}}\ \underline{\mathrm{R}}\ \underline{\mathrm{S}}) \vdash \underline{\mathrm{T}})$ to infer the term $((\underline{\mathrm{R}}\ \underline{\mathrm{S}}) \vdash \underline{\mathrm{Q}} \Rightarrow \underline{\mathrm{T}})$ using the rewrite rule for $\Rightarrow$I, which was introduced at the beginning of this section. To use this rule, we must form a term by combining the premise sequents such that the left hand side of the rewrite rule matches our new term. I claim that the following term is such a term.

$$(\underline{\Rightarrow}\mathrm{I}\ ((\underline{\mathrm{Q}}) \vdash \underline{\mathrm{Q}})\ ((\underline{\mathrm{Q}}\ \underline{\mathrm{R}}\ \underline{\mathrm{S}}) \vdash \underline{\mathrm{T}}))$$

Going back to the definition of *match* in section 2.3.3, a term x matches a term t if and only if there exists a function b such that $substitute(x, b) = t$. This function can be defined as follows.

Let v be a variable term.

$$b(v) = \begin{cases} \underline{\mathrm{Q}} & \text{if } v = .\mathrm{P} \\ \varepsilon & \text{if } v = ...\Gamma_1 \\ \underline{\mathrm{R}}\ \underline{\mathrm{S}} & \text{if } v = ...\Gamma_2 \\ \underline{\mathrm{T}} & \text{if } v = .\mathrm{Q} \end{cases}$$

Plugging $(\underline{\Rightarrow}\mathrm{I}\ ((.\mathrm{P}) \vdash .\mathrm{P})\ ((...\Gamma_1\ .\mathrm{P}\ ...\Gamma_2) \vdash .\mathrm{Q}))$ and b into the *substitute* operator will give us our desired term.

$$\begin{aligned} &substitute((\underline{\Rightarrow}\mathrm{I}\ ((.\mathrm{P}) \vdash .\mathrm{P})\ ((...\Gamma_1\ .\mathrm{P}\ ...\Gamma_2) \vdash .\mathrm{Q})), b) \\ &\quad = (\underline{\Rightarrow}\mathrm{I}\ ((b(.\mathrm{P})) \vdash b(.\mathrm{P}))\ ((b(...\Gamma_1)\ b(.\mathrm{P})\ b(...\Gamma_2)) \vdash b(.\mathrm{Q}))) \\ &\quad = (\underline{\Rightarrow}\mathrm{I}\ ((\underline{\mathrm{Q}}) \vdash \underline{\mathrm{Q}})\ ((\underline{\mathrm{Q}}\ \underline{\mathrm{R}}\ \underline{\mathrm{S}}) \vdash \underline{\mathrm{T}})) \end{aligned}$$

Therefore, the left hand side of the rewrite rule for $\Rightarrow$I matches $(\underline{\Rightarrow}\mathrm{I}\ ((\underline{\mathrm{Q}}) \vdash \underline{\mathrm{Q}})\ ((\underline{\mathrm{Q}}\ \underline{\mathrm{R}}\ \underline{\mathrm{S}}) \vdash \underline{\mathrm{T}}))$ so that we can use it to infer the term $((\underline{\mathrm{R}}\ \underline{\mathrm{S}}) \vdash (\underline{\mathrm{Q}} \Rightarrow \underline{\mathrm{T}}))$.

$$\begin{aligned} &rewrite((\underline{\Rightarrow}\mathrm{I}\ ((.\mathrm{P}) \vdash .\mathrm{P})\ ((...\Gamma_1\ .\mathrm{P}\ ...\Gamma 2) \vdash .\mathrm{Q})) \rightarrow ((...\Gamma_1\ ...\Gamma_2) \vdash (.\mathrm{P} \underline{\Rightarrow} .\mathrm{Q})), \\ &\qquad (\underline{\Rightarrow}\mathrm{I}\ ((\underline{\mathrm{Q}}) \vdash \underline{\mathrm{Q}})\ ((\underline{\mathrm{Q}}\ \underline{\mathrm{R}}\ \underline{\mathrm{S}}) \vdash \underline{\mathrm{T}}))) \\ &\qquad = substitute(((...\Gamma_1\ ...\Gamma_2) \vdash (.\mathrm{P} \underline{\Rightarrow} .\mathrm{Q})), b) \\ &\qquad = ((b(...\Gamma_1)\ b(...\Gamma_2)) \vdash (b(.\mathrm{P}) \underline{\Rightarrow} b(.\mathrm{Q}))) \\ &\qquad = ((\varepsilon\ \underline{\mathrm{R}}\ \underline{\mathrm{S}}) \vdash (\underline{\mathrm{Q}} \Rightarrow \underline{\mathrm{T}})) \\ &\qquad = ((\underline{\mathrm{R}}\ \underline{\mathrm{S}}) \vdash (\underline{\mathrm{Q}} \Rightarrow \underline{\mathrm{T}})) \end{aligned}$$

The four other rules of inference presented in this paper can be converted to terms and rewrite rules using the same process used for $\Rightarrow$I .

5.2.3 $\Rightarrow$E as a Rewrite Rule

The rule of inference $\Rightarrow$E allows us to infer a sequent whose conclusion is Q if given a sequent whose conclusion is $P \Rightarrow Q$ as its major premise, and a sequent whose conclusion is P as its minor premise.

$$\frac{\Gamma_1 \vdash P \Rightarrow Q \qquad \Gamma_2 \vdash P}{\Gamma_1, \Gamma_2 \vdash Q} \Rightarrow\mathrm{E}$$

The resulting sequent must have as its assumptions all the assumptions of both its major and minor premises.

We can construct this resulting set of assumptions by taking the list of terms representing the major premise's assumptions, $(...\Gamma_1)$, and combining them with the list of terms representing the minor premise's assumptions, $(...\Gamma_2)$, resulting in a new list of terms, $(...\Gamma_1\ ...\Gamma_2)$.

$$(\underline{\Rightarrow \text{E}}\ ((...\Gamma_1) \vdash (.\text{P} \underline{\Rightarrow}\ .\text{Q}))\ ((...\Gamma_2) \vdash .\text{P})) \rightarrow ((...\Gamma_1\ ...\Gamma_2) \vdash .\text{Q})$$

In this section we will infer the sequent $P \Rightarrow Q, P \vdash Q$ from the premises $P \Rightarrow Q \vdash P \Rightarrow Q$ and $P \vdash P$.

$$\frac{\overline{P \Rightarrow Q \vdash P \Rightarrow Q} \qquad \overline{P \vdash P}}{P \Rightarrow Q, P \vdash Q} \Rightarrow\text{E}$$

We start by converting the three involved sequents into terms.

$$\begin{aligned} P \Rightarrow Q \vdash P \Rightarrow Q \text{ becomes } (((\underline{\text{P}} \underline{\Rightarrow} \underline{\text{Q}})) \vdash (\underline{\text{P}} \underline{\Rightarrow} \underline{\text{Q}})), \\ P \vdash P \text{ becomes } ((\underline{\text{P}}) \vdash \underline{\text{P}}), \\ P \Rightarrow Q, P \vdash Q \text{ becomes } (((\underline{\text{P}} \underline{\Rightarrow} \underline{\text{Q}})\ \underline{\text{P}}) \vdash \underline{\text{Q}}) \end{aligned}$$

Then, we show that the term $t = (\underline{\Rightarrow \text{E}}\ (((\underline{\text{P}} \underline{\Rightarrow} \underline{\text{Q}})) \vdash (\underline{\text{P}} \underline{\Rightarrow} \underline{\text{Q}}))\ ((\underline{\text{P}}) \vdash \underline{\text{P}}))$ matches the left hand side of the rewrite rule for $\Rightarrow$E, which we'll call x, by finding a binding b such that $substitute(x, b) = t$.

$$b(v) = \begin{cases} (\underline{\text{P}} \Rightarrow \underline{\text{Q}}) & \text{if } v = ...\Gamma_1 \\ \underline{\text{P}} & \text{if } v = .\text{P} \\ \underline{\text{P}} & \text{if } v = ...\Gamma_2 \\ \underline{\text{Q}} & \text{if } v = .\text{Q} \end{cases}$$

Verifying that we have found a binding b such that the left hand side of the rewrite rule for $\Rightarrow$E matches our term t can be done as follows.

$$\begin{aligned} &substitute((\underline{\Rightarrow \text{E}}\ ((...\Gamma_1) \vdash (.\text{P} \underline{\Rightarrow}\ .\text{Q}))\ ((...\Gamma_2) \vdash .\text{P})), b) \\ &\quad = (\underline{\Rightarrow \text{E}}\ ((b(...\Gamma_1)) \vdash (b(.\text{P}) \underline{\Rightarrow}\ b(.\text{Q})))\ ((b(...\Gamma_2)) \vdash b(.\text{P}))) \\ &\quad = (\underline{\Rightarrow \text{E}}\ (((\underline{\text{P}} \underline{\Rightarrow} \underline{\text{Q}})) \vdash (\text{P} \underline{\Rightarrow} \text{Q}))\ ((\text{P}) \vdash \text{P})) \end{aligned}$$

Finally, we can use the rewrite rule for $\Rightarrow$E to infer the term $(((\underline{\text{P}} \underline{\Rightarrow} \underline{\text{Q}})\ \underline{\text{P}}) \vdash \underline{\text{Q}})$.

$$\begin{aligned} &rewrite((\underline{\Rightarrow \text{E}}\ ((...\Gamma_1) \vdash (.\text{P} \Rightarrow .\text{Q}))\ ((...\Gamma_2) \vdash .\text{P})) \rightarrow ((...\Gamma_1\ ...\Gamma_2) \vdash .\text{Q}), \\ &\qquad (\underline{\Rightarrow \text{E}}\ (((\underline{\text{P}} \underline{\Rightarrow} \underline{\text{Q}})) \vdash (\underline{\text{P}} \underline{\Rightarrow} \underline{\text{Q}}))\ ((\underline{\text{P}}) \vdash \underline{\text{P}}))) \\ &\qquad = substitute(((...\Gamma_1\ ...\Gamma_2) \vdash .\text{Q}), b) \\ &\qquad = ((b(...\Gamma_1)\ b(...\Gamma_2)) \vdash b(.\text{Q})) \\ &\qquad = (((\underline{\text{P}} \underline{\Rightarrow} \underline{\text{Q}})\ \underline{\text{P}}) \vdash \underline{\text{Q}}) \end{aligned}$$

5.2.4 &I and &E as Rewrite Rules

The rule of inference &I takes two sequents and produces a sequent whose assumptions are the assumptions of its premise sequents and whose conclusion is of the form $P\ \&\ Q$, where P is the conclusion of the first premise and Q is the conclusion of the second premise.

$$\frac{\Gamma_1 \vdash \text{P} \qquad \Gamma_2 \vdash \text{Q}}{\Gamma_1, \Gamma_2 \vdash P\ \&\ Q}\ \&\text{I}$$

The rewrite rule for &I is similar to the rewrite rule for $\Rightarrow$E, as the assumptions of its premises are combined in the resulting term sequent.

$$(\underline{\&\text{I}}\ ((...\Gamma_1) \vdash .\text{P})\ ((...\Gamma_2) \vdash .\text{Q})) \rightarrow ((...\Gamma_1\ ...\Gamma_2) \vdash (.\text{P}\ \underline{\&}\ .\text{Q}))$$

The rule of inference for &E comes in two forms, written as $\&\text{E}_1$ to infer the first first operand and $\&\text{E}_2$ to infer the second operand.

$$\frac{\Gamma \vdash (P\ \&\ Q)}{P}\ \&\text{E}_1$$

$$\frac{\Gamma \vdash (P\ \&\ Q)}{Q}\ \&\text{E}_2$$

$$(\underline{\&\text{E}_1}\ ((...\Gamma) \vdash (.\text{P}\ \underline{\&}\ .\text{Q}))) \rightarrow ((...\Gamma) \vdash .\text{P})$$
$$(\underline{\&\text{E}_2}\ ((...\Gamma) \vdash (.\text{P}\ \underline{\&}\ .\text{Q}))) \rightarrow ((...\Gamma) \vdash .\text{Q})$$

The examples given in the previous three sections all follow the same expository pattern, a pattern which might overstay its welcome if continued for introducing &I and &E as rewrite rules. With that in mind, I leave the work of verifying that these rewrite rules do, in fact accomplish what their corresponding rules of inference accomplish as an exercise to the reader.

6 Conclusion

This capstone has given me the knowledge and confidence necessary to implementing the application I first started dreaming of in 2017, while giving me the curiosity and motivation to produce software that goes far beyond my initial goals. Along with the work I present in 5, I also hoped to include my initial design for a Natural Deduction proof-checker and proof builder in this report. However, I kept running into issues on how to present the material, since my design relies on concepts I learned as a computer-science major that requires more exposition than I have time to write or space for in this paper. Section 5 was a compromise, as it allowed me to present some of the basics of how I planned to represent proofs in such a system using the concepts I had already presented in 2.

One of the goals I set out for myself at the beginning of the semester was to formalize the work I had done with Shoreline so that I could present it to a mathematical audience. I spent the first part of the semester familiarizing myself with Term Rewrite Systems in order to achieve this goal. The second goal I set for myself was to start proving simple statements using the extended version of Shoreline, which I called Coastline. I learned about Natural Deduction and sequents in order to get closer to this goal, but never defined Coastline well enough to realize it. Despite this shortcoming, I am happy with what I was able to achieve during Math 4020.

I plan on continuing this project in some form during the Spring 2020 semester, and Professor Fitelson has agreed to continue helping me with it. We still haven't sketched out an exact plan for this continued work, but I know I will be taking a closer look at condensed detachment, unification, and sequents. Between now and then I plan to start programming the proof builder/checker that I spent the last part of Math 4020 designing. I would then use this prototype to more quickly try out ideas I have for Coastline.

7 Acknowledgements

I am grateful for the professor's and students who gave me feedback on my project and encouraged me to continue working on it.

Professor Anthony Iarrobino, my professor for Math 4020, has been very encouraging throughout the semester. He suggested that I attend Applications of Computer Algebra in 2019, and

the experience ended up being extremely positive. I am very lucky to have been his student for two semesters in a row.

Professor Branden Fitelson acted as a consultant for my project, and provided invaluable feedback throughout the semester. I am grateful for the resources he supplied, advice he gave, and the many conversations we had both relevant to and entirely separate from my project. I am looking forward to continue working with him next semester.

Professor Lee-Peng Lee and Professor Stanley Eigen were very helpful while I was preparing my poster for Applications of Computer Algebra and while I was trying to come up with a proposal once the semester had begun. They helped me think through the motivation behind my project and gave suggestions on how to improve my poster.

I met Professor William Bauldry of Appalachian State University and Professor Wade Ellis of West Valley College while attending Applications of Computer Algebra. We met while I was presenting my poster, and I continued speaking with them throughout the conference. They both expressed interest in Shoreline and encouraged me to continue working on it. My conversations with them and other attendees at the conference further motivated me to continue working on Shoreline.

The feedback given by other Math 4020 students was also very helpful. If this paper is at all coherent it is because of corrections and suggestions given by everyone else in the class. The work they did on their own projects further motivated me to continue mine.

References

[1] Adjorlolo, Koissi. "Spring 2019 write-up on Shoreline" https://koissiadjorlolodotnet.wordpress.com/2019/08/31/spring-2019-write-up-on-shoreline/

[2] Baader, Franz, and Tobias Nipkow. Term rewriting and all that. Cambridge university press, 1999.

[3] Forbes, Graeme. "Modern logic: A text in elementary symbolic logic." (1994).

[4] Wadler, Philip. "Propositions as types." Commun. ACM 58.12 (2015): 75-84.

[5] Meyer, Robert K., Martin W. Bunder, and Lawrence Powers. "Implementing the 'Fool's model' of combinatory logic." Journal of Automated Reasoning 7.4 (1991): 597-630.

[6] Kalman, J. A. "Condensed detachment as a rule of inference." Studia Logica 42.4 (1983): 443-451.