12-2008

Two Categories of Refutation Decision Procedures for Classical and Intuitionistic Propositional Logic

Edward Doyle
Clemson University, tdoyle@cs.clemson.edu

TWO CATEGORIES OF REFUTATION DECISION PROCEDURES FOR CLASSICAL AND INTUITIONISTIC PROPOSITIONAL LOGIC

A Dissertation
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosphy
Computer Science

---

by
Edward J. Doyle II
December 2008

Accepted by:
Dr. D. E. Stevenson, Committee Chair
Dr. Steve T. Hedetniemi
Dr. David P. Jacobs
Dr. Daniel D. Warner

Abstract

An automatic theorem prover is a computer program that proves theorems without the assistance of a human being. Theorem proving is an important basic tool in proving theorems in mathematics, establishing the correctness of computer programs, proving the correctness of communication protocols, and verifying integrated circuit designs.

This dissertation introduces two new categories of theorem provers, one for classical propositional logic and another for intuitionistic propositional logic. For each logic a container property and generalized algorithm are introduced.

Many methods have been developed over the years to prove theorems in propositional logic. This dissertation describes and presents example proofs for five of these methods: natural deduction, Kripke tableau, analytic tableau, matrix, and resolution. Each of these methods uses refutation to prove a theorem. In refutation, the proposed theorem is assumed to be false. The theorem prover is successful, only if the analysis of this assumption leads to a contradiction.

Each of these methods, except resolution, share a common algorithm. To prove this, the container is introduced. A data structure used by a method is a *container*, if it meets a set of properties.

A generalized algorithm that proves theorems is introduced. Since each step in this algorithm uses only operations that are provided by the container. The steps it performs

can be translated to any method that can be described using a container. This allows the data structures representing a partial proof in one method, to be transformed into the data structures representing the "same" proof in another method. This can be very beneficial in a situation where another method would be more efficient in advancing the proof. In addition to being able to switch between methods, an heuristic for one method can be examined to see if it can be applied to the other methods.

This development is repeated for intuitionistic logic. Each of these methods, except resolution, is modified to prove theorems in intuitionistic logic. An intuitionistic container is presented. Each one of the intuitionistic methods is proven to have the properties of the intuitionistic container. Lastly, a generalized algorithm using the intuitionistic container is presented. This algorithm proves theorems in intuitionistic logic. Examples showing successful and unsuccessful proof attempts are presented.

Acknowledgements

I wish to express my appreciation to my first teachers, my parents, Gregory Doyle, Ph.D. and Mary Kaye Doyle. My supportive adviser Dr. D. E. Stevenson and the rest of my committee. Barbara Ramirez for her help with the editing of this dissertation. And Dr. Steve Hedetniemi his very careful reading and large number of helpful comments on an earlier version of this dissertation. However, I am responsible for any errors in this dissertation.

Frequently Used Notation and Functions

Logical Connectives

$\wedge$	conjunction	$\Rightarrow$	Implication
$\vee$	disjunction	$\neg$	Negation

- The Greek letters α and β are used for Smullyan's rule types. Smullyan uses γ and δ for rule types in first order logic.
- The Greek letters φ, ψ, and ρ, represent arbitrary formulas and signed formulas.
- The capital Greek letters Γ and Δ represent sets of formulas. These letter are used in natural deduction sequents.
- E is almost always used to represent a formula or signed formula. When more than one formula is needed, a subscript or a tick mark is added (i.e. E'). The choice of E was used to stand for expression. During the writing of this dissertation a decision was made the replace the term expression with formula to match the other literature. Since F is commonly used a to represent predicates or functions, a decision was made to stick with E.
- The lower case letter b represents either a branch in an analytic tableau or a box in a Kripke tableau.

- The lower case letters p and q are used for partial order elements used with intuitionistic analytic tableau. Also p is used to represent a path in a matrix.
- SM_r maps a signed formula to the Smullyan rule type (α or β).
- SM_c maps a signed formula to its child formula(s).
- KR_r maps a column-expression pair in a Kripke box to the expansion rule type $split$ or $no\ split$.
- KR_c maps a column-expression pair E in a Kripke box to the set of column-expression pairs that are generated by expanding E.
- $L(v)$ maps a vertex v in a tree to the leaf vertices that are descendants of v.
- $K2C$ maps a set Kripke column-formula pairs to a set of signed formulas.
- $B2C$ maps a branch in an analytic tableau to the set of signed formulas that label its vertices.
- $P2C$ maps a path in a matrix to the set of signed formulas that label its vertices.
- $S2C$ maps a sequent, represented as a set of sequent side formulas, to a set of signed formulas.
- $IB2C$ Intuitionistic branch to container. Maps an element of the partial order and a branch in an intuitionistic tableau to a set of signed formulas. $IB2C(b, p)$ is the set of signed formulas on branch b with either partial element p or elements preceding it in the partial order. Intuitionistic version of $B2C$.
- $P(v)$ maps a vertex in an intuitionistic analytic tableau to the partial order element labeling it.

- SF_n maps a sequent side formula to signed formula.
- SF_k map a column-formula pair in a Kripke box to a signed formula.
- $J(E)$ maps a signed formula to $present$, $some\_future$, or all_future. This function is used with SM_r and SM_c to describe the semantics of intuitionistic logic.

Table of Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

Automatic theorem proving (ATP) is used in many areas such as proving theorems in mathematics, establishing that computer programs meet their specifications, proving the properties of communication protocols, and verifying integrated circuit designs. The behavior of a system or object is reduced to logical formulas that are fed to an ATP, which then proves that the system functions as desired. Typically, the logical formulas that describe these systems are large, involving hundreds or thousands of variables. Thus, efficient procedures must to be used to prove or disprove theorems in a reasonable period of time.

1.1 The Development of Logic

Logic was developed in ancient Greece to assist philosophers in constructing and analyzing arguments. The origin of formal logic is credited to Aristotle [Smith 2007], who made many contributions including the notion of deduction. Deduction is the process of breaking an argument down into a chain of reasoning, where each step leads to the next by applying a simple logical rule. The logic of Aristotle was used extensively until the Nine-

teenth Century when mathematicians increased the rigor of their research and encountered the limits of Aristotle's logic.

George Boole's "Mathematical Analysis of Logic" (1847) and his later work "An Investigation of the Laws of Thought, on which are founded the Mathematical Theory of Logic and Probabilities" (1854) [Boole 1854] were very influential in defining the semantics of classical propositional logic, but the semantics that Boole used were different. In his work, Boole developed a system which allowed logical formulas to be treated as if they were numerical formulas. Boole viewed logical addition (disjunction) as the union of two disjoint sets, logical multiplication (conjunction) as the intersection of two sets, and the logical complement of x as $I - x$, abbreviated as $\overline{x}$, where I is the universe under discussion and o (denoted with the lowercase letter o) is the empty set. His semantics fit well into the computation of probability, where I is treated as one and o is treated as zero. The semantics used in propositional logic restrict Boolean variables to one of two values, namely treating I as true and o as false [Kneale and Kneale 1962][pages 404-420].

Frege in 1879 created a symbolic language which included an explicit, finite closed set of proof rules [von Plato 2008]. Frege's formulation of a proof system is in itself a mathematical object, thus subject to Gödel's completeness and incompleteness theorems [Gödel 1967].

One of Frege's contemporaries, Peirce, published a paper in 1883, which added subscripts to relations. For instance, the relation l_{ij} might indicate the individual i likes individual j. In this same paper, he also introduced the notion of the quantifiers "for all" and "there exists" which range over the sets of individuals. He used $\prod$ and $\sum$ to represent these quantifiers because he intended for them to represent logical multiplication (conjunction) and logical addition (disjunction) over a set of individuals[Kneale and Kneale 1962][page 430]. Today, these quantifiers are represented by the symbols $\forall$ and $\exists$, respectively.

Intuitionistic logic [Moschovakis 2007] was introduced by Brouwer and others as a basis for constructive mathematics. Classical logic assumes that every proposition has a known truth value, but intuitionistic logic allows for the truth value of a formula to be unknown. The principal difference between intuitionistic and classical logic is that intuitionistic logic rejects the excluded middle ($\varphi \vee \neg\varphi$). While the excluded middle is perfectly valid when applied to reasoning about a finite number of possibilities, it is unsound when extended to infinite domains. Brouwer's objections were later justified by Gödel's incompleteness theorem [Gödel 1967].

1.2 The Development of Automated Theorem Provers

As soon as the digital computer was developed, people started to program automated theorem provers. One of these early efforts is described in the following quote: "...in 1947 an electrical computer ... was constructed at Harvard by T. A. Kalin and W. Burkhart specially for the solution of Boolean problems involving up to twelve logical variables (i.e. propositional or class letters)"[Kneale and Kneale 1962][page 421].

The first experimental theorem proving programs for general purpose computers dates back to the 1950's. The early work included proving established theorems in mathematics and logic. It was soon recognized that Skolem functions and what later came be called the Herbrand universe were key tools for solving first-order theorems. Early theorem provers tried most, if not all, possibilities (aka the British Museum search) including implementing truth tables to test all possible assignments of truth values to variables [Davis 2001].

These provers included a semantic tableau method implementation by D. Prawitz, H. Prawitz, and Voghera[Prawitz et al. 1960] and the Davis-Putnum procedure [Davis and Putnam 1960]. There are many other distinct procedures and variants of these procedures.

The most popular of these is the resolution method, named for the resolution rule [Robinson 1965]. Today, the resolution method is the most commonly used automated theorem proving method.

1.3 Categories

A category [Pierce 1991] is a collection of objects and morphisms (sometimes called arrows). Each morphism has a domain of objects and a codomaim (range) such that morphism maps objects to. The morphism $f : A \rightarrow B$ indicates that f has a domain A and a codomian B. These morphisms have the following properties: 1) There is an identity function that maps an object to itself. 2) The morphisms can be composed that has the same effect as apply one followed by the second. 3) For any three morphisms, f, g, and h, not necessarily distinct $f : A \rightarrow B$, $g : B \rightarrow C$, and $h : C \rightarrow D$ such that $f \circ (g \circ h) = (f \circ g) \circ h$.

Categories are used in mathematics, theoretical computer science, and applied physics [Marquis 2007]. Some examples of categories are semi-groups, groups, fields, NP-complete problems, and partially ordered sets. This dissertation establishes two categories of theorem proving methods, one for classical propositional logic and another for intuitionistic propositional logic.

1.4 Categories of Theorem Provers

Proving the completeness of a proof method is an important result. Completeness means that if there exists a proof for a given formula, then the method will find it in a finite period of time. Many texts on logic show that a method is complete with respect to a logic and, thus, is equivalent to all other complete methods for that logic. For this

dissertation, completeness of these methods is not enough. This dissertation seeks a deeper understanding of the various methods, finding commonalities between the procedural steps used by each.

Several ATP methods are used by computers. In this dissertation, five of these methods are examined: natural deduction, Kripke tableau, analytic tableau, resolution, and matrix. For both classical and intuitionistic logic, a container is defined. The container's properties capture the important commonalities of four of the five methods. The properties of a container are used to construct a generalized theorem proving algorithm. This generalized algorithm is at the core of all of the methods except resolution. A different container and generalized algorithm are used for classical and intuitionistic logic.

A fifth method, resolution, is also discussed for classical logic, but does not use a data structure that meets the properties of the classical container. The resolution method as described by Robinson [Robinson 1965] cannot be used in intuitionistic logic since in general formulas in intuitionistic logic cannot be represented in conjunctive normal form. The inverse method (also called resolution caluci) makes modifications to resolution to support intuitionistic logic [Tammet 1996], [Mints 1994] and [Degtyarev and Voronkov 2001]. The inverse method attempts to construct the proposed theorem by starting with axioms and attempting to construct the proposed theorem. This approach is opposite to that taken by containers which start with the proposed theorem and break it down into smaller expressions in an attempt to find a contradiction. Hence, the inverse method is not discussed in this dissertation. Although resolution in classical logic does not meet the requirements of the container, a transformation between it and the matrix method is presented.

For both classical logic and intuitionistic logic, a container is defined, and each of the four methods is proven to use a data structure that has the properties of the container. A *container method* is a method that uses a data structure that is a container.

The properties of a container guarantee that at points during a proof, a partial proof in one container method can be transformed into a partial proof in another container method, preserving all of the proof steps made so far. There are test results[Tammet 1996] showing that for some problems resolution is faster than analytic tableau and for other problems analytic tableau is faster than resolution. If situations can be identified where one method is slower than another, then these results allow the partial proof to be transformed into the faster method and continued. A transformation between methods may be made several times during the proof of a formula. In addition, this deep level of understanding allows a heuristic or optimization developed for one container method to be more easily evaluated for its usefulness in other container methods.

The second chapter presents the terminology and notation that will be used in the remainder of the dissertation. The third chapter describes each of the five ATP methods used in classical logic. The fourth chapter introduces the concept of a container for classical logic and proves that each method except resolution has the properties of the container. The generalized proof algorithm for classical logic is introduced; this algorithm's operations use only the properties of a container to prove theorems and shows that the four methods share a common algorithm at their core. The fifth chapter begins by introducing intuitionistic logic and giving a brief description of how it differs from classical logic. For each classical method, except resolution, modifications are described that allow these methods to be used in intuitionistic logic. The intuitionistic container is introduced, and each of the intuitionistic methods is proven to use a data structure having the properties of the intuitionistic container. The chapter ends with the presentation and description of a generalized algorithm for intuitionistic logic. This algorithm uses only properties of the intuitionistic container. The sixth chapter concludes the dissertation by reviewing the material in the previous chapters and presenting areas for future work.

The dissertation includes an ML program that implements Kripke's method for intuitionistic logic. To save paper, it is available only in electronic form. It is also available from the author upon request.

Chapter 2

Background

2.1 Propositional Logic

Propositional logic explores the truth of formulas composed of logic constants, variables, and connectives. The two logical constants are true and false, denoted by $\top$ and $\bot$, respectively. Propositional variables represent statements that may be true or false. *Atomic formulas* consist of propositional variables and logical constants, and, thus, they cannot be broken down any further; i.e., they contain no connectives. The four connectives consist of the unary connective, negation ($\neg$), and the binary connectives, disjunction ($\vee$), conjunction ($\wedge$), and implication ($\Rightarrow$).

A *(logical) formula* ρ is inductively defined:

1. Base Cases
   (a) a propositional variable
   (b) a logical constant, i.e. $\top$ or $\bot$
2. General cases where φ and ψ are logical formulas
   (a) $(\varphi \wedge \psi)$

(b) $(\varphi \vee \psi)$

(c) $(\varphi \Rightarrow \psi)$

(d) $(\neg\varphi)$

In a logical formula, parentheses are used to make a formula unambiguous by indicating the structure of its underlying parse tree. It is assumed that the reader is familiar with parse trees as described in [Aho et al. 2006].

The *primary connective* of a non-atomic formula φ is the connective at the root of the formula's parse tree. In the inductive definition above, the primary connective is the last connective added in the construction of the formula.

2.2 Formula Notations

Formulas Using Only Connectives	**Signed Formulas**
φ	$\langle T,\ \varphi\rangle$
$\neg\varphi$	$\langle F,\ \varphi\rangle$
$\neg\neg\varphi$	$\langle F,\ \neg\varphi\rangle$
$\varphi \vee \psi$	$\langle T,\ \varphi \vee \psi\rangle$
$\neg(\varphi \vee \psi)$	$\langle F,\ \varphi \vee \psi\rangle$
$\varphi \Rightarrow \psi$	$\langle T,\ \varphi \Rightarrow \psi\rangle$
$\neg(\varphi \Rightarrow \psi)$	$\langle F,\ \varphi \Rightarrow \psi\rangle$

Table 2.1: Each row in this table contains an unsigned formula in the left column and the equivalent signed formula in the right column. Unsigned formulas use only connectives, while signed formulas consist of a sign, T or F, and a formula.

There are two notations used in this dissertation to represent formulas, unsigned and signed. Unsigned formulas use only logical connectives, while a formula in *signed notation*, $\langle S, E\rangle$, is represented by an ordered pair consisting of a sign S (T or F) and a formula E. In signed notation, angle brackets are used in running text to prevent confusion,

A	B	$A \wedge B$
$\bot$	$\bot$	$\bot$
$\bot$	$\top$	$\bot$
$\top$	$\bot$	$\bot$
$\top$	$\top$	$\top$

A	B	$A \vee B$
$\bot$	$\bot$	$\bot$
$\bot$	$\top$	$\top$
$\top$	$\bot$	$\top$
$\top$	$\top$	$\top$

A	B	$A \Rightarrow B$
$\bot$	$\bot$	$\top$
$\bot$	$\top$	$\top$
$\top$	$\bot$	$\bot$
$\top$	$\top$	$\top$

A	$\neg A$
$\bot$	$\top$
$\top$	$\bot$

Table 2.2: Truth tables for the basic connectives.

and are not typically used in figures and tables. In unsigned notation, a negation operator is added to the formula (i.e.$\neg\varphi$) to indicate that the formula φ is false, while the unmodified formula (φ) indicates that it is true. Adding negation connectives to subformulas yields correct results in classical logic, however, it can lead to incorrect results in other logics, such as intuitionistic. In classical logic, the signed formula $\langle T, \varphi\rangle$ signifies that φ is true, while $\langle F, \varphi\rangle$ signifies that φ is false. Table 2.1 shows examples of these two notations.

When a formula is determined to be true or false, it leads to conclusion(s) about the truth or falsity of its subformula(s). For example, in non-signed notation if it is determined that $\varphi \vee \psi$ is false, denoted by $\neg(\varphi \vee \psi)$, then in the next step of the analysis, it can inferred from this formula that both φ and ψ must be false. Hence, the formulas $\neg\varphi$ and $\neg\psi$ are added to the proof. If the analysis leads to the conclusion that a formula should evaluate to false, a $\neg$ (negation operator) is added. Repeating this analysis in signed notation, if it is known that $\varphi \vee \psi$ is false, denoted by $\langle F, \varphi \vee \psi\rangle$, then in the next step of the analysis, it can be inferred from this formula that both φ and ψ must be false. Hence, the formulas $\langle F, \varphi\rangle$ and $\langle F, \psi\rangle$ are added to the proof. Smullyan's eight rules, as shown in Table 2.4, are sufficient to analyze any classical propositional logic formula using signed formulas.

2.3 Classic Truth Determined by Truth Tables

$\mathbf{A}$	$\mathbf{B}$	$\neg\mathbf{A}$	$\mathbf{A} \Rightarrow \mathbf{B}$	$\neg\mathbf{A} \vee \mathbf{B}$	$(\mathbf{A} \Rightarrow \mathbf{B}) \Rightarrow (\neg\mathbf{A} \vee \mathbf{B})$
$\bot$	$\bot$	$\top$	$\top$	$\top$	$\top$
$\bot$	$\top$	$\top$	$\top$	$\top$	$\top$
$\top$	$\bot$	$\bot$	$\bot$	$\bot$	$\top$
$\top$	$\top$	$\bot$	$\top$	$\top$	$\top$

Table 2.3: Truth table for $(A \Rightarrow B) \Rightarrow (\neg A \vee B)$.

Truth tables are the least complicated way to determine if a formula is a theorem; the table systematically checks every possible assignment of truth values to the variables in the formula. In a truth table there is a separate column for each propositional variable and one for the formula being tested. Sometimes extra columns are added for each proper subformula of the formula being tested. A truth table has 2^n rows, where n is the number of propositional variables. Each row tests one of the possible truth value assignments to the propositional variables. The non-variable entries in a row are filled in, computing the truth value for the expression using the definitions of the connectives in the tables in Table 2.2. A formula is a theorem in classical logic if and only if the column under it contains only $\top$ values. An example of a truth table being used to prove the formula $(A \Rightarrow B) \Rightarrow (\neg A \vee B)$ is presented in Table 2.3.

2.4 Proofs and Types of Proofs

A *proof* is a demonstration that a formula is true for any possible assignment of values to its variables. Truth tables, discussed in Section 2.3, are a type of proof, but there are also others that are more efficient since they do not try every possibility.

A proof can be either a direct proof or a refutation. A direct proof constructs a *theorem* by using axioms and inference rules to derive the proposed theorem, while refuta-

tion proofs assume that the proposed theorem is false and then show that this assumption leads to a contradiction. If φ is the proposed theorem, then the refutation assumes $\neg\varphi$. If a contradiction is found, then $(\neg\varphi) \Rightarrow \bot$ has been proven. Since $(\neg\varphi) \Rightarrow \bot \equiv \neg\neg\varphi$ and $\neg\neg\varphi \equiv \varphi$, φ is established as a theorem.

2.5 A Valuation Function for Logical Formulas

If v is a function that maps logical formulas to truth values, then it can be defined deductively using the rules shown below:

- $v(\neg\varphi) = \top$ if and only if $v(\varphi) = \bot$
- $v(\neg\varphi) = \bot$ if and only if $v(\varphi) = \top$
- $v(\varphi \wedge \psi) = \top$ if and only if $v(\varphi) = \top$ and $v(\psi) = \top$
- $v(\varphi \wedge \psi) = \bot$ if and only if $v(\varphi) = \bot$ or $v(\psi) = \bot$
- $v(\varphi \vee \psi) = \top$ if and only if $v(\varphi) = \top$ or $v(\psi) = \top$
- $v(\varphi \vee \psi) = \bot$ if and only if $v(\varphi) = \bot$ and $v(\psi) = \bot$
- $v(\varphi \Rightarrow \psi) = \top$ if and only if $v(\varphi) = \bot$ or $v(\psi) = \top$
- $v(\varphi \Rightarrow \psi) = \bot$ if and only if $v(\varphi) = \top$ and $v(\psi) = \bot$

Using this deductive definition of v, the truth values of the subformulas can be determined. In section 2.6, Smullyan's rules present a systematic way to implement these rules.

Signed Formula E	**Rule Type $\mathbf{SM_r(E)}$**	**Child Subformulas $\mathbf{SM_c(E)}$**
$\langle T,\ \varphi \wedge \psi \rangle$	α	$\{\langle T,\ \varphi \rangle, \langle T,\ \psi \rangle\}$
$\langle F,\ \varphi \wedge \psi \rangle$	β	$\{\langle F,\ \varphi \rangle, \langle F,\ \psi \rangle\}$
$\langle T,\ \varphi \vee \psi \rangle$	β	$\{\langle T,\ \varphi \rangle, \langle T,\ \psi \rangle\}$
$\langle F,\ \varphi \vee \psi \rangle$	α	$\{\langle F,\ \varphi \rangle, \langle F,\ \psi \rangle\}$
$\langle T,\ \varphi \Rightarrow \psi \rangle$	β	$\{\langle F,\ \varphi \rangle, \langle T,\ \psi \rangle\}$
$\langle F,\ \varphi \Rightarrow \psi \rangle$	α	$\{\langle T,\ \varphi \rangle, \langle F,\ \psi \rangle\}$
$\langle T,\ \neg\varphi \rangle$	α	$\{\langle F,\ \varphi \rangle\}$
$\langle F,\ \neg\varphi \rangle$	α	$\{\langle T,\ \varphi \rangle\}$

Table 2.4: Smullyan's expansion rules. A system of rules for analyzing logical [Smullyan 1995] formulas used in many different proof methods. Each row describes one of Smullyan's rules. The expansion rule for a formula E is located by finding its sign and primary connective in the first column of a row. The second column (Rule Type $SM_r(E)$) of this row indicates the rule's type and the third column (Child Subformulas $SM_c(E)$) contains the child subformulas generated by the application of the rule.

2.6 Smullyan's Rules

Each of the proof methods discussed in this dissertation has rules that allow new formulas to be generated from existing formulas. Either explicitly or implicitly, all five methods employ Smullyan's rules as defined in Table 2.4. Smullyan's rules [Smullyan 1995] are used to analyze a logical formula. They implement the semantics of the evaluation function v discussed in Section 2.5. In a signed formula, the sign and the formula's primary connective determine which Smullyan rule to apply. There are eight rules, one for each combination of the two signs and the four connectives.

The term *expand* and its noun form *expansion* are used in the descriptions of the proof methods to indicate the application of a Smullyan rule. The standard descriptions of natural deduction and the Kripke C-tableau use their own sets of rules. However, this dissertation proves that these rule systems manipulate formulas in the same way as Smullyan's rules.

For propositional logic, Smullyan identified two types of rules: α and β. This dissertation separates the α-rules into unary and binary types. During an expansion, unary α-rules add one new formula to the proof, while binary α and β-rules add two. The α-rules add new formulas to the current proof, while β-rules split the proof into two subproofs, adding a new formula to each. An *α-formula* is a formula that requires an α-rule to expand it, and a *β-formula* is a formula that requires a β-rule to expand it.

To facilitate the discussion of Smullyan's rules, two functions, SM_r and SM_c, are introduced. The function SM_r maps a formula to either α or β, the two categories of Smullyan's expansion rules. The function SM_c maps a formula to a set of formulas added by an expansion and $|SM_c(E)|$ is the number of formulas added by the expansion of the formula E. Other texts use α and β as their rule types, this dissertation splits the α rules into two types, unary α when the expansion adds one formula and binary α when the expansion adds two formulas.

The actions taken for each rule type are described below and an example of its application is presented:

- If $SM_r(E) = \alpha$ and $|SM_c(E)| = 1$, then E is a unary α-formula. When an unary α-rule is applied, the formula in $SM_c(E)$ is added to the current proof. For example, $\langle T, \neg\varphi\rangle$ indicates that $\neg\varphi$ is a true formula. From this formula, it can be inferred that φ is false. Thus, $\langle F, \varphi\rangle$ is added to the proof.

- If $SM_r(E) = \alpha$ and $|SM_c(E)| = 2$, then E is a binary α-formula. When a binary α-rule is applied, the two formulas in $SM_c(E)$ are added to the current proof. For example, $\langle T, \varphi \wedge \psi\rangle$ indicates that $\varphi \wedge \psi$ is a true formula. From this formula, it can be inferred that both φ and ψ are true. Thus, $\langle T, \varphi\rangle$ and $\langle T, \psi\rangle$ are added to the proof.

- If $SM_r(E) = \beta$, then E is a β-formula. When a β-rule is applied, the current proof splits into two subproofs; one formula from $SM_c(E)$ is added to each subproof. For

example, $\langle T, \varphi \vee \psi \rangle$ indicates that $\varphi \vee \psi$ is a true formula. From this formula, it can be inferred that one or both of φ and ψ are true. The current proof is replaced with two new subproofs. If $\mathcal{E}$ is the set of formulas before the expansion, then after this expansion one subproof contains the formulas $\mathcal{E} \cup \{\langle T, \varphi \rangle\}$, and the other subproof contains the formulas $\mathcal{E} \cup \{\langle T, \psi \rangle\}$.

The goal of a refutation theorem prover is to close all of the subproofs it creates. A subproof *closes* if it contains contradictory formulas. As Smullyan's rules add formulas to subproofs or split them, a subproof closes when a formula is added that contradicts a formula already in the subproof.

An application of a β-rule could also create a third subproof with both subexpressions from $SM_c(E)$. Since this third subproof would contain the formulas $\mathcal{E} \cup SM_c(E)$, it would close if either of the other two subproofs close. Thus, the subproof with both child formulas is omitted.

2.7 Disjuncts and Clauses

Disjuncts and clauses are used in resolution, a proof method described in Section 3.5. Both are sets of formulas that represent the disjunction of their members. A *disjunct* contains both atomic and non-atomic formulas, while a *clause* contains only atomic formulas. An empty clause or disjunction is defined as being false. In this dissertation, a disjunct is displayed as a list of formulas between square brackets; for example, $[A, B]$, $[\neg A, C, D]$, and $[\neg A]$ are three clauses using the variables A, B, C, and D. Equation 2.1 shows how to transform a disjunct or clause into an equivalent logical formula.

$$[E_1, E_2, \ldots, E_n] = \bigvee_{i=1}^{n} E_i = E_1 \vee E_2 \vee \ldots \vee E_n \tag{2.1}$$
where $E_1, E_2, \ldots, E_n$ are formulas

A set of disjuncts $D_1, D_2, \ldots, D_m$ can be transformed into a logical formula using Equation 2.3.

$$\{D_1, D_2, \ldots, D_m\} = \bigwedge_{i=1}^{m} \bigvee_{j=1}^{|D_i|} E_{ij} \tag{2.2}$$
where $E_{ij}$is the j^{th}formula in disjunct D_i

Applying this equation to a set of disjuncts transforms them into an equivalent logical formula as shown in Equations 2.3 and 2.4.

$$\{[A, B], [\neg A, C, D], [\neg A]\} \equiv (A \vee B) \wedge (\neg A \vee C \vee D) \wedge (\neg A) \tag{2.3}$$

$$\begin{aligned}\{[A \Rightarrow B, \neg C], [D], [\neg E \wedge F, (G \vee H) \wedge I]\} \equiv \\ ((A \Rightarrow B) \vee \neg C) \wedge D \wedge ((\neg E \wedge F) \vee ((G \vee H) \wedge I))\end{aligned} \tag{2.4}$$

In unsigned notation, a *literal* is a variable or its negation (e.g. A or $\neg A$). In signed notation, a *literal* consists of a sign and a propositional variable (e.g. $\langle T, A\rangle$ or $\langle F, A\rangle$).

The formula in Equation 2.3 is in *conjunctive normal form* (CNF), while the formula in Equation 2.4 is not in CNF. A formula is in this form if 1) each formula in a clause is a literal and 2) the entire formula is a conjunction of clauses; the formula representing the set of clauses is a conjunction of disjunctions. Clauses always represent formulas in CNF, while disjuncts can represent formulas that are not in CNF.

2.8 Some Basic Graph Theory

A graph can be used to represent a variety of abstract and concrete relationships. A *graph* $G = (V, E)$ is composed of a finite set of vertices V and a set of edges $E \subseteq V \times V$ that connect vertices. An *edge* consists of a pair of vertices. In an undirected graph, edges represent a symmetric relation; for example, an edge xy could mean that x has met y, and, hence, y has also met x. In a directed graph, edges represent relationships that are not necessarily symmetric; for example, if one author u cited a second author v but the second author has not cited the first, then uv is an edge, but vu is not an edge. An undirected graph represents edges using an unordered pair xy of vertices, while directed graphs represent edges using an ordered pair (x, y) of vertices.

A *path* from vertex v_1 to a vertex v_n is a sequence of distinct vertices $v_1, v_2, \ldots, v_n$ such that for $1 \leq i < n,\ v_i v_{i+1}$ is an edge in the graph. A *cycle* is a path where the starting and ending vertices are the same, i.e. $v_1 = v_n$.

An *acyclic graph* is a graph that does not contain any cycles. In an undirected graph there is no path leading from any vertex back to itself. In directed graphs there is no path leading from any vertex following the directions of the edges that leads back to itself.

A *connected* graph is one where there is at least one vertex that has a path to every vertex in the graph. Equivalently, in an undirected, connected graph there is a path between every pair of vertices.

A *rooted tree* is a directed, acyclic, connected graph where there is a directed path from a single *root* vertex to every other vertex in the tree. If there is an edge (u, v) in a rooted tree, then u is called the *parent* of v and v is the *child* of u. A vertex w is a *descendant* of u if there is a directed path from u to w. A vertex u is an *ancestor* of w if w is a descendant of u. Two distinct vertices u and w are *unrelated* if u is neither an ancestor nor descendant of w. A *leaf* is a vertex that has no children. The function L is used to describe the construction of trees for an analytic tableau in Section 3.3. In a rooted tree, the expression $L(v)$ is the set of leaves that are descendants of the vertex v.

A partial order is a set of elements S and an ordering relation $<$ on $\subseteq S \times S$. If $p, q \in S$ and $p \neq q$, then either $p < q$, $q < p$, or p and q are unrelated. When $p < q$, then p is called a *predecessor* of q and q is the *successor* of p. Partial orders are often displayed as lattices. An element $q \in S$ is called *minimal* if there does not exist any element $p \in S$ with $p < q$.

2.9 Intuitionistic Truth: Frames, Worlds, and Forcing

Intuitionistic logic adds the notion that the truth value of a formula may not be known at a given time. This logic adds a time-like component to the analysis of the formulas. Some truth values may currently be unknown but may become known. A *world* is the state of knowledge at a given time. There is a directed acyclic graph (DAG) of possible future worlds; one shifts to another world when new information becomes available. Some formulas are *forced* (true), while others are unforced in a given world (state of knowledge). If a formula is true (forced) in a world, it is true (forced) in all possible successor worlds. Once the truth value is established, it is neither changed nor contradicted by future information.

In analyzing formulas in intuitionistic logic, a frame is constructed. A *frame* is a three-tuple $(P, \geq, A(p))$ where P is a set of worlds, $\geq$ a transitive and reflexive relation on P, and $A(p)$ the set of forced formulas in world p. The relation $\geq$ on the set P defines a partially ordered set. In this partially ordered set, each element in P represents a possible world or state of knowledge.

If q is a successor world of p, denoted as $q \geq p$, then $A(p) \subseteq A(q)$. The *monotonicity* property states that if a formula is forced in a world, it must also be forced in all successor worlds. The operator $\Vdash$ is used to indicate that the world on the left forces the formula on the right. Thus, the statement $p \Vdash \varphi$ indicates that φ is forced in world p, while the statement $p \nVdash \psi$ indicates that ψ is not forced in world p.

Nerode and Shore [Nerode and Shore 1993] use the function $A(p)$ for atomic formulas and define the semantics of the logical connectives, using frames and worlds in terms of forcing as shown in Figure 2.1. Later, Nerode and Shore in their Lemma 2.12 prove that for any forced formula φ, if $p \Vdash \varphi$, then for all $q \geq p$, $q \Vdash \varphi$.

In the semantics presented in Figure 2.1, disjunction ($\vee$) and conjunction ($\wedge$) are similar to the same operators in classical logic, but the semantics of negation ($\neg$) and implication ($\Rightarrow$) are different. (These differences are explored in detail in Chapter 5.)

2.10 Derivability and Validity

A logic is defined by its rules of inference and axioms. A formula φ is *derivable*, ($\vdash \varphi$), if it can be obtained using axioms and rules of inference. A logic is *complete* if every true formula can be derived using its inference rules and axioms. Truth tables are used in classical logic to establish the validity or truthfulness of a formula, but this approach does not work in intuitionistic logic. Instead, intuitionistic logic uses frames and forcing to establish validity, as discussed in Section 2.9. Classical and intuitionistic propositional

1. For all atomic formulas $\varphi, p \Vdash \varphi$ if and only if φ is in $A(p)$.
2. If $p \Vdash \varphi \Rightarrow \psi$, then for all $q \geq p$, $q \Vdash \varphi$ implies $q \Vdash \psi$.
3. If $p \nVdash \varphi \Rightarrow \psi$, then there exists a $q \geq p$ where $q \Vdash \varphi$, but $q \nVdash \psi$.
4. If $p \Vdash \neg\varphi$, then for all $q \geq p$, $q \nVdash \varphi$ (q does not force φ).
5. If $p \nVdash \neg\varphi$, then there exists a $q \geq p$ such that $q \Vdash \varphi$.
6. $p \Vdash \varphi \wedge \psi$ if and only if $p \Vdash \varphi$ and $p \Vdash \psi$.
7. $p \Vdash \varphi \vee \psi$ if and only if $p \Vdash \varphi$ or $p \Vdash \psi$.

Figure 2.1: Semantics of logical connectives in intuitionistic logic.

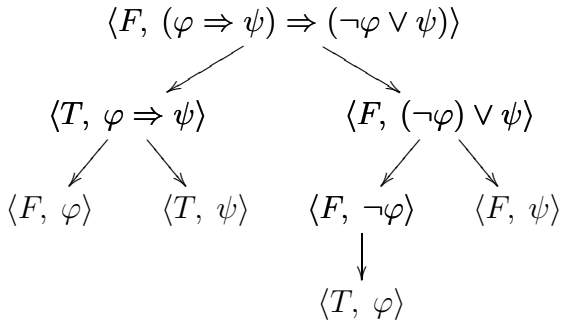


Figure 2.2: Formula tree for $(\varphi \Rightarrow \psi) \Rightarrow (\neg\varphi \vee \psi)$. This tree represents a partial order indicating the restrictions on the order in which signed formulas may be expanded during a proof. The subscripts on the operators have been added to aid in the discussion.

logic are both complete; every true (valid) formula can be derived from axioms using rules of inference.

2.11 Formula Trees

Formula trees are similar to parse trees, but the children of each vertex are labeled with the subformulas generated by using Smullyan's rules. The root of the formula tree for the proposed theorem φ is labeled with $\langle F, \varphi \rangle$. If a non-leaf vertex v is labeled with the

signed expression E, then its children are labeled with the formulas in $SM_c(E)$. The leaves are labeled with atomic formulas. The formula tree for $(\varphi \Rightarrow \psi) \Rightarrow (\neg\varphi \vee \psi)$ is shown in Figure 2.2. This tree is constructed solely by the function SM_c; the function SM_r does not affect the tree's construction.

Chapter 3

Proof Methods in Classical Logic

All logical formulas can be classified into one of three categories: theorems (tautologies), contingencies, and contradictions. Theorems evaluate to true for all possible assignments of values to their variables. Contingencies evaluate to true for some assignments and false for others. Contradictions evaluate to false regardless of the assignments made to their variables.

One way to demonstrate that the logical formula $A \Rightarrow A$ is a theorem is by assigning one of the two logical values, true ($\top$) or false ($\bot$), to A. After the assignment, the formula becomes either $\top \Rightarrow \top$ or $\bot \Rightarrow \bot$. Both assignments result in formulas that evaluate to true; hence, it is a theorem. As the number of variables grows, the number of possible assignments increases exponentially. Thus, trying all possible assignments becomes tedious and impractical. There are 2^n assignments for a logical formula with n variables. If a formula has ten or twenty variables, the number of possible assignments is $2^{10} = 1024$ or $2^{20} = 1,048,576$, respectively. Because of the large number of possible assignments, several algorithms have been developed that more efficiently decide if a logical formula is a theorem.

Since the early Twentieth Century, several methods have been developed to prove theorems more efficiently; five of these methods are natural deduction, Kripke tableau, resolution, analytic tableau, and matrix. There are other methods and minor variations of these methods, but the discussion in this dissertation is limited to these five. Each of these uses a different data structure to record the truth values of the formulas they manipulate. There are also differences in the way they handle how a proof is split into subproofs.

The two principal strategies that theorem provers use are direct proof and refutation. A *proposed theorem* is a formula that a method is trying to prove. For a proposed theorem P, a direct proof attempts to find a proof by working from the axioms/assumptions towards P, using the rules of inference. A refutation proof, on the other hand, shows that $\neg P$ is a contradiction. Recall, a contradiction is a formula that evaluates to false for all assignments. Thus, $\neg P$ evaluates to false; hence $(\neg P) \Rightarrow \perp \equiv \neg\neg P \equiv P$. Thus, P must evaluate to true for all assignments, and, hence, is a theorem. The five Sections 3.1 through 3.5 each describe one of the proof methods: natural deduction, Kripke C-tableau, analytic tableau, matrix, and resolution.

Chapter 4 shows that every method, except resolution, uses the same underlying algorithm. Section 3.6 shows how the clauses of the resolution method can be transformed into an equivalent matrix. Using these findings this dissertation demonstrates how a theorem prover can switch between methods during a proof.

3.1 Sequents and Natural Deduction Method

Sequents, introduced by Gentzen [Gentzen 1935], consist of two sets of logical formulas separated by a sequent arrow $\longrightarrow$. A natural deduction proof is a tree of sequents, with axioms (true statements) at its leaves and a sequent with the proposed theorem at its root. Axiom sequents are sequents expressing formulas that are accepted as true. The se-

$$\neg\bot \Leftrightarrow \top \tag{3.1}$$
$$\neg\top \Leftrightarrow \bot \tag{3.2}$$
$$\varphi \Rightarrow \psi \Leftrightarrow \neg\varphi \vee \psi \tag{3.3}$$
$$\varphi \wedge \bot \Leftrightarrow \bot \tag{3.4}$$
$$\varphi \vee \top \Leftrightarrow \top \tag{3.5}$$
$$\neg\varphi \vee \varphi \Leftrightarrow \top \quad \text{Excluded Middle} \tag{3.6}$$
$$\neg(\varphi \wedge \psi) \Leftrightarrow \neg\varphi \vee \neg\psi \quad \text{De Morgan's Law 1} \tag{3.7}$$
$$\neg(\varphi \vee \psi) \Leftrightarrow \neg\varphi \wedge \neg\psi \quad \text{De Morgan's Law 2} \tag{3.8}$$

Figure 3.1: List of common identities in classical logic.

quent rewrite rules describe which sequents can be placed above or below a given sequent. The rewrite rules are used to construct the tree of sequents that link the leaf sequents with the root sequent. This section begins by defining a sequent, moves on to describing sequent rules, and concludes with a step-by-step description of a proof's construction. The proof's construction illustrates how the sequent rules are combined.

Sequents, introduced by Gentzen [Gentzen 1935], consist of two sets of logical formulas separated by a sequent arrow $\longrightarrow$. The set on the left of the arrow is the antecedent, and the succedent is the set on the right. A sequent represents the statement: if all of the formulas in the antecedent are true, then at least one formula in the succedent must be true. Gentzen specified the semantics in terms of logical operators using equivalence in Formula 3.9, transforming a sequent into a logical formula.

$$\varphi_1, \varphi_2, \ldots, \varphi_n \longrightarrow \psi_1, \psi_2, \ldots, \psi_m \quad \equiv \quad \varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_n \Rightarrow \psi_1 \vee \psi_2 \vee \ldots \vee \psi_m \tag{3.9}$$

For example, the sequent

$$A \wedge B, B \vee \neg C \longrightarrow A \Rightarrow B, C$$

contains the formulas $A \wedge B$ and $B \vee \neg C$ in the antecedent and the formulas $A \Rightarrow B$ and C in the succedent. Applying Equivalence 3.9, this sequent can be transformed into the logical formula below:

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

In a proof, each sequent is linked to the sequent(s) above or below it by one of the sequent rewrite rules seen in Figure 3.2. The number of sequents in a proof depends on the order in which rewrite rules are applied and on the size and content of the proposed theorem's formula tree.

Rewrite and axiom sequent rules describe how to build a proof tree. Rewrite rules link sequents in a chain, while axiom rules indicate where the chain ends. In a rewrite rule, a horizontal line separates the premise sequent(s) above it from the conclusion sequent below it. Axiom rules consist of just one sequent.

For example, the rewrite rule $r\neg$ indicates that the sequent ($A \longrightarrow$) may be placed directly above the sequent ($\longrightarrow \neg A$). Figure 3.3 shows this rule with an example of its application. All sequent rules are stated in the most general terms using schema variables. The schema variables are represented by Greek letters. When a rule is applied, substitutions are made for the these letters; lower case letters represent a formula, and upper case letters represent a set of formulas. In Figure 3.3, when the rule is applied, φ is substituted for A, and the empty set is substituted for Γ and Δ.

$$\Gamma, \varphi \longrightarrow \varphi, \Delta \quad (Axiom, Ax1)$$

$$\Gamma, \bot \longrightarrow \Delta \quad (Axiom, Ax2) \qquad \Gamma \longrightarrow \top, \Delta \quad (Axiom, Ax3)$$

Rewrite Rules

$$\frac{\Gamma, \varphi, \psi \longrightarrow \Delta}{\Gamma, \varphi \wedge \psi \longrightarrow \Delta} l\wedge \qquad \frac{\Gamma \longrightarrow \varphi, \Delta \quad \Gamma \longrightarrow \psi, \Delta}{\Gamma \longrightarrow \varphi \wedge \psi, \Delta} r\wedge$$

$$\frac{\Gamma, \varphi \longrightarrow \Delta \quad \Gamma, \psi \longrightarrow \Delta}{\Gamma, \varphi \vee \psi \longrightarrow \Delta} l\vee \qquad \frac{\Gamma \longrightarrow \varphi, \psi, \Delta}{\Gamma \longrightarrow \varphi \vee \psi, \Delta} r\vee$$

$$\frac{\Gamma \longrightarrow \varphi, \Delta \quad \Gamma, \psi \longrightarrow \Delta}{\Gamma, \varphi \Rightarrow \psi \longrightarrow \Delta} l\Rightarrow \qquad \frac{\Gamma, \varphi \longrightarrow \psi, \Delta}{\Gamma \longrightarrow \varphi \Rightarrow \psi, \Delta} r\Rightarrow$$

$$\frac{\Gamma \longrightarrow \varphi, \Delta}{\Gamma, \neg\varphi \longrightarrow \Delta} l\neg \qquad \frac{\Gamma, \varphi \longrightarrow \Delta}{\Gamma \longrightarrow \neg\varphi, \Delta} r\neg$$

Figure 3.2: Classical logic sequent rules. This figure contains the axioms and rewrite rules for classical logic expressed using sequents. Γ and Δ represents sets of formulas that are not involved in the current rule. This set of sequent rules is a modification of the rules in [Wallen 1990].

The names of rewrite rules come from the formula E that appears in the conclusion but not in the premise(s). The name is a combination of the side of the sequent arrow where E appears, l indicating the antecedent and r the succedent, and E's primary connective (the operator at the root of E's parse tree). Since E is non-atomic, it must be one of four forms, either $\neg\rho$, $\varphi \wedge \psi$, $\varphi \vee \psi$, or $\varphi \Rightarrow \psi$. Thus, E's primary connective is one of the following: $\neg$, $\wedge$, $\vee$, or $\Rightarrow$. The eight rewrite rules result from the combination of the four primary connectives and the two sides of the sequent arrow. Thus, there is exactly one rule for each non-atomic formula wherever it appears in a sequent.

For example, the $l\vee$ sequent rule divides the formula $\varphi \vee \psi$ in the antecedent into two subformulas. The conclusion sequent C of the rule is ($\Gamma, \varphi \vee \psi \longrightarrow \Delta$). The rule's

Sequent Rule	**Example of This Rule's Application**
$\dfrac{\Gamma, \varphi \longrightarrow \Delta}{\Gamma \longrightarrow \neg\varphi, \Delta}\, r\neg$	$\dfrac{A \longrightarrow}{\longrightarrow \neg A}\, r\neg$

Figure 3.3: On the left the $r\neg$ rule is shown and on the right an example showing its application. In the rule's application A is substituted for φ, and the empty set is substituted for both Γ and Δ.

$$\frac{\Gamma, \varphi \longrightarrow \Delta \quad \Gamma, \psi \longrightarrow \Delta}{\Gamma, \varphi \vee \psi \longrightarrow \Delta}\, l\vee$$

$$\frac{A \wedge B, D \longrightarrow B \wedge \neg D, E \quad C \Rightarrow B, D \longrightarrow B \wedge \neg D, E}{(A \wedge B) \vee (C \Rightarrow B), D \longrightarrow B \wedge \neg D, E}\, l\vee$$

Figure 3.4: The $l\vee$ sequent rule and an example of its application. The top of the figure shows the $l\vee$ sequent rule and the bottom of the figure has an example of this rule's application. In the application of the rule, $A \wedge B$ is substituted for φ, $C \Rightarrow B$ for ψ, $\{D\}$ for Γ, and $\{B \wedge \neg D, E\}$ for Δ.

first premise sequent P_1 ($\Gamma, \varphi \longrightarrow \Delta$) is obtained from C by removing $\varphi \vee \psi$ and adding φ to the antecedent. The rule's second premise sequent P_2 ($\Gamma, \psi \longrightarrow \Delta$) is obtained from C by removing $\varphi \vee \psi$ and adding ψ to the antecedent. Figure 3.4 displays this sequent rule and an example of its application. In the application of this rule, the following substitutions are made: φ for $A \wedge B$, ψ for $C \Rightarrow B$, Δ for the set $\{B \wedge \neg D, E\}$, and Γ for the set $\{D\}$. Unlike the $r\neg$ rule, this rule has two premise sequents. Two premise rules split a proof tree branch into two branches, each representing a subproof. Each of these subproofs must be successful for the main proof to be successful. Generally, a proof of a non-trivial theorem has many subtrees.

In contrast to rewrite rules that govern the construction of a chain of sequents, axiom rules indicate the sequents which terminate the chain. Axiom sequents represent a formula that is accepted as true and, thus, is a terminal leaf. In classical propositional logic, axiom

sequents have one of three forms: (1) the antecedent contains $\bot$, (2) the succedent contains $\top$, or (3) the antecedent and the succedent have a formula in common. The top three rules in Figure 3.2 show these axioms represented as sequent schema. If an axiom sequent is substituted into Equivalence 3.9, the resulting logical formula can easily be proven to be true. Each axiom rule type is proven in Figures 3.5, 3.6, and 3.7.

Statement Number	**Statement**	**Justification**
1)	$A, \bot \longrightarrow B$	Given
2)	$A \wedge \bot \Rightarrow B$	By Identity 3.9
3)	$\bot \Rightarrow B$	By Identity 3.4
4)	$\neg\bot \vee B$	By Identity 3.3
5)	$\top \vee B = \top$	By Identities 3.1 and 3.5

Figure 3.5: Proof of the sequent, axiom rule $\varphi, \bot \longrightarrow \psi$. In the proof A is substituted for φ and B for ψ.

Statement Number	**Statement**	**Justification**
1)	$A \longrightarrow \top, B$	Given
2)	$A \Rightarrow \top \vee B$	Using Identity 3.9
3)	$A \Rightarrow \top$	Using Identity 3.5
4)	$\neg A \vee \top = \top$	Using Identities 3.3 and 3.5

Figure 3.6: Proof of the sequent, axiom rule $\varphi \longrightarrow \top, \psi$. In the proof A is substituted for φ and B for ψ.

A rewrite rule can be used as inference or reduction. An inference adds the conclusion sequent beneath the sequent(s) that match the premise(s) of a rule, while a reduction adds the premise sequent(s) above a leaf sequent that matches the conclusion sequent of a rule. An inference constructs the formula in the conclusion sequent from simpler formula(s) in the premise sequent(s). On the other hand, a reduction decomposes one formula E in the conclusion into its subformula(s) that replace E in the rule's premise(s). A completed proof can be viewed as either a reduction proof working from the proposed theorem

Statement Number	Statement	Justification
1)	$A, B \longrightarrow B, C$	Given
2)	$(A \wedge B) \Rightarrow (B \vee C)$	Using Identity 3.9
3)	$\neg(A \wedge B) \vee (B \vee C)$	Using Identity 3.3
4)	$(\neg A \vee \neg B) \vee (B \vee C)$	Using De Morgan's law 3.7
5)	$(\neg A \vee C) \vee (\neg B \vee B)$	Associativity and Community of $\vee$
6)	$(\neg A \vee C) \vee \top = \top$	Excluded middle 3.6 and Identity 3.5

Figure 3.7: Proof of the sequent, axiom rule $\varphi, \psi \longrightarrow \psi, \rho$. In the proof A is substituted for φ, B for ψ, and C for ρ.

to the axioms or an induction proof working from the axioms to the proposed theorem. By examining the completed proof tree, it is impossible to determine if it was created using inferences, reductions, or a combination of the two.

$$\dfrac{\dfrac{A \Rightarrow B \longrightarrow \neg A, B}{A \Rightarrow B \longrightarrow \neg A \vee B}\, r\vee}{\longrightarrow (A \Rightarrow B) \Rightarrow (\neg A \vee B)}\, r \Rightarrow$$

Figure 3.8: Incomplete natural deduction proof. This figure contains the first two steps of a natural deduction proof of $(A \Rightarrow B) \Rightarrow (\neg A \vee B)$.

To illustrate how these sequent rules are used to construct a proof tree for a formula, each step of the construction of the proof for $(A \Rightarrow B) \Rightarrow (\neg A \vee B)$ is discussed. Every proof is rooted with an *endsequent*, which has an empty antecedent and only the proposed theorem in the succedent. Figure 3.8 shows the first two steps in the proof. Assuming that this proof is constructed using reductions, the first step is to construct the endsequent for the proposed theorem at its root. The first sequent rule applied is $r \Rightarrow$, adding the sequent $A \Rightarrow B \longrightarrow \neg A \vee B$ followed by $r\vee$ adding the sequent $A \Rightarrow B \longrightarrow \neg A, B$.

After the first rule $r \Rightarrow$ is applied, either the rule $l \Rightarrow$ can be applied to $A \Rightarrow B$ or the rule $r\vee$ can be applied to $\neg A \vee B$. Two proofs for $A \Rightarrow B \longrightarrow \neg A \vee B$ are shown in

$$\dfrac{\dfrac{\dfrac{\dfrac{A \longrightarrow A, B}{\longrightarrow A, \neg A, B}\, r\neg \qquad B \longrightarrow \neg A, B}{A \Rightarrow B \longrightarrow \neg A, B}\, l\Rightarrow}{A \Rightarrow B \longrightarrow \neg A \vee B}\, r\vee}{\longrightarrow (A \Rightarrow B) \Rightarrow (\neg A \vee B)}\, r\Rightarrow \qquad \left| \qquad \dfrac{\dfrac{\dfrac{\dfrac{A \longrightarrow A, B}{\longrightarrow A, \neg A, B}\, r\neg}{\longrightarrow A, \neg A \vee B}\, r\vee \qquad \dfrac{B \longrightarrow \neg A, B}{B \longrightarrow \neg A \vee B}\, r\vee}{A \Rightarrow B \longrightarrow \neg A \vee B}\, l\Rightarrow}{\longrightarrow (A \Rightarrow B) \Rightarrow (\neg A \vee B)}\, r\Rightarrow \right.$$

Figure 3.9: Two natural deduction proofs. This figure contains two natural deduction proofs of $(A \Rightarrow B) \Rightarrow (\neg A \vee B)$. The proof on the left applies the $r\vee$ rule before the $l \Rightarrow$ rule, while the proof on the right applies the rules in the opposite order. Because of this difference, the proof on the left needs to apply the $r\vee$ rule only once, while the proof on the right needs to apply this rule twice.

Figure 3.9. The proof on the left side of the figure applies the $r\vee$ rule next, while the proof on the right applies the $l \Rightarrow$ rule next. In the proof on the right, since the two premise rule $l \Rightarrow$ was applied before the $r\vee$ rule, the $r\vee$ rule has to applied twice, once in each subtree. However, in the proof on the left, since the $r\vee$ rule was applied before the tree split, it needs to be applied only once.

Concentrating on the proof on the left side, the application of the $r\vee$ generates only one premise, $A \Rightarrow B \longrightarrow \neg A, B$. Next, the $l \Rightarrow$ is applied to $A \Rightarrow B$, splitting the tree and adding the sequents $\longrightarrow A, \neg A, B$ and $B \longrightarrow \neg A, B$. The second of these is an axiom; thus, the right subproof has concluded successfully. On the left branch, the $r\neg$ rule is applied, adding the sequent $A \longrightarrow A, B$. This new sequent is also an axiom; thus, the left subproof has concluded successfully. Since both subproofs (subtrees) concluded successfully, the entire proof is successful. The proof on the right side of the figure follows a similar set of rule applications to construct a different proof.

During almost every step of a proof, often there is more than one rule that can be applied, and, thus, a choice must be made. A good ordering of these choices creates a shorter proof, while a poor ordering generates a longer proof. In Figure 3.9, there was a

point where two rules could have been applied; the choice made on the right led to a longer proof. In general, it is wise to defer an application of a two premise rule when a one premise rule can be applied. Since two premise rules copy all formulas to each subtree (premise sequent), the number of copies of a given formula may grow quickly. If a one premise rule is applied to the formula ψ, replacing it with ψ_1 and ψ_2 followed by several two premise rules, then each leaf sequent will contain ψ_1 and ψ_2. However, if several two premise rules are applied first, then the rule replacing ψ with ψ_1 and ψ_2 will need to be applied on each of the branches to have the same effect. There is only a difference of one proof step between the proofs in Figure 3.9. However, for more complex formulas, the order in which formulas are expanded will lead to proofs with a very large difference in the number of steps. Even though the number and order of steps in the proofs for a given formula are different, the same set of axiom sequents are present at the leaves.

3.2 Kripke C-Tableau Method

The Kripke tableau algorithm [Kripke 1965] attempts to prove theorems in intuitionistic logic by constructing a tree of boxes. Here a simplified version, referred to as Kripke C-tableau, for classical logic is presented. The full version is presented in subsection 5.6.2.

In a Kripke C-tableau, a proof is constructed using a tree of boxes. Each box is divided into two columns, with antecedent formulas placed in the left column and succedent formulas in the right column. These columns were designed to mirror the antecedent and succedent of natural deduction [Beth 1956], and the sequent rules were converted into rules for manipulating formulas within boxes. In this method, the rewrite rules are used only as reductions. Single premise rules add their new formulas to the current box, while two premise rules split the box. Each time a box splits, the tree grows. A proof begins with a

root box containing the proposed theorem in the right column and then tries to construct boxes corresponding to axiom sequents.

This section begins by defining the column-formula pair notation for a Kripke box. Next, it explains how the rewrite and axiom rules are applied in a Kripke C-tableau with two examples of the rules. The section concludes with a step-by-step construction of a proof.

The contents of a box are represented as a set of Kripke column-formula (KCF) pairs, each consisting of a column $left$ or $right$ and a formula. For example, the ordered pair $\langle left, \neg A \rangle$ indicates that the logical formula $\neg A$ appears in the left column. The KCF set and the Kripke box in Figure 3.10 both represent the same set of formulas.

	$\varphi \Rightarrow \psi$
φ	ψ

The set of KCF $\{\langle right, \varphi \Rightarrow \psi \rangle, \langle left, \varphi \rangle, \langle right, \psi \rangle\}$ is another representation of this Kripke box.

Figure 3.10: An example of a Kripke box and its Kripke column-formula (KCF) notation. The contents of a Kripke tableau are represented as a set of Kripke column-formula pairs.

The rules for manipulating the formulas in a Kripke C-tableau have only a verbal description in [Kripke 1965]; it contains no example proofs. Kripke's rules are derived from the rules of natural deduction. Just as in natural deduction, the rewrite rule used to expand the formula E depends on the column and the primary connective of E. To facilitate the discussion, the functions KR_r and KR_c are introduced in Table 3.11; they indicate the actions to be taken to expand a formula. The function $KR_r(E)$ indicates if the current box should be split, and $KR_c(E)$ contains the formulas added in the expansion. The functions KR_r and KR_c describe the rewrite rules in [Kripke 1965] and were designed to mirror the functions SM_r and SM_c.

Parent Formula $(\mathbf{E})$ **Column and Form**	**Rule Type** $\mathbf{KR_r(E)}$	**Children** $\mathbf{KR_c(E)}$
$\langle left, \varphi \wedge \psi \rangle$	$no\ split$	$\{\langle left, \varphi \rangle,\ \langle left, \psi \rangle\}$
$\langle right, \varphi \wedge \psi \rangle$	$split$	$\{\langle right, \varphi \rangle,\ \langle right, \psi \rangle\}$
$\langle left, \varphi \vee \psi \rangle$	$split$	$\{\langle left, \varphi \rangle,\ \langle left, \psi \rangle\}$
$\langle right, \varphi \vee \psi \rangle$	$no\ split$	$\{\langle right, \varphi \rangle,\ \langle right, \psi \rangle\}$
$\langle left, \varphi \Rightarrow \psi \rangle$	$split$	$\{\langle right, \varphi \rangle,\ \langle left, \psi \rangle\}$
$\langle right, \varphi \Rightarrow \psi \rangle$	$no\ split$	$\{\langle left, \varphi \rangle, \langle right, \psi \rangle\}$
$\langle left, \neg\varphi \rangle$	$no\ split$	$\{\langle right, \varphi \rangle\}$
$\langle right, \neg\varphi \rangle$	$no\ split$	$\{\langle left, \varphi \rangle\}$

Figure 3.11: Kripke C-tableau rule table. This table distills the classical portion of Kripke's description in [Kripke 1965] of the rules for manipulating formulas into two functions, KR_r and KR_c. Suppose that E is a column-formula pair. The function $KR_r(E)$ function returns $split$ or $no\ split$ to indicate whether the expansion rule splits the current box. The function $KR_c(E)$ returns a set containing column-formula pairs that are added by the expansion rule.

The rules for closing a box are also derived from the three axiom rules of natural deduction. Like sequents, there are three kinds of axioms: 1) $\bot$ appears in the left column (the antecedent), 2) $\top$ appears in the right column (the succedent), and 3) the same formula appears in both columns (both sides of the arrow). If all leaf boxes close, then the theorem has been proven.

The method begins by creating a root box, similar to the endsequent in natural deduction, by placing the proposed theorem in the right column. Each subsequent step expands a formula E. If $KR_r(E) = no\ split$, similar to applying a one premise sequent rewrite rule, then the Kripke C-tableau adds the formula(s) in $KR_c(E)$ to the current box. However, if $KR_r(E) = split$, similar to applying a two premise sequent rule, then the Kripke C-tableau box splits, creating two boxes under the current box. All the formulas in the parent are copied to each new box, and one formula from $KR_c(E)$ is added to each child box. Kripke C-tableaux are more efficient than natural deduction since all formulas in a box are copied only when that box splits.

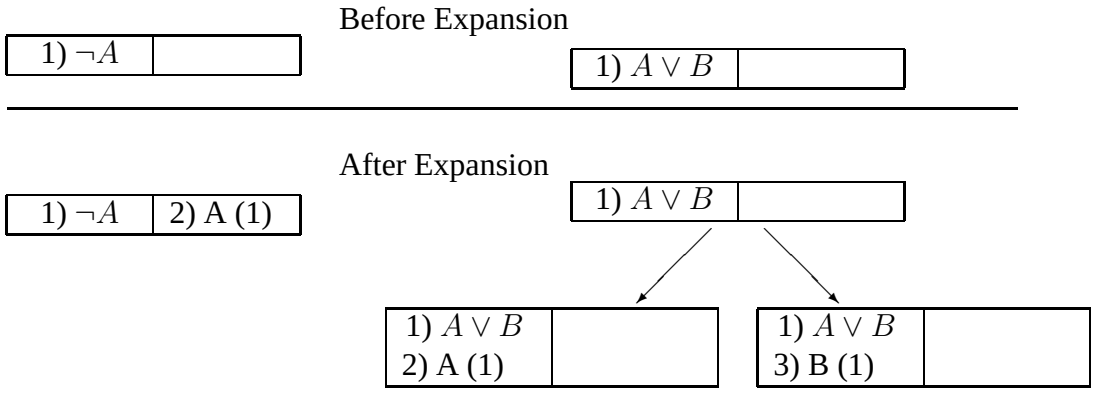


Figure 3.12: Examples of a split and an non-split rule in a Kripke C-tableau. This figure shows one example of each of the two Kripke rule types; the example on the left shows a rule that does not split the box, while the example on the right splits the box.

Examples of the application of the two Kripke rule types, $split$ and $no\ split$, are shown in Figure 3.12. On the left of the figure, the Formula E_a $\langle left, \neg A\rangle$ is expanded. Since $KR_r(E_a) = no\ split$ and $KR_c(E_a) = \{\langle right, A\rangle\}$, the formula A is added to the right column of the current box. On the right, the Formula E_b, $\langle left, A \vee B\rangle$, is expanded. Since $KR_r(E_b) = split$ and $KR_c(E_b) = \{\langle left, A\rangle, \langle left, B\rangle\}$, the current box is split into two new ones, one of which contains A in its left column and the other B in its left column.

Now a step-by-step description of the Kripke C-tableau proof for $(\varphi \Rightarrow \psi) \Rightarrow (\neg\varphi \vee \psi)$ is presented; the Kripke C-tableau is shown in Figure 3.13. This description shows how the Kripke rules are combined to construct a proof. For convenience, E_i will represent the KCF pair numbered i. The first step of the proof is to place the proposed theorem, numbered 1, in the right column of the root box. Thus, $E_1 = \langle right, (\varphi \Rightarrow \psi) \Rightarrow (\neg\varphi \vee \psi)\rangle$. Since E_1 is the only logical formula, it is chosen for expansion. Because $KR_r(E_1) = no\ split$, the subformula(s) generated by the expansion of E_1 are added to this box. These formulas are $KR_c(E_1) = \{\langle left, \varphi \Rightarrow \psi\rangle, \langle right, \neg\varphi \vee \psi\rangle\}$. These two formulas are added to the root box and numbered as 2 and 3, respectively.

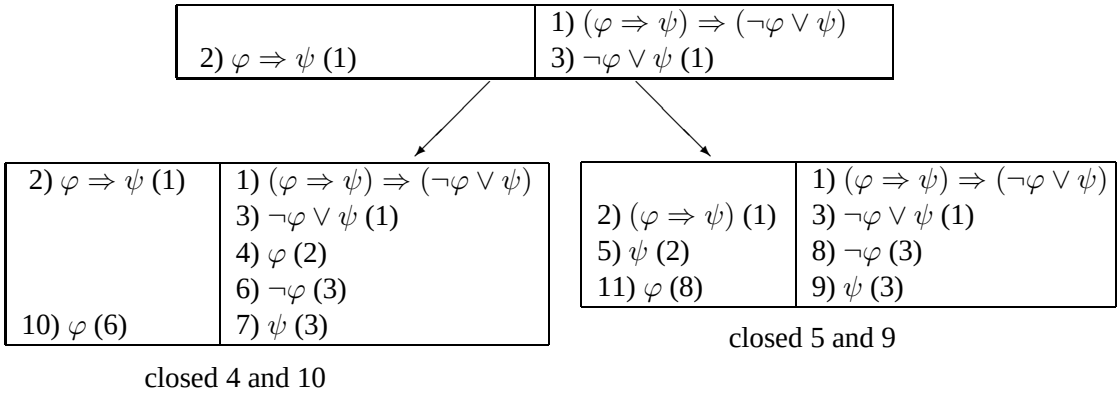


Figure 3.13: A Kripke C-tableau proof of $(\varphi \Rightarrow \psi) \Rightarrow (\neg\varphi \vee \psi)$

In the next expansion step, either Formula 2 or 3 can be expanded. In this version of the proof, Formula 2 is expanded next. Because $KR_r(E_2) = split$, the box is split into a left and a right box. The two new boxes are shown under the root box in Figure 3.13. Formulas 1, 2, and 3 are copied from the root box into the left and right boxes. Next the KCF formulas in $KR_c(E_2)$ are placed in these boxes, Formula 4 $\langle right, \varphi\rangle$ in the left box and Formula 5 $\langle left, \psi\rangle$ in the right box.

Since the root box split before Formula 3 was expanded, it needs to be expanded twice, once in each leaf box. On the other hand, if Formula 3 was expanded before Formula 2, this duplication would have been avoided. Returning to the proof, next Formula 3 in the left box is expanded. Since $KR_r(E_3) = no\ split$, the members of $KR_c(E_3) = \{\langle right, \neg\varphi\rangle, \langle right, \psi\rangle\}$ are added to left box as formulas 6 and 7. Next, the same formula is expanded in the right box, adding the same formulas, but numbered 8 and 9. The right box can be closed since the same formula appears in both columns, $\langle left, \psi\rangle$ (Formula 5) and $\langle right, \psi\rangle$ (Formula 9).

The only non-atomic formula in the left box is 6 $\langle right, \neg\varphi\rangle$, which is expanded next. Since $KR_r(E_6) = no\ split$, the formula in $KR_c(E_6) = \{\langle left, \varphi\rangle\}$ is added as

Formula 10. This box can now be closed since the same formula appears in both columns, $\langle right, \varphi \rangle$ (Formula 4) and $\langle left, \varphi \rangle$ (Formula 10). Even though the right box is already closed, Formula 8 (which is the same as Formula 6) may be expanded, adding Formula 11 to the right box. Since all the leaf boxes are closed, the proposed theorem has been proven.

3.3 Analytic Tableau Method

Analytic tableau, a refutation method, proves theorems by constructing a binary tree in which each vertex is labeled with a signed formula. In an attempt to prove that the formula φ is a theorem, the first step creates a root vertex labeled with $\langle F, \varphi \rangle$, assuming that the proposed theorem is false. Each subsequent step expands a vertex using one of Smullyan's rules and then determines if the modified branch(es) can be closed. A branch is closed if it contains an obvious contradiction. A theorem is proven when all of the branches close. Figure 3.14 shows an example of an application of each rule type: unary α, binary α, and β.

When a vertex v labeled with the formula E is expanded, its rule type determines how a new vertex or vertices are added as descendants of each leaf vertex in $L(v)$. Recall that the function $L(v)$ is defined to be the set of leaves beneath v; each case below describes how a rule type adds new vertices to the tree.

- If $SM_r(E) = \alpha$ and $|SM_c(E)| = 1$, then a unary α-rule is applied, adding a single vertex as the child of each vertex in $L(v)$. Each of these new vertices is labeled with the formula in $SM_c(E)$.

- If $SM_r(E) = \alpha$ and $|SM_c(E)| = 2$, then a binary α-rule is applied, adding two vertices as descendants of each vertex in $L(v)$. If $v_\ell \in L(v)$, one vertex is added as

Rule Type	An Application of the Rule Type
Unary α-rule	$T,\ \neg\varphi$ $\downarrow$ $F,\ \varphi$
Binary α-rule	$T,\ \varphi \wedge \psi$ $\downarrow$ $T,\ \varphi$ $\downarrow$ $T,\ \psi$
β-rule	$T,\ \varphi \vee \psi$ $\swarrow \quad \searrow$ $T,\ \varphi \qquad T,\ \psi$

Figure 3.14: Examples of the application of an α and a β rule type in analytic tableau. Since confusion is not likely, the signed formulas are presented without angle brackets.

the child of v_ℓ and the other as the child of the first, making it the grandchild of v_ℓ. The new descendants of v_ℓ are labeled with the formulas in $SM_c(E)$.

- If $SM_r(E) = \beta$, then a β-rule is applied, adding two vertices as siblings beneath each vertex in $L(v)$. These new vertices are labeled with the formulas in $SM_c(E)$.

After vertices are added, the tree is examined; any of the branches that have become obviously contradictory are closed. A branch is closed when it meets one of three conditions: 1) the branch contains a vertex labeled with $\langle T, \bot \rangle$, 2) the branch contains a vertex labeled with $\langle F, \top \rangle$, or 3) for a formula φ, the branch contains vertices labeled with $\langle T, \varphi \rangle$ and $\langle F, \varphi \rangle$. The first two conditions close the branch because they represent the inherently

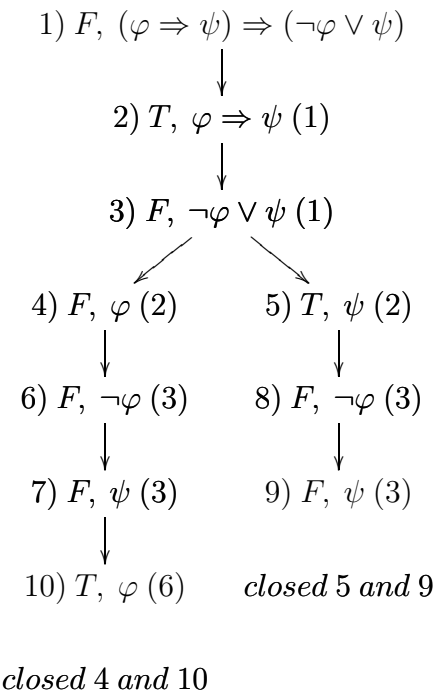


Figure 3.15: Analytic tableau proof of $(\varphi \Rightarrow \psi) \Rightarrow (\neg\varphi \vee \psi)$. Each vertex contains three items: 1) an number to identify it, 2) a formula labeling it, and 3) a number in parentheses. The number in parentheses is the number of the vertex whose expansion created this vertex.

contradictory statements that "false is a true formula" or "true is a false formula." The third condition closes the branch because it indicates that the same formula is both true and false. Once closed, a branch cannot be reopened.

Now a step-by-step description of an analytic tableau proof of $(\varphi \Rightarrow \psi) \Rightarrow (\neg\varphi \vee \psi)$ shown in Figure 3.15 is presented. In the discussion of the construction of this proof, v_i represents the vertex numbered with i, and E_i represents the signed formula labeling vertex v_i. The first step of the method creates Vertex v_1, labeled with the assumption that the proposed theorem is false. To expand E_1, first its rule type needs to be determined; since $SM_r(E_1) = \alpha$ and $|SM_c(E_1)| = 2$, a binary α-rule is applied, adding vertices v_2 and v_3 as the child and grandchild of Vertex v_1. These new vertices are labeled with the formulas in $SM_c(E_1)$; hence, $\langle T, \varphi \Rightarrow \psi\rangle$ labels Vertex v_2 and $\langle F, \neg\varphi \vee \psi\rangle$ labels Vertex v_3.

Next, Vertex v_2 is expanded; since $SM_r(E_2) = \beta$, the expansion uses a β-rule, adding vertices v_4 and v_5 as the children of Vertex v_3. These vertices are labeled with the signed formulas in the set $SM_c(E_2) = \{\langle F, \varphi\rangle, \langle T, \psi\rangle\}$. Next, Vertex v_3 is expanded using a binary α-rule. At this point, $L(v_3) = \{v_4, v_5\}$, meaning two new vertices are added beneath each leaf. Vertices v_6 and v_8 labeled with $\langle F, \neg\varphi\rangle$ are added as children of vertices v_4 and v_5, respectively, and vertices v_7 and v_9 labeled with $\langle F, \psi\rangle$ are added as the children of v_6 and v_8, respectively. As can be seen in Figure 3.15, Vertex v_7 is the grandchild of Vertex v_4, and Vertex v_9 is the grandchild of Vertex v_5. The right branch can now be closed since vertices v_5 and v_9 are labeled with contradictory formulas.

On the left branch, Vertex v_6, labeled with $\langle F, \neg\varphi\rangle$, can still be expanded. Since $SM_r(E_6) = \alpha$ and $SM_c(E_6) = \{\langle T, \varphi\rangle\}$, Vertex v_{10}, labeled with $\langle T, \varphi\rangle$, is added as the child of Vertex v_7. The left branch may now be closed since vertices v_4 and v_{10} are labeled with contradictory formulas. Since no open branches remain, the proof has succeeded.

As the example illustrates, the number of branches increases with each application of a β-rule. Thus, a simple heuristic is to apply α-rules before β-rules so that the number of open branches present in the middle of the proof is reduced. There are two good reasons for this heuristic. First, fewer open branches mean that fewer branches need to be closed in order to complete the proof. Second, each expansion step adds vertices beneath each leaf vertex; thus, fewer branches mean that fewer vertices need to be added during an expansion step. In the example presented here, if Vertex v_3 was expanded before Vertex v_2, only two vertices would have been added instead of four.

The matrix method, presented next, saves space by never having to add more than four entries or vertices during each expansion. A matrix uses paths through its structure where the analytic tableau uses branches; thus, a matrix proof is successful if all paths contain contradictory formulas. However, recognizing that a proof is successful in the analytic tableau is much easier than in the matrix method. In the analytic tableau method,

as soon as contradictory formula(s) are added to a branch, it is closed. If all of the branches are closed, the proof is successful. In the matrix method, a path is just a set of vertices in a directed acyclic graph (DAG). Many paths may share a given vertex or edge; hence, there is nothing unique to a given path that can be marked to indicate that it is contradictory. Thus, one must wait until all expansions have been performed before checking that all paths through the DAG are contradictory.

3.4 Matrix Method

The matrix method, described in [Andrews 1981] and [Bibel 1987], constructs a graph from the proposed theorem. This method consists of two phases; the first constructs a matrix using Smullyan's rules, and the second searches for a contradiction-free path through it. If no such path can be found, then the theorem has been proven. The phrase *path through a matrix* is used to indicate paths that are maximal within a matrix; non-maximal paths can lengthened by adding a vertex to one of their ends. Traditionally, this method has used nested matrices; this dissertation introduces two DAG representations.

3.4.1 Nesting Matrices

A nested matrix is defined recursively using one of three structures: 1) a signed formula, 2) a single column of matrices, or 3) a single row of matrices. A nested matrix visually illustrates the semantic structure of the formula from which it was created. In a matrix, a signed formula represents itself. A column of submatrices represents a disjunction, while a row of submatrices represents a conjunction.

The first step in the construction of the matrix proof for φ creates a matrix with $\langle F, \varphi \rangle$ as its only entry. Each subsequent step replaces a formula E with a matrix using Smullyan's rules. If a unary α-rule is applied, E is replaced with the signed formula in

Rule Type	**Matrix Before Expansion**	**Matrix Generated by Expansion**
unary $-\ \alpha$	$[\langle T, \neg\varphi\rangle]$	$[\langle F, \varphi\rangle]$
binary $-\ \alpha$	$[\langle T, \varphi \wedge \psi\rangle]$	$[\langle T, \varphi\rangle \quad \langle T, \psi\rangle]$
β	$[\langle T, \varphi \vee \psi\rangle]$	$\begin{bmatrix} \langle T, \varphi\rangle \\ \langle T, \psi\rangle \end{bmatrix}$

Figure 3.16: Nested matrix examples of the three rule types. There are three rule types: unary α, binary α, and β. One example of the application of each is illustrated above.

$SM_c(E)$. If a binary α-rule is applied, E is replaced with a single row containing the two formulas in $SM_c(E)$. If a β-rule is applied, E is replaced with a single column with two rows, each containing a formula in $SM_c(E)$. An example of an application of each of these rule types is shown in Figure 3.16. The construction phase is complete when all formulas are atomic, meaning that there are no formulas to which Smullyan's rules can be applied.

The row and column structure of a matrix determines the location of the paths through the matrix. A *path* is a set of formulas found as one moves from the left to the right of the matrix. Each part of the path depends on the type of entry. If the entry is a formula E, then the path through this matrix consists of only E. A column of submatrices represents a disjunction; it contains multiple paths, one through each submatrix. A path through a column consists of a path through one submatrix in the column. A row of submatrices, on the other hand, represents a conjunction. A path through a row consists of the concatenation of a path from each of its submatrices.

In the path-checking phase, a search is made for a path through the matrix that does not contain an obvious contradiction. Similar to analytic tableau, a path contains such a

$$\left[\ \langle F, (\varphi \Rightarrow \psi) \Rightarrow (\neg\varphi \vee \psi)\rangle\ \right] \quad (A)$$

$$\left[\ \langle T, \varphi \Rightarrow \psi\rangle \quad \langle F, \neg\varphi \vee \psi\rangle\ \right] \quad (B)$$

$$\left[\ \left[\ \begin{matrix}\langle F, \varphi\rangle \\ \langle T, \psi\rangle\end{matrix}\ \right] \quad \langle F, \neg\varphi \vee \psi\rangle\ \right] \quad (C)$$

$$\left[\ \left[\ \begin{matrix}\langle F, \varphi\rangle \\ \langle T, \psi\rangle\end{matrix}\ \right] \quad \langle F, \neg\varphi\rangle \quad \langle F, \psi\rangle\ \right] \quad (D)$$

$$\left[\ \left[\ \begin{matrix}\langle F, \varphi\rangle \\ \langle T, \psi\rangle\end{matrix}\ \right] \quad \langle T, \varphi\rangle \quad \langle F, \psi\rangle\ \right] \quad (E)$$

Figure 3.17: The matrix proof of the formula $(\varphi \Rightarrow \psi) \Rightarrow (\neg\varphi \vee \psi)$. The matrix derivation sequence is labeled in alphabetical order. The initial matrix is labeled with (A) and the final matrix with (E). The other matrices are derived from the previous one by the expansion of a single formula.

contradiction if there is a formula ψ where $\langle T, \psi\rangle$ and $\langle F, \psi\rangle$ both appear on the path or the path contains $\langle T, \bot\rangle$ or $\langle F, \top\rangle$.

Matrices A through E in Figure 3.17 show the matrix proof after each step for the Formula $E_1 = (\varphi \Rightarrow \psi) \Rightarrow (\neg\varphi \vee \psi)$. A is the initial matrix. Since $SM_r(E_1) = \alpha$ and $|SM_c(E_1)| = 2$, matrix B is obtained from A by applying a binary α-rule to replace E_1 with a row containing the two formulas in $SM_c(E_1)$. For convenience, E_2 and E_3 represent the formulas in $SM_c(E_1)$, where $E_2 = \langle T, \varphi \Rightarrow \psi\rangle$ and $E_3 = \langle F, \neg\varphi \vee \psi\rangle$. Since $SM_r(E_2) = \beta$, matrix C is obtained from B by replacing E_2 with a one-column matrix having two rows, each containing one formula from $SM_c(E_2)$. Matrix D is the result of expanding E_3 in matrix C; since $SM_r(E_3) = \alpha$ and $|SM_c(E_3)| = 2$, a binary α-rule replaces E_3 with a matrix having one row with two entries, each labeled with a formula in $SM_c(E_3)$. Finally, matrix E is obtained using an unary α-rule to replace $\langle F, \neg\varphi\rangle$ with $\langle T, \varphi\rangle$.

Matrix E has two paths, p_1 and p_2, passing through it, where p_1 contains $\langle F, \varphi\rangle$, p_2 contains $\langle T, \psi\rangle$, and both contain $\langle T, \varphi\rangle$ and $\langle F, \psi\rangle$. Since p_1 contains both $\langle F, \varphi\rangle$ and $\langle T, \varphi\rangle$, it is contradictory; p_2 is also contradictory since it contains both $\langle F, \psi\rangle$ and $\langle T, \psi\rangle$. Since all paths through the matrix are contradictory, the proposed theorem has been proven.

Since only one or two entries are added in each step, the final matrix is small, but the number of paths can be large. If E is a β-formula and E_1 and E_2 represent the formulas in $SM_c(E)$, then expansion of E replaces it with a one column matrix with two rows, one containing E_1 and the other containing E_2. For every path that passed through E, there are now two, one passing through E_1 and the other through E_2. This is similar to the analytic tableau, where the β-rules double the number of branches containing a formula. Unlike analytic tableau, the final matrix size does not depend on the order in which the formulas are expanded. However, since any vertex or edge in a matrix may be part of multiple paths, there is no simple way to eliminate contradictory paths as the matrix is constructed.

Most of the literature describing the matrix method focuses on matrix construction, leaving readers to develop their own procedures to show that all paths through the matrix are contradictory. For non-classical logics, the literature describes the extra requirements for that specific logic. Thus, for classical logic the construction of a matrix transforms the question of theoremhood to the graph theory question of showing that all paths through the matrix are contradictory.

The nested matrix representation works well for small examples, but larger ones cannot be feasibly presented on a page. When the nesting becomes very deep, the paths through the nested matrices become difficult to identify. To avoid this situation, this dissertation introduces two new DAG representations.

3.4.2 Construction of the DAG Matrix

In the construction of a DAG matrix, each expansion step uses a Smullyan rule to replace a vertex with a subgraph. The kind of subgraph that replaces a vertex depends upon which of the three rule types is applied. These subgraphs contain structural and formula vertices. Structural vertices are unlabeled and are used to connect the new subgraph with

the vertices adjacent to the vertex being replaced. Each of the formula vertices includes a formula number, the formula itself E, and in parentheses the number of the formula E' whose expansion generated E. The DAG matrix is a series-parallel graph, first introduced by Duffin in [Duffin 1965] to describe electrical circuits.

The steps below describe the construction procedure for a DAG proof from the proposed theorem φ. The initial DAG is a path with three vertices, two structural and one formula. The endpoints are structural vertices called $input$ and $output$. The middle vertex, a formula vertex, is labeled with the formula $\langle F, \varphi \rangle$. There is an edge from the $input$ to the middle vertex and another from the middle vertex to $output$. During each subsequent step, one of Smullyan's rules is used to replace a vertex with a subgraph. This process continues until all formula vertices are labeled with atomic formulas. If all paths between the $input$ and $output$ are contradictory, then the theorem has been proven.

In an expansion step, the subgraph replacing a vertex v is determined by the rule type used to expand the formula that labels v; α-rules replace v with a path and β-rules replace v with a diamond-shaped subgraph (hereafter called a diamond) as shown in Table 3.1. As this table shows, the diamond contains two structural vertices named in and out but not labeled; these vertices connect the subgraph to the vertices that were adjacent to the vertex v. To make each in and out vertex distinct, a subscript is attached to their names; thus, the k^{th} application of a β-rule creates vertices named in_k and out_k.

The choice of whether to insert a path or a diamond reflects the semantics of the logical connectives. Assuming the vertex v is chosen for expansion and it is labeled with E, Figure 3.1 shows an example of how each of the three rule types is applied.

- If E is an unary α-formula, then v is replaced by the vertex v' labeled with the formula in $SM_c(E)$. Any path that passed through v now passes through v'.

Rule Type	Before Expansion	After Expansion
Unary-α	$\mathcal{G}_1 \longrightarrow \langle T, \neg\varphi\rangle \longrightarrow \mathcal{G}_2$	$\mathcal{G}_1 \longrightarrow \langle F, \varphi\rangle \longrightarrow \mathcal{G}_2$
Binary-α	$\mathcal{G}_1 \longrightarrow \langle T, \varphi \wedge \psi\rangle \longrightarrow \mathcal{G}_2$	$\mathcal{G}_1 \longrightarrow \langle T, \varphi\rangle \rightarrow \langle T, \psi\rangle \longrightarrow \mathcal{G}_2$
β	$\mathcal{G}_1 \longrightarrow \langle T, \varphi \vee \psi\rangle \longrightarrow \mathcal{G}_2$	$\mathcal{G}_1 \longrightarrow in_k \nearrow \langle T, \psi\rangle \searrow$ $\langle T, \varphi\rangle \nearrow out_k \longrightarrow \mathcal{G}_2$

Table 3.1: Matrix expansion rules. This table shows examples of the subgraphs inserted into the DAG for the α and β-rule types. The symbols $\mathcal{G}_1$ and $\mathcal{G}_2$ represent the parts of the graph surrounding the vertex being expanded.

- If E is a binary α-formula, then v is replaced by a path consisting of two vertices v_1 and v_2. Any path that passed through v now passes through both v_1 and v_2.

- If E is a β-formula, then v is replaced with a diamond. This subgraph has two structural vertices and two formula vertices. The structural vertices provide connections with the surrounding graph, while the formula vertices, each labeled with one formula in $SM_c(E)$, are on separate paths through the subgraph. For any path that passed through v, there are now two paths, each containing one vertex labeled with a formula in $SM_c(E)$.

Similar to the nesting matrix, a contradictory path between the $input$ and $output$ vertices meets one of three following conditions: 1) it contains $\langle T, \bot\rangle$, 2) it contains $\langle F, \top\rangle$, or 3) for a formula φ, there is a vertex labeled with $\langle T, \varphi\rangle$ and another with $\langle F, \varphi\rangle$. The procedure for finding a non-contradictory path is the same as that for nested matrices.

3.4.3 An Alternate DAG Representation and Its Rejection

When considering possible DAG representations to replace the nested matrix, two possibilities were considered. The graphs at the top and bottom of Figure 3.18 illustrate these two alternatives. The neighborhood representation at the top adds edges between

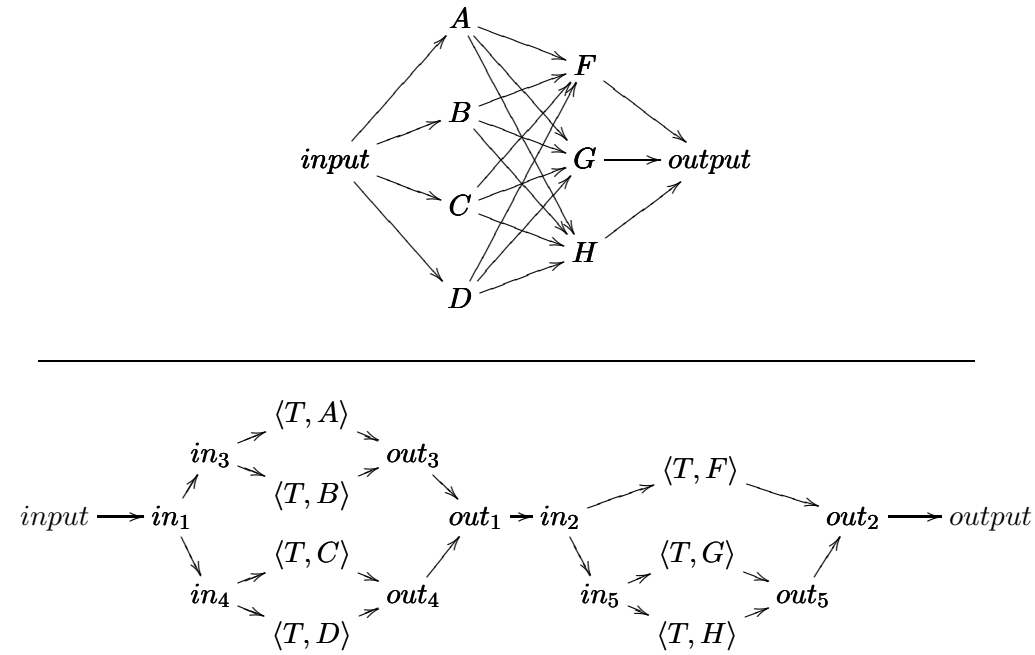


Figure 3.18: Comparison of two DAG representations of the formula $\langle T, ((A \vee B) \vee (C \vee D)) \wedge (F \vee (G \vee H))\rangle$. The DAG at the top uses the neighborhood representation, while the DAG at the bottom use the diamond representation. The DAGs use diamonds with structural vertices labeled in_k and out_k.

vertices labeled with formulas, while the diamond representation at the bottom adds two structural vertices called in and out to the graph each time a β-rule is applied. These structural vertices are inserted to reduce the number of edges in a graph, especially for complete bipartite subgraphs, as shown at the top of Figure 3.18.

The neighborhood graph illustrates an alternative representation in which no structural vertices are used. The *in-neighborhood* of a vertex v, denoted as $N^-(v)$, is $\{x \mid xv \in E(G)\}$ and the *out-neighborhood* of vertex v, denoted as $N^+(v)$, is $\{x \mid vx \in E(G)\}$ [West 1996][page 46]. In the neighborhood representation, the size of a vertex v's in-neighborhood and out-neighborhood can become very large and difficult to manage. For instance, if vertex v is labeled with $\langle T, \varphi \vee \psi\rangle$ and there are k vertices in $N^-(v)$ and ℓ vertices in $N^+(v)$, then expanding v would replace it with v_1 and v_2; the edges adjacent to

v could be removed, but $2(k+\ell)$ edges need to be added. For each vertex v^- in $N^-(v)$, two edges v^-v_1 and v^-v_2 need to be added, and for each vertex v^+ in $N^+(v)$, two edges v_1v^+ and v_2v^+ need to be added. The description of this β-rule is not simple, and its execution can be tedious.

The diamond representation illustrates how the diamond subgraph makes it easier to construct the DAG, to identify paths through the DAG, and to separate a DAG into subgraphs so that large DAGs may be easily presented in separate figures. The neighborhood representation assumes that the formula is presented in CNF, while the diamond representation can handle any formula.

The neighborhood graph contains many edges; thus, it is difficult to identify all of its paths. For this small example in Figure 3.18, it is feasible to identify all of the paths through the graph, but for larger formulas the number and size of clauses produce more and larger complete bipartite subgraphs, making the identification of the paths much more difficult, while in the diamond representation, the use of diamonds reduces the number edges required to represent a given formula. Thus, the diamond representation was chosen over the neighborhood representation.

3.4.4 Appending Matrix DAG

The last modification to the matrix representation is called the *appending DAG representation*. This representation differs from the previous one; during the construction phase, instead of replacing the vertex v with a subgraph, a subgraph is inserted immediately after v. A rule is added that a vertex may be expanded only once. Since formulas are preserved in the appending matrix, the appending DAG matrix is useful in proving that paths in the matrix method are containers (as discussed in Chapter 4) and presenting a transformation between the matrix and resolution methods in Section 3.6.

Appending Matrix

$input \longrightarrow T,\ (\upsilon \wedge \varphi) \wedge (\chi \vee \psi) \rightarrow output$

$input \longrightarrow \underline{T,\ (\upsilon \wedge \varphi) \wedge (\chi \vee \psi)} \rightarrow T,\ \upsilon \wedge \varphi \rightarrow T,\ \chi \vee \psi \rightarrow output$

$input \rightarrow \underline{T,\ (\upsilon \wedge \varphi) \wedge (\chi \vee \psi)} \longrightarrow \underline{T,\ \upsilon \wedge \varphi} \longrightarrow T,\ \upsilon \longrightarrow T,\ \varphi \rightarrow T,\ \chi \vee \psi \longrightarrow output$

$input \rightarrow \underline{T,\ (\upsilon \wedge \varphi) \wedge (\chi \vee \psi)} \longrightarrow \underline{T,\ \upsilon \wedge \varphi} \longrightarrow$

$T,\ \upsilon \longrightarrow T,\ \varphi \rightarrow \underline{T,\ \chi \vee \psi} \rightarrow \boldsymbol{in}_1 \begin{matrix} \nearrow T,\ \psi \searrow \\ \searrow T,\ \chi \nearrow \end{matrix} \boldsymbol{out}_1 \longrightarrow output$

Replacement Matrix

$input \longrightarrow T,\ (\upsilon \wedge \varphi) \wedge (\chi \vee \psi) \rightarrow output$

$input \longrightarrow T,\ \upsilon \wedge \varphi \rightarrow T,\ \chi \vee \psi \longrightarrow output$

$input \rightarrow T,\ \upsilon \rightarrow T,\ \varphi \rightarrow T,\ \chi \vee \psi \rightarrow output$

$input \rightarrow T,\ \upsilon \rightarrow T,\ \varphi \rightarrow \boldsymbol{in}_1 \begin{matrix} \nearrow T,\ \psi \searrow \\ \searrow T,\ \chi \nearrow \end{matrix} \boldsymbol{out}_1 \rightarrow output$

Figure 3.19: Replacement and appending matrix representations. The appending matrix representation is shown in the top half of this figure and replacement (standard) matrix representation in the bottom half. The underlines in the appending version mark the vertices that have already been expanded.

Figure 3.19 shows each step in the matrix proofs of $(\varphi \Rightarrow \psi) \Rightarrow (\neg\varphi \vee \psi)$ using both the replacement and the appending matrix representations. After the final expansion, all paths through both DAGs are contradictory.

3.5 Resolution Method

Resolution, the most commonly used refutation method, [Bachmair and Ganzinger 2001], [Robinson 1965], and [Wos et al. 1984], consists of two phases, first converting the proposed theorem into a set of clauses containing only atomic formulas and second using the resolution rule to combine two clauses to create a new one. The method is successful only if the empty clause can be derived. The first step of the first phase creates an initial disjunct containing a single formula that assumes the proposed theorem is false. Each subsequent step of the first phase uses one of Smullyan's rules to expand a formula in a disjunct, replacing the clause with either one or two disjuncts.

A distinction needs to be made between the resolution method and the resolution rule. The resolution rule, described in [Robinson 1965], combines two clauses to generate a third; the resolution method, which gets its name from this rule, refers to both phases of the method. Although some literature assumes that the proposed theorem is supplied in CNF, this dissertation takes the broader view that a proof method should correctly handle any formula presented.

In this presentation of the resolution method, the signed formula notation is used even though the standard representation uses unsigned notation.

The first phase begins by creating an initial disjunct and then using Smullyan's rules to generate an equivalent set of clauses in CNF, where all clauses contain only atomic formulas. If φ is the proposed theorem, then the initial disjunct is $[\langle F, \varphi \rangle]$, which assumes that φ is false. Each subsequent step applies one of Smullyan's rules to expand a formula E in a disjunct D. An application of an unary α-rule replaces D with a disjunct D', in which E is replaced with the formula in $SM_c(E)$. An application of a binary α-rule replaces D with two disjuncts, D_1 and D_2, each of which removes E from D and adds one formula in $SM_c(E)$. An application of a β-rule replaces E in the disjunct D with the formulas

First Clause	Second Clause	Clause Produced Using Resolution Rule
$[\langle T, A\rangle, \langle T, B\rangle, \langle F, C\rangle]$	$[\langle T, C\rangle, \langle T, D\rangle, \langle T, E\rangle]$	$[\langle T, A\rangle, \langle T, B\rangle, \langle T, D\rangle, \langle T, E\rangle]$
$[\langle T, A\rangle, \langle T, B\rangle]$	$[\langle F, B\rangle, \langle T, C\rangle, \langle T, D\rangle]$	$[\langle T, A\rangle, \langle T, C\rangle, \langle T, D\rangle]$
$[\langle T, A\rangle, \langle T, B\rangle]$	$[\langle F, A\rangle, \langle F, C\rangle]$	$[\langle T, B\rangle, \langle F, C\rangle]$
$[\langle T, X\rangle, \langle T, Y\rangle]$	$[\langle F, X\rangle, \langle T, Y\rangle, \langle F, Z\rangle]$	$[\langle T, Y\rangle, \langle F, Z\rangle]$
$[\langle T, G\rangle]$	$[\langle F, G\rangle, \langle T, H\rangle, \langle F, K\rangle]$	$[\langle T, H\rangle, \langle F, K\rangle]$

Table 3.2: Examples of the application of the resolution rule. The first and second clauses, shown in the first two columns, are combined using the resolution rule to produce a new clause shown in the third column.

in $SM_c(E)$. The first phase ends when all clauses contain only atomic formulas. At this point the proposed theorem has been converted into conjunctive normal form (CNF). The disjuncts now contain only atomic formulas and thus are now clauses.

To satisfy a formula in CNF, one formula in each clause must be true. If any clause cannot be satisfied, then the entire set of clauses does not have a satisfying truth assignment. A proof is successful if an empty clause can be derived; by definition the empty clause is false.

The second phase of the method uses the resolution rule. This rule combines two clauses to generate a new one. Signed formulas that differ only in their signs are called *complementary*. The resolution rule is applied to a variable φ and a pair of clauses, one that contains the formula $\langle T, \varphi\rangle$ and the other $\langle F, \varphi\rangle$). Applying the rule creates a new clause containing all of the formulas in both clauses except for the formulas containing φ. Examples of the use of the resolution rule are shown in Table 3.2. In addition to eliminating the variable on which the resolution rule is being performed, duplicate formulas are also removed as shown in the fourth row of the table where $\langle T, Y\rangle$ appears in both input clauses but only once in the clause generated by the resolution rule.

If two clauses have more than one complementary pair in common, then the resolution rule must be applied to one pair at a time. If this restriction is ignored, then applying

resolution to $[\langle T, A\rangle, \langle T, B\rangle]$ and $[\langle F, A\rangle, \langle F, B\rangle]$, resolving on both A and B, yields the empty clause, indicating that the original two clauses cannot be satisfied. However, in this instance, assigning true to one variable and false to the other satisfies both of the original clauses.

1)	$[\langle F, (\varphi \Rightarrow \psi) \Rightarrow (\neg\varphi \vee \psi)\rangle]$	given
2)	$[\langle T, \varphi \Rightarrow \psi\rangle]$	(1, α)
3)	$[\langle F, \neg\varphi \vee \psi\rangle]$	(1, α)
4)	$[\langle F, \varphi\rangle, \langle T, \psi\rangle]$	(2, β)
5)	$[\langle F, \neg\varphi\rangle]$	(3, α)
6)	$[\langle F, \psi\rangle]$	(3, α)
7)	$[\langle T, \varphi\rangle]$	(5, α)
8)	$[\langle T, \psi\rangle]$	(4, 7, resolution)
9)	[]	(6, 8, resolution)

Figure 3.20: Resolution proof of $(\varphi \Rightarrow \psi) \Rightarrow (\neg\varphi \vee \psi)$

The resolution proof for $(\varphi \Rightarrow \psi) \Rightarrow (\neg\varphi \vee \psi)$ is shown in Figure 3.20. It starts by creating Disjunct 1, the initial disjunct $[\langle F, (\varphi \Rightarrow \psi) \Rightarrow (\neg\varphi \vee \psi)\rangle]$, containing the signed formula E_1 composed of the F sign and the proposed theorem, representing the assumption that the proposed theorem is false. Since the initial disjunct contains only E_1, the next step must expand it. Since $SM_r(E_1) = \alpha$ and $|SM_c(E_1)| = 2$, a binary α-rule is applied, adding disjuncts 2 and 3, each replacing E_1 with one formula in $SM_c(E_1) = \{\langle T, \varphi \Rightarrow \psi\rangle, \langle F, \neg\varphi \vee \psi\rangle\}$. For convenience, the first formula is referred to as E_2 and the second as E_3. Next, either formula E_2 in Disjunct 2 or E_3 in Disjunct 3 can be expanded. In the proof, E_2 in Disjunct 2 is expanded first; since $SM_r(E_2) = \beta$, a β-rule creates Disjunct 4 from Disjunct 2 by replacing E_2 with $\langle F, \varphi\rangle$ and $\langle T, \psi\rangle$. Since both formulas in Disjunct 4 are atomic, the only unexpanded non-atomic formula is E_3. The expansion of E_3 adds disjuncts 5 and 6 with the formulas $\langle F, \neg\varphi\rangle$ and $\langle F, \psi\rangle$. The only unexpanded non-atomic formula remaining is $\langle F, \neg\varphi\rangle$ in Disjunct 5. To expand it, an unary α-rule is applied, adding Disjunct 7 containing the formula $\langle T, \varphi\rangle$. At this point all formulas have

either been expanded or are atomic, so the first phase is complete; the initial disjunct has been transformed into a CNF representation consisting of disjuncts 4, 6, and 7. Since this disjuncts only contain atomic formulas these three disjuncts are clauses.

The proof now moves to the second phase, looking for pairs of clauses containing complementary formulas. The first application of the resolution rule resolves on the variable φ in clauses 4 and 7 containing $[\langle F, \varphi\rangle, \langle T, \psi\rangle]$ and $[\langle T, \varphi\rangle]$; thus the resolution rule generates clause 8, $[\langle T, \psi\rangle]$. The next application of the resolution rule resolves on the variable ψ in clauses 6 and 8, generating clause 9, an empty clause. This starts a chain reaction of reasoning. Since the empty clause has been derived, the set of clauses is not satisfiable. Because the set of clauses were derived from the initial disjunct containing only $\langle F, (\varphi \Rightarrow \psi) \Rightarrow (\neg\varphi \vee \psi)\rangle$, this initial formula must be a contradiction, and, hence, the formula $(\varphi \Rightarrow \psi) \Rightarrow (\neg\varphi \vee \psi)$ has been proven to be a theorem.

$1)[\langle T, A \vee B\rangle, \langle T, C \wedge D\rangle]$

$2x)[\langle T, A\rangle, \langle T, B\rangle, \langle T, C \wedge D\rangle]\ (1, \beta)$	$2y)[\langle T, A \vee B\rangle, \langle T, C\rangle]\ (1, \alpha)$
$3x)[\langle T, A\rangle, \langle T, B\rangle, \langle T, C\rangle]\ (2x, \alpha)$	$3y)[\langle T, A \vee B\rangle, \langle T, D\rangle]\ (1, \alpha)$
$4x)[\langle T, A\rangle, \langle T, B\rangle, \langle T, D\rangle]\ (2x, \alpha)$	$4y)[\langle T, A, \rangle, \langle T, B\rangle, \langle T, C\rangle]\ (2y, \beta)$
	$5y)[\langle T, A, \rangle, \langle T, B\rangle, \langle T, D\rangle]\ (3y, \beta)$

Table 3.3: The same set of clauses are generated from a disjunct regardless of expansion order. Columns x and y show two different expansion orders used to reduce clause 1 into a set of clauses containing only atomic formulas. Note column x requires one less expansion than column y.

In the proof in Figure 3.20, the number of expansion rules applied does not change if a different expansion order is used. However, there are situations when the expansion order does affect the number of expansions required. Table 3.3 shows an example in which two different expansion orders require a different number of steps to complete the first phase. In this example, the number of rules applied differs by one between the two orders; however,

there are situations when a change in the expansion order can make a large difference in the number of rules that need to be applied. Unlike other methods, resolution requires fewer steps if β-rules are applied before α-rules.

In the second phase, the order in which the pairs of clauses are chosen for resolution also can have a dramatic effect on the number of steps required to find a proof. In Figure 3.20 the proof has only a few pairs where resolution can be applied, but as problems grow larger, the number of pairs grows quickly. In larger problems, the majority of time is spent using the resolution rule, and hence, much research has been devoted to developing strategies to derive the empty clause more quickly.

3.6 A Mapping Between Resolution and Matrix Methods

This section shows that the empty clause can be derived using resolution, for the proposed theorem φ if and only if all paths through the appending matrix constructed for φ contain a contradiction. The section begins by defining an obstruction set, which is also a vertex cut set [West 1996]. It then proves that the initial clause in the resolution method is an obstruction set in the initial matrix. Next, it shows that as Smullyan's rules are applied in resolution and in the matrix method, each clause is an obstruction set in the corresponding matrix. Next, semantic obstruction sets are introduced; all paths through a matrix must either contain a vertex in this set or are contradictory. The resolution rule is then shown to generate clauses that are semantic obstruction sets. Thus, if resolution can derive an empty clause, then the empty set must be a semantic obstruction set; therefore, all paths through that matrix are contradictory.

Definition 1 *If M is an appending matrix DAG and $v(M)$ is the set of vertices in M, then a set $S \subseteq v(M)$ is an* obstruction set *for M if every path through M contains at least one vertex in S.*

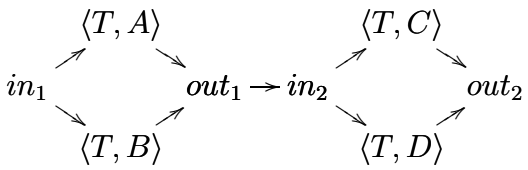


Figure 3.21: Matrix representation of $\langle T, (A \vee B) \wedge (C \vee D)\rangle$

The clauses generated by Smullyan rules for the formula $\langle T, (A \vee B) \wedge (C \vee D)\rangle$ are $[\langle T, A\rangle, \langle T, B\rangle]$ and $[\langle T, C\rangle, \langle T, D\rangle]$. This matrix has four paths through it $(\langle T, A\rangle, \langle T, C\rangle)$, $(\langle T, A\rangle, \langle T, D\rangle)$, $(\langle T, B\rangle, \langle T, C\rangle)$ and $(\langle T, B\rangle, \langle T, D\rangle)$. Every path through the DAG contains at least one formula from each of the clauses; thus each clause is an obstruction set.

Smullyan's rules decompose formulas based on their syntactical form. In the first phase of resolution, Smullyan's rules are applied to convert the proposed theorem into CNF. In the second phase, the resolution rule combines two clauses (sets of formulas) that contain a pair of complementary formulas. Applying this rule creates a new clause that has all of the formulas in the two clauses except for the complementary formulas.

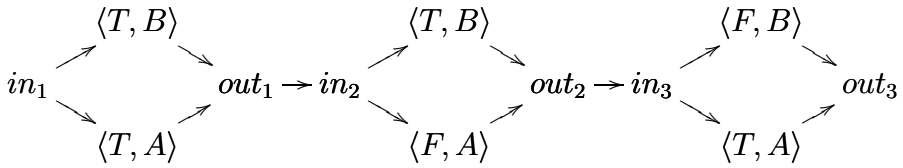


Figure 3.22: Matrix representation of $\langle T, (A \vee B) \wedge (\neg A \vee B) \wedge (A \vee \neg B)\rangle$

The clauses $[\langle T, A\rangle, \langle T, B\rangle]$, $[\langle F, A\rangle, \langle T, B\rangle]$ and $[\langle T, A\rangle, \langle F, B\rangle]$ are equivalent to the matrix shown in Figure 3.22. Each path through the DAG contains one formula from each clause. The goal of a refutation method is to show that there are no satisfying truth assignments to the variables in the proposed theorem. Thus, any path that contains both a formula and its complement can be eliminated from consideration. The DAG in the Figure 3.22 has eight paths through it, many of which are contradictory. When the resolution rule is applied to the variable A in the clauses $[\langle T, A\rangle, \langle T, B\rangle]$ and

$[\langle F, A\rangle, \langle T, B\rangle]$, the clause $[\langle T, B\rangle, \langle T, B\rangle]$ is generated. After removing the duplicate formula, the clause $[\langle T, B\rangle]$ remains. This new clause is added to the set of clauses. Thus, any non-complementary path must contain $\langle T, B\rangle$. All paths contain either a vertex labeled with $\langle T, B\rangle$ or contain a vertex labeled with $\langle T, A\rangle$ and another vertex labeled with $\langle F, A\rangle$. Thus, the resolution rule has eliminated formulas from the clause that, if chosen, would lead to paths containing complementary formulas.

Definition 2 *An* expansion order *is a sequence of vertices from the formula tree such that the parent of a vertex must appear in the sequence before its children. If O is an expansion order and O_i is the i^{th} vertex in the formula tree, then the formula that labels O_i is expanded in step i. O_0 is $\langle F, \varphi\rangle$ where φ is the proposed theorem.*

The first phase of both the matrix and the resolution methods reduce the proposed theorem into atomic formulas. After this first phase, the matrix method attempts to show that all maximal paths contain contradictory formulas, while the resolution method uses the resolution rule to combine clauses in an attempt to generate the empty clause.

Theorem 3 *For a fixed expansion order O, after expansion step i, let $\mathcal{C}_i$ be the set of disjuncts and M_i be the appending matrix. If $C \in \mathcal{C}_i$, then C is an obstruction set in M_i.*

Proof: (by induction on the number of expansion steps)

Base Case: If φ is the proposed theorem, then the initial set of disjuncts has only one member $C_0 = \{[\langle F, \varphi\rangle]\}$, while the initial matrix has only one formula vertex labeled with $\langle F, \varphi\rangle$. Hence, C_0 is an obstruction set for the initial matrix.

Induction Hypothesis: After n expansion steps, every disjunct in $\mathcal{C}_n$ is an obstruction set in M_n.

Induction Step: Let E be a non-atomic signed formula labeling the vertex O_{n+1} in the formula tree. Suppose C is a disjunct in $\mathcal{C}_{n+1}$ that contains E. This formula is expanded using either an α or β-rule.

Case E is expanded by applying a binary α-rule: The application of the rule generates two subformulas, denoted by α_1 and α_2. In the resolution method, if C is the disjunct that contains E, then expanding E adds two new disjuncts C_1 and C_2. These two disjuncts contain all the formulas in C except E, which is replaced by α_1 in C_1 and α_2 in C_2.

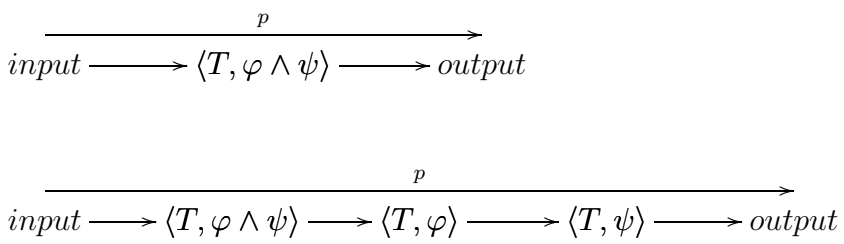


Figure 3.23: Binary α matrix rule. Example of a binary α-rule applied in an appending matrix. The expansion inserts two new vertices labeled with the formulas in $SM_c(E)$. These vertices are the immediate successors of the original vertex.

In the matrix method, if a vertex v is labeled with E as shown in Figure 3.23, then expanding v inserts a path containing two vertices as its immediate successor. These new vertices are labeled with α_1 and α_2. Any path that passed through v must now pass through both of these new vertices labeled with α_1 and α_2. Thus, both C_1 and C_2 are obstruction sets.

Case E is expanded by applying a unary α-rule: The application of the rule generates a new formula α' in $SM_c(E)$.

In resolution, if E is in disjunct C, then expanding E adds a new disjunct C', which is created by replacing E in C with α', i.e. $C' = (C \setminus \{E\}) \cup \{\alpha'\}$. In the matrix method, a vertex v' labeled with α' is inserted as the immediate successor of v. Since any path that passed through v must now also pass through v', the new disjunct is an obstruction set.

Case E is expanded by applying a β-rule: Let β_1 and β_2 be two new signed formulas in $SM_c(E)$. In resolution, if E is in disjunct C, then expanding E adds a new disjunct C'. The disjunct C' is obtained by replacing E with β_1 and β_2, i.e. $C' = (C \setminus \{E\}) \cup \{\beta_1, \beta_2\}$.

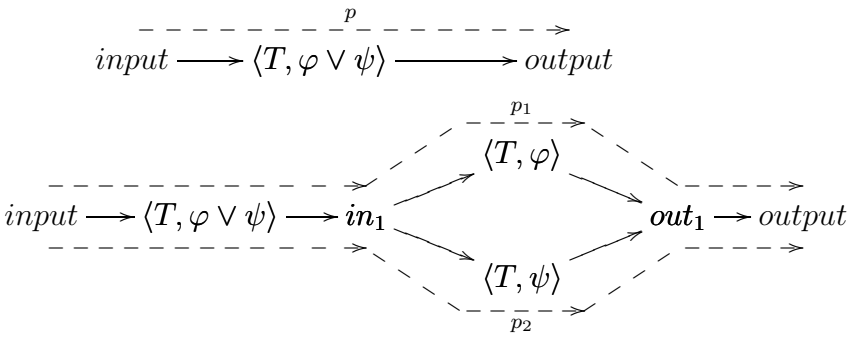


Figure 3.24: Matrix β-rule. Expansion of the β formula $\langle T, \varphi \vee \psi \rangle$ in an appending matrix. This figure illustrates a β-rule expansion, the matrix before (top) the expansion and after (bottom) the expansion. This expansion inserts two new vertices labeled with subformulas. The bottom matrix contains the new paths p_1 and p_2.

In the matrix method, if the vertex v labeled with the formula E is expanded, then a diamond containing v_1 and v_2 is inserted as the immediate successor of v, as shown in Figure 3.24. Suppose $SM_c(E) = \{\beta_1, \beta_2\}$ are the new formulas that label v_1 and v_2 in this diamond. In the subgraph, v_1 and v_2 are on separate paths; thus any path that passed through v must now also pass through either v_1 or v_2. Hence, disjunct C' is an obstruction set since it contains both β_1 and β_2.

Each case above checks one of Smullyan's three rule types, and in each the resolution rule creates a new disjunct(s) that is an obstruction sets. □

Let Ψ be a function that maps a clause to a set of vertices. The vertex $v \in \Psi(C)$ if and only if v is labeled with a signed formula in clause C.

$$\Psi(C) = \{v \mid \text{ the formula labeling } v \text{ is in } C\} \tag{3.10}$$

Since the same formula may label more than one vertex in the matrix, $|C| \leq |\Psi(C)|$. Let G be the graph of the matrix and $V(G)$ be the set of vertices in G.

Definition 4 *The set S of formulas is a* semantic obstruction set *if all paths through a matrix either contain a vertex labeled with a formula in S or contain contradictory formulas.*

If $\Psi(S) \subseteq V(G)$ is a semantic obstruction set, then the set of maximal paths (a path between $input$ and $output$) can be divided into those that are complementary and those that contain a vertex in S. If the empty set is a semantic obstruction set, then all paths through $V(G)$ must be complementary.

Recall that the matrix method proves a theorem by showing that all paths through DAG G are contradictory. Thus, if it can shown that the empty set is a semantic obstruction set, then the the matrix method is successful.

Theorem 5 *If two clauses are semantic obstruction sets that contain a complementary pair of formulas, then the clause generated by using the resolution rule on these two clauses is also a semantic obstruction set.*

Proof:

Suppose C_1 and C_2 are clauses that are both semantic obstruction sets and for a variable φ, C_1 contains $\langle T, \varphi \rangle$ and C_2 contains $\langle F, \varphi \rangle$. By resolving on φ, clauses C_1 and C_2 are combined to generate clause C_r, where $C_r = (C_1 \cup C_2) \setminus \{\langle T, \varphi \rangle, \langle F, \varphi \rangle\}$.

Assume that C_r is not a semantic obstruction set. Then there exists a non-contradictory path p through the matrix that does not pass through a vertex labeled with a formula in C_r. Since C_1 and C_2 are semantic obstruction sets, p must contain a vertex v_1 in $\Psi(C_1)$ and a vertex v_2 in $\Psi(C_2)$. Now, v_1 must be labeled with $\langle T, \varphi \rangle$ because $\Psi(C_1) \setminus \Psi(C_r) = \Psi(\langle T, \varphi \rangle)$. By a similar argument, v_2 is labeled with $\langle F, \varphi \rangle$ because $\Psi(C_2) \setminus \Psi(C_r) = \Psi(\langle F, \varphi \rangle)$. But this requires p to contain vertices labeled with contradictory formulas, which contradicts the assumption that C_r is not a semantic obstruction set. $\square$

3.7 Summary of Methods used in Classical Logic

This chapter describes the five methods explored in this research: natural deduction, Kripke C-Tableau, analytic tableau, matrix, and resolution. This chapter ends by showing a transformation between the matrix and the resolution methods. The descriptions of the methods will be used in Chapter 4 where the container for classical logic and the generalized algorithm for classical logic are presented.

Chapter 4

The Container and Generalized Algorithm for Classical Logic

Of the five methods discussed in Chapter 3, four share the same underlying procedure; resolution uses a different one. In this chapter, the logical container is introduced, and for each of the four methods, a proof is presented, showing that the method's data structure has the properties of the container. Next, a generalized algorithm is presented which uses only the operations provided by a container to prove theorems in classical logic. Thus, this algorithm captures the commonalities of these four methods.

4.1 Logical Container

A *logical container* $\langle C, S \rangle$ consists of a data structure C that stores logical formulas and a function S that maps a data structure to a set of signed formulas. In the following sections, proofs are presented, demonstrating that sequents are containers in natural deduction, branches are containers in analytic tableaux, boxes are containers in Kripke C-tableaux, and paths are containers in matrices.

A logical container is composed of a data structure C that stores formulas and a function S having the following properties:

LC1– Two Sets Logical formulas stored in a container C can be separated into two sets T and F, denoted as $T(C)$ and $F(C)$.

LC2– S **function** The S function maps logical formulas in C to a set of signed formulas, where the formulas with a T sign come from the $T(C)$ set and the formulas with an F sign come from the $F(C)$ set.

LC3– Initial Container When the initial container is created, it contains only the proposed theorem in its F set.

LC4– Expansion Suppose that C is a container and E is a signed formula in $S(C)$:

LC4a– α **expansion** If $SM_r(E) = \alpha$, then there is an expansion rule that adds the formula(s) in $SM_c(E)$ to C. If C' represents the container after the expansion, then $S(C') = S(C) \cup SM_c(E)$.

LC4b– β **expansion** If $SM_r(E) = \beta$, then there is an expansion rule that splits C into two new containers C_1 and C_2. Suppose E_1 and E_2 are the signed formulas in $SM_c(E)$, then the contents of these new containers are $S(C_1) = S(C) \cup \{E_1\}$ and $S(C_2) = S(C) \cup \{E_2\}$.

LC4c– Only α **and** β **expansion rules** All rules which change the contents of a container or which create new containers must fall into one of these two categories or be decomposed into a sequence of rules from these two categories.

LC5– Closure conditions A container C is closed when its contents become contradictory. A container is contradictory if one of three conditions exist: 1) there exists a formula in both $T(C)$ and $F(C)$, 2) $\bot \in T(C)$, or 3) $\top \in F(C)$.

LC6– Non-reopening Once a container becomes closed, it remains closed. Expansion rules may applied to formulas in a closed container, but the changes made cannot cause the container to reopen. If an expansion rule splits a closed container, then both of the new containers are closed.

LC7– Success A proof succeeds if and only if the application of expansion rules causes all of the containers to close.

4.2 Extending Set Operator Definitions to Containers

To simplify the notation and facilitate the discussion, the definitions of the operators set union ($\cup$) and set difference ($\setminus$) are extended to containers. Recall that set union combines the contents of two sets, eliminating duplicates. The union of two containers C_1 and C_2 is a container C with $T(C) = T(C_1) \cup T(C_2)$ and $F(C) = F(C_1) \cup F(C_2)$. Recall that set difference removes the members of the set on the right of the operator from the set on the left of the operator. This definition is extended to containers by performing a set difference operation between the T sets and another between the F sets (i.e. $T(C_1 \setminus C_2) = T(C_1) \setminus T(C_2)$ and $F(C_1 \setminus C_2) = F(C_1) \setminus F(C_2)$).

In addition, the definition of union $\cup$ is also extended to a mixture of a signed formula and a container. To accomplish this, a signed formula will be treated as a container with only one formula. A signed formula of the form $\langle T, \varphi \rangle$ will be treated as a container with φ in its T set and an empty F set (i.e. $T(\langle T, \varphi \rangle) = \{\langle T, \varphi \rangle\}$ and $F(\langle T, \varphi \rangle) = \emptyset$), while a signed formula of the form $\langle F, \psi \rangle$ is a container with ψ in its F set and an empty T set (i.e. $T(\langle F, \psi \rangle) = \emptyset$ and $F(\langle F, \psi \rangle) = \{\langle F, \psi \rangle\}$).

4.3 The Containers Used by the Proof Methods

This section presents proofs that demonstrate that each proof method, except resolution, uses a data structure that has the properties of a container. Resolution is omitted because it differs from the other methods; it represents logical formulas as a conjunction of disjunctions, whereas the other methods represent formulas as a disjunction of conjunctions. The container property was designed for this latter representation.

4.3.1 A Branch in an Analytic Tableau

The proof in this subsection demonstrates that a branch in an analytic tableau meets all the properties of a container. In order to facilitate the discussion, the function $B2C$ (short for branch to container) is introduced:

$$B2C(b) = \{E \mid E \text{ labels a vertex on } b\} \tag{4.1}$$

If b is a branch in an analytic tableau, then $B2C(b)$ is the set of the signed formulas that labels the vertices on b.

Theorem 6 *A branch in an analytic tableau is a container.*

Proof:

For each property (LC1- LC7) of the container, a brief argument proving that the branch has that property is presented.

The signed formulas that label the vertices on a branch can be separated into two sets based on their signs. The T set contains formulas with a T sign, while the formulas with an F sign are placed in the F set. This fulfills the two set property (LC1).

The $B2C$ function, defined by Equation 4.1, maps a branch to the set of signed formulas that labels its vertices. This function fulfills the S function property (LC2).

Suppose that φ is the proposed theorem. The initial tableau consists of only the root vertex labeled with $\langle F, \varphi \rangle$; hence, there is one branch b which contains this vertex. Since $B2C(b) = \{\langle F, \varphi \rangle\}$, the branch meets the initial container property (LC3).

The expansion properties (LC4a and LC4b) are demonstrated by examining how the analytic tableau uses expansion rules to create and modify branches. Suppose that E is the formula that labels a vertex on branch b and v_ℓ is the leaf vertex of b. If a unary α-rule is used to expand E, then a single vertex labeled with the formula in $SM_c(E)$ is added as the child of v_ℓ. If a binary α-rule is used to expand E, then two new vertices labeled with the formulas in $SM_c(E)$ are added as the child and grandchild of v_ℓ. In both cases, $B2C(b') = B2C(b) \cup SM_c(E)$, where b' is the branch after the expansion. This meets the α expansion property (LC4a).

If a β-rule is used to expand E, then two vertices are added as the children of v_ℓ, splitting branch b into two branches b_1 and b_2. The leaves of these two branches are labeled with the formulas in $SM_c(E)$. Let E_1 and E_2 be the formulas in $SM_c(E)$. WLOG, it can be assumed that E_1 labels the leaf of b_1 and E_2 labels the leaf of b_2. Hence, $B2C(b_1) = B2C(b) \cup \{E_1\}$ and $B2C(b_2) = B2C(b) \cup \{E_2\}$, meeting the β expansion property (LC4b).

The third property (LC4c) requires that all rules that can create or change the contents of a branch meet the α or β expansion rules. Since this method uses only α and β-rules to change the contents of the tableau, it meets the *only α and β* property (LC4c).

A branch b is closed if $B2C(b)$ contains $\langle T, \bot \rangle$, $\langle F, \top \rangle$, or two signed formulas $\langle T, \varphi \rangle$ and $\langle F, \varphi \rangle$ for some unsigned formula φ. These are the same conditions that cause a container to close; thus the branch meets the closure property (LC5).

Expansions performed on a closed branch cannot cause it to reopen since every expansion rule adds formulas to a branch. On a closed branch b, if an α-rule expands a formula, vertices are added to the branch. If b' represents b after the expansion, then $B2C(b) \subseteq B2C(b')$. Since $B2C(b)$ is contradictory, $B2C(b')$ must also be contradic-

tory. On the other hand, if on a closed branch b a β-rule expands a formula, b is split into b_1 and b_2, with a new vertex added to each branch; thus, $B2C(b) \subseteq B2C(b_1)$ and $B2C(b) \subseteq B2C(b_2)$. Since $B2C(b)$ is contradictory, $B2C(b_1)$ and $B2C(b_2)$ must both be contradictory and, thus, closed. Therefore, the branch meets the non-reopening property (LC6).

An analytic tableau proof succeeds only if all of its branches close. Hence, it meets the success property (LC7).

The branch in an analytic tableau has met every property of a container. □

4.3.2 A Path Through a Matrix

In this subsection, a lemma and a theorem demonstrate that a path through a matrix meets the properties of a container. To facilitate these proofs, the function $P2C$ is introduced. For a path p through a matrix, $P2C(p)$ (short for path to container) is the set of signed formulas that label vertices on p:

$$P2C(p) = \{E \mid E \text{ labels a vertex on } p\} \tag{4.2}$$

Lemma 7 *Using $P2C$ as its S function, a path through a matrix meets the expansion properties (LC4a, LC4b, and LC4c) of a container.*

Proof:

Each time a formula E labeling a vertex v_m on path p is expanded, one of three rule types is applied, unary α, binary α, or β. Thus, there are three cases to consider.

Case 1: If $SM_r(E) = \alpha$ and $|SM_c(E)| = 1$, then an unary α-rule expands E. The new path p' is obtained from p by inserting a new vertex as the successor of v_m. This new vertex is labeled with the formula in $SM_c(E)$; hence, $P2C(p') = P2C(p) \cup SM_c(E)$.

Case 2: If $SM_r(E) = \alpha$ and $|SM_c(E)| = 2$, then a binary α-rule expands E. The new path p' is obtained from p by inserting as the successor of v_m a path containing two new vertices, each labeled with a formula in $SM_c(E)$; hence, $P2C(p') = P2C(p) \cup SM_c(E)$.

Together cases 1 and 2 show that when $SM_r(E) = \alpha$, the path meets the α expansion rule property (LC4a).

Case 3: If $SM_r(E) = \beta$, then a β-rule expands E. The β-rule inserts a diamond as the successor of v_m, splitting p into two paths p_1 and p_2. Let E_1 and E_2 be the formulas in $SM_c(E)$. WLOG, assume that the vertex unique to p_1 is labeled with E_1, and the vertex unique to p_2 is labeled with E_2. Hence, $P2C(p_1) = P2C(p) \cup \{E_1\}$ and $P2C(p_2) = P2C(p) \cup \{E_2\}$, meeting the β expansion rule property (LC4b).

Since a path through a matrix is only created or modified using one of these rule types, this satisfies the only α and β expansion rule property (LC4c). □

Theorem 8 *A path though an appending DAG representation of the matrix is a container.*

Proof:

For each property (LC1- LC7) of a container, a brief justification is presented demonstrating that a path through a matrix meets that property.

The signed formulas that label the vertices on a path can be separated into two sets based on their signs. The T set contains formulas that have a T sign, while the F set contains formulas that have an F sign, meeting the two set property (LC1).

The function $P2C$ maps a path to the set of signed formulas that label it. This function meets the S function property (LC2).

Suppose that φ is the proposed theorem. The only path through the initial matrix contains one labeled vertex. This vertex is labeled with $\langle F, \varphi \rangle$, and, thus, the initial path meets the initial container property (LC3).

Lemma 7 proves that the path meets the expansion properties (LC4a-c).

A path p is closed if $P2C(b)$ contains $\langle T, \bot \rangle$, $\langle F, \top \rangle$ or two signed formulas $\langle T, \varphi \rangle$ and $\langle F, \varphi \rangle$ for an unsigned formula φ. Thus, a path through a matrix meets the closure property (LC5).

On a closed path p, any expansion made will always add but never remove signed formulas from $P2C(p)$. If a β-rule is used to expand a vertex on p, then p is split into two paths p_1 and p_2. Since $P2C(p)$ is contradictory, $P2C(p) \subseteq P2C(p_1)$ and $P2C(p) \subseteq P2C(p_2)$, both $P2C(p_1)$ and $P2C(p_2)$ must also be contradictory, meeting the non-reopening property (LC6).

The matrix method succeeds if all paths through the matrix are contradictory (closed), and, hence, it meets success property (LC7).

A path through a matrix meets every property of the container. □

4.3.3 A Sequent in Natural Deduction Proof

This subsection proves that a sequent in natural deduction is a container. It begins with a discussion of how the sequent rewrite rules can be viewed as inferences or reductions. To facilitate the discussion, it introduces the SSF (sequent side formula) representation of sequents. It concludes by using tree lemmas and a theorem to prove that a sequent in natural deduction is a container.

A natural deduction proof tree may be created using rewrite rules as either inferences or reductions. When rewrite rules are used as inferences, the proof tree is constructed by working from axioms at the leaves towards the endsequent at the root. However, when rewrite rules are used as reductions, the proof tree is constructed by working from the proposed theorem at the root towards axioms at the leaves.

When a completed proof tree is presented, it is impossible to determine if it was constructed using sequent rules as inferences or reductions. To prove that a sequent in

natural deduction is a container, it will be assumed that the natural deduction proof tree was constructed using reduction rules. In addition, for technical reasons it is assumed that the formula expanded in the conclusion is retained in the premise(s). As long as a formula is expanded only once, this modification does not affect the correctness of natural deduction.

To simplify the discussion of sequents, the *SSF* (sequent side formula) representation is introduced. The standard representation of sequents is two sets of unsigned formulas separated by a sequent arrow. The set on the left of the arrow is the antecedent, and the set on the right is the succedent. But in the SSF representation, a sequent is represented by a set of ordered pairs. The first component of these ordered pairs is either $ante$ or $succ$, indicating whether the unsigned formula in the second component is in the antecedent or in the succedent. Figure 4.1 presents a sequent in both the standard representation and the SSF set representation. The SSF representation was chosen both to facilitate the proofs that follow and to resemble the signed formula notation.

$$\varphi, \psi, \rho \longrightarrow \chi, \sigma, \tau \equiv \{\langle ante, \varphi\rangle, \langle ante, \psi\rangle, \langle ante, \rho\rangle, \langle succ, \chi\rangle, \langle succ, \sigma\rangle\langle succ, \tau\rangle\}$$

$$\varphi \wedge \psi \longrightarrow \rho, \sigma \vee \tau \equiv \{\langle ante, \varphi \wedge \psi\rangle, \langle succ, \rho\rangle, \langle succ, \sigma \vee \tau\rangle\}$$

$$\varphi \wedge \psi \longrightarrow \emptyset \equiv \{\langle ante, \varphi \wedge \psi\rangle\}$$

Figure 4.1: Three examples of sequents using both representations. This figure shows three different sequents using two different representations, the standard and the SSF.

To facilitate the proofs, two functions SF_n and $S2C$ (short for sequent to container) are introduced. The function SF_n defined by Equation 4.3 maps an SSF formula to a signed

formula. The other function $S2C$ maps a sequent represented as a set of SSF formulas to a set of signed formulas as defined in Equation 4.4.

$$SF_n(E) = \begin{cases} \langle T, E' \rangle & \text{if } E \text{ is of the form } \langle ante, E' \rangle \\ \langle F, E' \rangle & \text{if } E \text{ is of the form } \langle succ, E' \rangle \end{cases} \tag{4.3}$$

$$S2C(K) = \{\langle T, \varphi \rangle \mid \langle ante, \varphi \rangle \in K\} \cup \{\langle F, \varphi \rangle \mid \langle succ, \varphi \rangle \in K\} \tag{4.4}$$

The function $S2C$ could equivalently be defined using SF_n by the equation $S2C(K) = \{SF_n(E) | E \in K\}$.

The rewrite rule used to expand a formula E in natural deduction is determined by both E's primary connective ($\wedge$, $\vee$, $\Rightarrow$, or $\neg$) and the side of the sequent arrow where E appears. Combining the four logical connectives with the two possible sides of the sequent, there are eight cases to consider in the following lemmas.

Lemma 9 *If E is a non-atomic SSF formula, then the sequent rule used to expand E has one premise if $SM_r(SF_n(E)) = \alpha$ and two premises if $SM_r(SF_n(E)) = \beta$.*

Proof:

If a formula E is chosen for expansion, there are eight possible sequent rewrite rules, one for each of the eight combinations of the two sides of the sequent arrow and the four primary connectives.

Case 1. E is of the form $\langle ante, \varphi \wedge \psi \rangle$:

For this case $l\wedge$ is the sequent rule $\dfrac{\Gamma, \varphi, \psi \longrightarrow \Delta}{\Gamma, \varphi \wedge \psi \longrightarrow \Delta}\, l\wedge$.

This rule has one premise, and

$$SM_r(SF_n(E)) = SM_r(SF_n(\langle ante, \varphi \wedge \psi \rangle)) = SM_r(\langle T, \varphi \wedge \psi \rangle) = \alpha$$

Case 2. E is of the form $\langle succ, \varphi \wedge \psi \rangle$:

For this case $r\wedge$ is the sequent rule $\dfrac{\Gamma \longrightarrow \varphi, \Delta \quad \Gamma \longrightarrow \psi, \Delta}{\Gamma \longrightarrow \varphi \wedge \psi, \Delta}\, r\wedge$.

This rule has two premises, and

$$SM_r(SF_n(E)) = SM_r(SF_n(\langle succ, \varphi \wedge \psi \rangle)) = SM_r(\langle F, \varphi \wedge \psi \rangle) = \beta$$

Case 3. E is of the form $\langle ante, \varphi \vee \psi \rangle$:

For this case $l\vee$ is the sequent rule $\dfrac{\Gamma, \varphi \longrightarrow \Delta \quad \Gamma, \psi \longrightarrow \Delta}{\Gamma, \varphi \vee \psi \longrightarrow \Delta}\, l\vee$.

This rule has two premises, and

$$SM_r(SF_n(E)) = SM_r(SF_n(\langle ante, \varphi \vee \psi \rangle)) = SM_r(\langle T, \varphi \vee \psi \rangle) = \beta$$

Case 4. E is of the form $\langle succ, \varphi \vee \psi \rangle$:

For this case $r\vee$ is the sequent rule $\dfrac{\Gamma \longrightarrow \varphi, \psi, \Delta}{\Gamma \longrightarrow \varphi \vee \psi, \Delta}\, r\vee$.

This rule has one premise, and

$$SM_r(SF_n(E)) = SM_r(SF_n(\langle succ, \varphi \vee \psi \rangle)) = SM_r(\langle F, \varphi \vee \psi \rangle) = \alpha$$

Case 5. E is of the form $\langle ante, \varphi \Rightarrow \psi \rangle$:

For this case $l \Rightarrow$ is the sequent rule $\dfrac{\Gamma \longrightarrow \varphi, \Delta \quad \Gamma, \psi \longrightarrow \Delta}{\Gamma, \varphi \Rightarrow \psi \longrightarrow \Delta}\, l \Rightarrow$.

This rule has two premises, and

$$SM_r(SF_n(E)) = SM_r(SF_n(\langle ante, \varphi \Rightarrow \psi \rangle)) = SM_r(\langle T, \varphi \Rightarrow \psi \rangle) = \beta$$

Case 6. E is of the form $\langle succ, \varphi \Rightarrow \psi \rangle$:

For this case $r \Rightarrow$ is the sequent rule $\dfrac{\Gamma, \varphi \longrightarrow \psi, \Delta}{\Gamma \longrightarrow \varphi \Rightarrow \psi, \Delta}\, r \Rightarrow$.

This rule has one premise, and

$$SM_r(SF_n(E)) = SM_r(SF_n(\langle succ, \varphi \Rightarrow \psi\rangle)) = SM_r(\langle F, \varphi \Rightarrow \psi\rangle) = \alpha$$

Case 7. E is of the form $\langle ante, \neg\varphi\rangle$:
For this case $l\neg$ is the sequent rule $\dfrac{\Gamma \longrightarrow \varphi, \Delta}{\Gamma, \neg\varphi \longrightarrow \Delta}\, l\neg$.
This rule has one premise, and

$$SM_r(SF_n(E)) = SM_r(SF_n(\langle ante, \neg\varphi\rangle)) = SM_r(\langle T, \neg\varphi\rangle) = \alpha$$

Case 8. E is of the form $\langle succ, \neg\varphi\rangle$:
For this case $r\neg$ is the sequent rule $\dfrac{\Gamma, \varphi \longrightarrow \Delta}{\Gamma \longrightarrow \neg\varphi, \Delta}\, r\neg$.
This rule has one premise, and

$$SM_r(SF_n(E)) = SM_r(SF_n(\langle succ, \neg\varphi\rangle)) = SM_r(\langle F, \neg\varphi\rangle) = \alpha$$

In all eight cases if $SM_r(SF_n(E)) = \alpha$, then the sequent rule for E has one premise, but when $SM_r(SF_n(E)) = \beta$, then the sequent rule for E has two premises. □

Lemma 9 established a correspondence between the Smullyan rule type and the number of premises in a sequent rule. Lemma 10 establishes that the formulas added to the premise(s) by the sequent rewrite rules for E can be converted using $S2C$ to the signed formulas that Smullyan's rules add to a container when expanding $SF_n(E)$.

Lemma 10 *In a sequent rule if E is the SSF formula in the conclusion that is expanded and $\mathcal{E}$ is the set of formulas added to the conclusion to obtain the premise sequent(s), then $S2C(\mathcal{E}) = SM_c(SF_n(E))$.*

Proof:

Let E represent the SSF formula in the conclusion which is rewritten to obtain the premise(s). Then the conclusion C can be written as the set $\{E\} \cup \Gamma \cup \Delta$, where Γ is the set of formulas in the antecedent and Δ is the set of formulas in the succedent that are not modified by the rule's application.

The choice of the rewrite rule depends on whether E is in the antecedent or succedent and on E's primary connective. The proof consists of eight cases, one for each combination of the two sides of the sequent arrow and the four logical connectives.

Case 1. E is of the form $\langle ante, \varphi \wedge \psi \rangle$:

For this case $l\wedge$ is the sequent rule $\dfrac{\Gamma, \varphi, \psi \longrightarrow \Delta}{\Gamma, \varphi \wedge \psi \longrightarrow \Delta}\, l\wedge$.

In this case, the premise is obtained by adding $\langle ante, \varphi \rangle$ and $\langle ante, \psi \rangle$ to C:

$$S2C(\mathcal{E}) = S2C(\{\langle ante, \varphi \rangle, \langle ante, \psi \rangle\}) = \{\langle T, \varphi \rangle, \langle T, \psi \rangle\}$$
$$= SM_c(\langle T, \varphi \wedge \psi \rangle) = SM_c(SF_n(\langle ante, \varphi \wedge \psi \rangle)) = SM_c(SF_n(E))$$

Case 2. E is of the form $\langle succ, \varphi \wedge \psi \rangle$:

For this case $r\wedge$ is the sequent rule $\dfrac{\Gamma \longrightarrow \varphi, \Delta \quad \Gamma \longrightarrow \psi, \Delta}{\Gamma \longrightarrow \varphi \wedge \psi, \Delta}\, r\wedge$.

In this case, one premise is obtained by adding $\langle succ, \varphi \rangle$ to C, and the other premise is obtained by adding $\langle succ, \psi \rangle$ to C:

$$S2C(\mathcal{E}) = S2C(\{\langle succ, \varphi \rangle, \langle succ, \psi \rangle\}) = \{\langle F, \varphi \rangle, \langle F, \psi \rangle\}$$
$$= SM_c(\langle F, \varphi \wedge \psi \rangle) = SM_c(SF_n(\langle succ, \varphi \wedge \psi \rangle)) = SM_c(SF_n(E))$$

Case 3. E is of the form $\langle ante, \varphi \vee \psi \rangle$:

For this case $l\vee$ is the sequent rule $\dfrac{\Gamma, \varphi \longrightarrow \Delta \quad \Gamma, \psi \longrightarrow \Delta}{\Gamma, \varphi \vee \psi \longrightarrow \Delta}\, l\vee$.

In this case, one premise is obtained by adding $\langle ante, \varphi \rangle$ to C, and the other premise

is obtained by adding $\langle ante, \psi \rangle$ to C:

$$S2C(\mathcal{E}) = S2C(\{\langle ante, \varphi \rangle, \langle ante, \psi \rangle\}) = \{\langle T, \varphi \rangle, \langle T, \psi \rangle\}$$
$$= SM_c(\langle T, \varphi \vee \psi \rangle) = SM_c(SF_n(\langle ante, \varphi \vee \psi \rangle)) = SM_c(SF_n(E))$$

Case 4. E is of the form $\langle succ, \varphi \vee \psi \rangle$:

For this case $r\vee$ is the sequent rule $\dfrac{\Gamma \longrightarrow \varphi, \psi, \Delta}{\Gamma \longrightarrow \varphi \vee \psi, \Delta}\, r\vee$.

In this case, the premise is obtained by adding $\langle succ, \varphi \rangle$ and $\langle succ, \psi \rangle$ to C:

$$S2C(\mathcal{E}) = S2C(\{\langle succ, \varphi \rangle, \langle succ, \psi \rangle\}) = \{\langle F, \varphi \rangle, \langle F, \psi \rangle\}$$
$$= SM_c(\langle F, \varphi \vee \psi \rangle) = SM_c(SF_n(\langle succ, \varphi \vee \psi \rangle)) = SM_c(SF_n(E))$$

Case 5. E is of the form $\langle ante, \varphi \Rightarrow \psi \rangle$:

For this case $l \Rightarrow$ is the sequent rule $\dfrac{\Gamma \longrightarrow \varphi, \Delta \quad \Gamma, \psi \longrightarrow \Delta}{\Gamma, \varphi \Rightarrow \psi \longrightarrow \Delta}\, l \Rightarrow$.

In this case, one premise is obtained by adding $\langle succ, \varphi \rangle$ to C, and the other premise is obtained by adding $\langle ante, \psi \rangle$ to C:

$$S2C(\mathcal{E}) = S2C(\{\langle succ, \varphi \rangle, \langle ante, \psi \rangle\}) = \{\langle F, \varphi \rangle, \langle T, \psi \rangle\}$$
$$= SM_c(\langle T, \varphi \Rightarrow \psi \rangle) = SM_c(SF_n(\langle ante, \varphi \Rightarrow \psi \rangle)) = SM_c(SF_n(E))$$

Case 6. E is of the form $\langle succ, \varphi \Rightarrow \psi \rangle$:

For this case $r \Rightarrow$ is the sequent rule $\dfrac{\Gamma, \varphi \longrightarrow \psi, \Delta}{\Gamma \longrightarrow \varphi \Rightarrow \psi, \Delta}\, r \Rightarrow$.

In this case, the premise is obtained by adding $\langle ante, \varphi \rangle$ and $\langle succ, \psi \rangle$ to C:

$$S2C(\mathcal{E}) = S2C(\{\langle ante, \varphi \rangle, \langle succ, \psi \rangle\}) = \{\langle T, \varphi \rangle, \langle F, \psi \rangle\}$$
$$= SM_c(\langle F, \varphi \Rightarrow \psi \rangle) = SM_c(SF_n(\langle succ, \varphi \Rightarrow \psi \rangle)) = SM_c(SF_n(E))$$

Case 7. E is of the form $\langle ante, \neg\varphi\rangle$:

For this case $l\neg$ is the sequent rule $\dfrac{\Gamma \longrightarrow \varphi, \Delta}{\Gamma, \neg\varphi \longrightarrow \Delta}\, l\neg$.
In this case, the premise is obtained by adding $\langle succ, \varphi\rangle$ to C:

$$S2C(\mathcal{E}) = S2C(\{\langle succ, \varphi\rangle\}) = \{\langle F, \varphi\rangle\}$$
$$= SM_c(\langle T, \neg\varphi\rangle) = SM_c(SF_n(\langle ante, \neg\varphi\rangle)) = SM_c(SF_n(E))$$

Case 8. E is of the form $\langle succ, \neg\varphi\rangle$:

For this case $r\neg$ is the sequent rule $\dfrac{\Gamma, \varphi \longrightarrow \Delta}{\Gamma \longrightarrow \neg\varphi, \Delta}\, r\neg$.
In this case, the premise is obtained by adding $\langle ante, \varphi\rangle$ to C:

$$S2C(\mathcal{E}) = S2C(\{\langle ante, \varphi\rangle\}) = \{\langle T, \varphi\rangle\}$$
$$= SM_c(\langle F, \neg\varphi\rangle) = SM_c(SF_n(\langle succ, \neg\varphi\rangle)) = SM_c(SF_n(E))$$

□

Lemma 11 *The sequent rules of classical logic shown in Figure 3.2 meet the expansion properties (LC4a- LC4c) of the container.*

Proof:

Lemma 9 proves that when a rewrite rule is used to expand an SSF formula E, the number of premises depends on the value of $SM_r(SF_n(E))$; when $SM_r(SF_n(E)) = \alpha$, the expansion generates one premise, but when $SM_r(SF_n(E)) = \beta$, the expansion generates two premises. Lemma 10 proves that the new formulas added to the premise(s) by the sequent rules correspond to the formulas generated by Smullyan's rules. Together the results of these two lemmas demonstrate that the sequent in the natural deduction method meets the α and β expansion properties (LC4a and LC4b) of the container.

Since every sequent rewrite rule was examined in Lemma 9 were found to have either one or two premises, all of the rules fall into α and β-rule types. This meets the only α and β-rules property (LC4c). $\square$

Theorem 12 *The sequents of the natural deduction have the properties of the container.*

Proof:

All of the formulas in a sequent are in one of two sets, the antecedent or succedent, meeting the two set property (LC1).

The function $S2C$ maps a sequent to a set of signed formulas. Suppose that K is a sequent. If E is a formula in the antecedent of K, then $\langle T, E\rangle \in S2C(K)$. However, if E is in the succedent of K, then $\langle F, E\rangle \in S2C(K)$. Since the T set contains formulas in the antecedent and the F set contains formulas in succedent, $S2C$ has the S function property (LC2).

Suppose that φ is the proposed theorem. The endsequent K_0 at the root of the natural deduction proof contains only the formula φ in its succedent, i.e. $S2C(K_0) = \{\langle F, \varphi\rangle\}$. Hence, the sequent meets the initial container property (LC3).

For all eight sequent reduction rules shown in Figure 3.2, lemmas 9, 10, and 11 prove that these rules have the expansion rule properties (LC4a- LC4c).

A sequent K is an axiom if $S2C(K)$ contains 1) $\langle T, \bot\rangle$ ($\bot$ appears in the antecedent), 2) $\langle F, \top\rangle$ ($\top$ appears in the succedent), or 3) two signed formulas $\langle T, \varphi\rangle$ and $\langle F, \varphi\rangle$ (φ appears in both the antecedent and succedent). This meets the closure property (LC5).

If an expansion rule is applied to an axiom, then the new sequent(s) created remain axioms because all formulas are preserved in the application of the sequent rules. Thus, sequents in the natural deduction method have the non-reopening property (LC6).

If all of the leaf sequents in the proof tree are axioms, then the proposed theorem has been proven, meeting the success property (LC7).

It has been shown that the sequent in natural deduction has all the properties of the container. □

4.3.4 A Box in a Kripke C-Tableau

This subsection presents two lemmas and a proof demonstrating that a box in the Kripke C-tableau is a container. To facilitate the proofs, two functions SF_k and $K2C$ are introduced. The function SF_k defined by Equation 4.5 maps a formula in a Kripke box to a signed formula. The function $K2C$ (short for Kripke to container) maps the formulas in a box to a set of signed formulas as described in Equation 4.6.

$$SF_k(E) = \begin{cases} \langle T, E' \rangle & \text{if } E \text{ is of the form } \langle left, E' \rangle \\ \langle F, E' \rangle & \text{if } E \text{ is of the form } \langle right, E' \rangle \end{cases} \tag{4.5}$$

$$K2C(b) = \{SF_k(E) \mid E \in b\} \tag{4.6}$$

Lemma 13 *Expanding a formula E splits the Kripke tableau box, $KR_r(E) = split$ if and only if $SM_r(SF_k(E)) = \beta$.*

Proof:

When a KCF formula E is expanded, one of eight expansion rules is used, one for each combination of the four primary connectives and the two columns. Each combination is checked individually.

Case 1. E is of the form $\langle left, \varphi \wedge \psi \rangle$:

$KR_r(E) = no\ split$, so this rule box does **not** split the box, and

$SM_r(SF_k(\langle left, \varphi \wedge \psi \rangle)) = SM_r(\langle T, \varphi \wedge \psi \rangle) = \alpha$

Case 2. E is of the form $\langle right, \varphi \wedge \psi \rangle$:

$KR_r(E) = split$, so this rule **splits** the box, and

$SM_r(SF_k(\langle right, \varphi \wedge \psi \rangle)) = SM_r(\langle F, \varphi \wedge \psi \rangle) = \beta$

Case 3. E is of the form $\langle left, \varphi \vee \psi \rangle$:

$KR_r(E) = split$, so this rule **splits** the box, and

$SM_r(SF_k(\langle left, \varphi \vee \psi \rangle)) = SM_r(\langle T, \varphi \vee \psi \rangle) = \beta$

Case 4. E is of the form $\langle right, \varphi \vee \psi \rangle$:

$KR_r(E) = no\ split$, so this rule box does **not** split the box, and

$SM_r(SF_k(\langle right, \varphi \vee \psi \rangle)) = SM_r(\langle F, \varphi \vee \psi \rangle) = \alpha$

Case 5. E is of the form $\langle left, \varphi \Rightarrow \psi \rangle$:

$KR_r(E) = split$, so this rule **splits** the box, and

$SM_r(SF_k(\langle left, \varphi \Rightarrow \psi \rangle)) = SM_r(\langle T, \varphi \Rightarrow \psi \rangle) = \beta$

Case 6. E is of the form $\langle right, \varphi \Rightarrow \psi \rangle$:

$KR_r(E) = no\ split$, so this rule box does **not** split the box, and

$SM_r(SF_k(\langle right, \varphi \Rightarrow \psi \rangle)) = SM_r(\langle F, \varphi \Rightarrow \psi \rangle) = \alpha$

Case 7. E is of the form $\langle left, \neg\varphi \rangle$:

$KR_r(E) = no\ split$, so this rule box does **not** split the box, and

$SM_r(SF_k(\langle left, \neg\varphi \rangle)) = SM_r(\langle T, \neg\varphi \rangle) = \alpha$

Case 8. E is of the form $\langle right, \neg\varphi \rangle$:

$KR_r(E) = no\ split$, so this rule box does **not** split the box, and

$SM_r(SF_k(\langle right, \neg\varphi \rangle)) = SM_r(\langle F, \neg\varphi \rangle) = \alpha$

In all eight cases, expanding E splits a box if and only if $SM_r(SF_k(E)) = \beta$. $\square$

Now the issue of whether expanding a KCF formula E generates the same formulas as expanding $SF_k(E)$ in a container is addressed. This is expressed by the equality $K2C(KR_c(E)) = SM_c(SF_k(E))$.

Lemma 14 *For each non-atomic KCF formula E, $SM_c(SF_k(E)) = K2C(KR_c(E))$.*

Proof: There are eight Kripke expansion rules, one for each combination of the two columns and the four connectives. When expanding a non-atomic KCF formula E, the rule applied is determined by E's column and E's primary connective. Each of the eight combinations is checked separately below.

Case 1. E is of the form $\langle left, \varphi \wedge \psi \rangle$:

$$\begin{aligned} SM_c(SF_k(E)) &= SM_c(SF_k(\langle left, \varphi \wedge \psi \rangle)) = SM_c(\langle T, \varphi \wedge \psi \rangle) \\ &= \{\langle T, \varphi \rangle, \langle T, \psi \rangle\} = K2C(\{\langle left, \varphi \rangle, \langle left, \psi \rangle\}) \\ &= K2C(KR_c(\langle left, \varphi \wedge \psi \rangle)) = K2C(KR_c(E)) \end{aligned}$$

Case 2. E is of the form $\langle right, \varphi \wedge \psi \rangle$:

$$\begin{aligned} SM_c(SF_k(E)) &= SM_c(SF_k(\langle right, \varphi \wedge \psi \rangle)) = SM_c(\langle F, \varphi \wedge \psi \rangle) \\ &= \{\langle F, \varphi \rangle, \langle F, \psi \rangle\} = K2C(\{\langle right, \varphi \rangle, \langle right, \psi \rangle\}) \\ &= K2C(KR_c(\langle right, \varphi \wedge \psi \rangle)) = K2C(KR_c(E)) \end{aligned}$$

Case 3. E is of the form $\langle left, \varphi \vee \psi \rangle$:

$$\begin{aligned} SM_c(SF_k(E)) &= SM_c(SF_k(\langle left, \varphi \vee \psi \rangle)) = SM_c(\langle T, \varphi \vee \psi \rangle)) \\ &= \{\langle T, \varphi \rangle, \langle T, \psi \rangle\} = K2C(\{\langle left, \varphi \rangle, \langle left, \psi \rangle\}) \\ &= K2C(KR_c(\langle left, \varphi \vee \psi \rangle)) = K2C(KR_c(E)) \end{aligned}$$

Case 4. E is of the form $\langle right, \varphi \vee \psi \rangle$:

$$\begin{aligned} SM_c(SF_k(E)) &= SM_c(SF_k(\langle right, \varphi \vee \psi \rangle)) = SM_c(\langle F, \varphi \vee \psi \rangle) \\ &= \{\langle F, \varphi \rangle, \langle F, \psi \rangle\} = K2C(\{\langle right, \varphi \rangle, \langle right, \psi \rangle\}) \\ &= K2C(KR_c(\langle right, \varphi \vee \psi \rangle)) = K2C(KR_c(E)) \end{aligned}$$

Case 5. E is of the form $\langle left, \varphi \Rightarrow \psi \rangle$:

$$\begin{aligned} SM_c(SF_k(E)) &= SM_c(SF_k(\langle left, \varphi \Rightarrow \psi \rangle)) = SM_c(\langle T, \varphi \Rightarrow \psi \rangle) \\ &= \{\langle F, \varphi \rangle, \langle T, \psi \rangle\} = K2C(\{\langle right, \varphi \rangle, \langle left, \psi \rangle\}) \\ &= K2C(KR_c(\langle left, \varphi \Rightarrow \psi \rangle)) = K2C(KR_c(E)) \end{aligned}$$

Case 6. E is of the form $\langle right, \varphi \Rightarrow \psi \rangle$:

$$\begin{aligned} SM_c(SF_k(E)) &= SM_c(SF_k(\langle right, \varphi \Rightarrow \psi \rangle)) = SM_c(\langle F, \varphi \Rightarrow \psi \rangle) \\ &= \{\langle T, \varphi \rangle, \langle F, \psi \rangle\} = K2C(\{\langle left, \varphi \rangle, \langle right, \psi \rangle\}) \\ &= K2C(KR_c(\langle right, \varphi \Rightarrow \psi \rangle)) = K2C(KR_c(E)) \end{aligned}$$

Case 7. E is of the form $\langle left, \neg\varphi \rangle$:

$$\begin{aligned} SM_c(SF_k(E)) &= SM_c(SF_k(\langle left, \neg\varphi \rangle)) = SM_c(\langle T, \neg\varphi \rangle) \\ &= \{\langle F, \varphi \rangle\} = K2C(\{\langle right, \varphi \rangle\}) \\ &= K2C(KR_c(\langle left, \neg\varphi \rangle)) = K2C(KR_c(E)) \end{aligned}$$

Case 8. E is of the form $\langle right, \neg\varphi \rangle$:

$$\begin{aligned} SM_c(SF_k(E)) &= SM_c(SF_k(\langle right, \neg\varphi \rangle)) = SM_c(\langle F, \neg\varphi \rangle) \\ &= \{\langle T, \varphi \rangle\} = K2C(\{\langle left, \varphi \rangle\}) \\ &= K2C(KR_c(\langle right, \neg\varphi \rangle)) = K2C(KR_c(E)) \end{aligned}$$

For each of the eight rules, the equality $SM_c(SF_k(E)) = K2C(KR_c(E))$ holds, and, thus, the lemma has been proven. □

Theorem 15 *The box in a Kripke C-tableau proof is a container.*

Proof:

Each logical formula is in one of the two columns. The T set consists of logical formulas in the left column, and the F set consists of logical formulas in the right column, thereby meeting the two set property (LC1).

The $K2C$ function described above maps a KCF formula to a set of signed formulas such that a KCF formula with $left$ as its column is mapped to a signed formula with a T flag, while a KCF formula with $right$ as its column is mapped to a signed formula with a F flag. $K2C$ has the S function property (LC2).

Suppose that φ is the proposed theorem. The root box b_0 at the start of a Kripke C-tableau proof contains only the formula φ in right column of b_0 (i.e. $K2C(b_0) = \{\langle F, \varphi\rangle\}$). Hence, the box meets the initial container property (LC3).

Lemma 13 proves that expanding a KCF formula E splits a Kripke box ($KR_r(E) = split$) if and only if $SM_r(SF_k(E)) = \beta$. Lemma 14 proves that formulas generated by expanding E in a Kripke C-tableau match those generated by expanding $SF_k(E)$ in a container. Together these two lemmas demonstrate that the expansion rules for the box meet the α and β expansion properties (LC4a and LC4b). The cases considered in lemmas 13 and 14 cover all of the rules used to alter or create a Kripke box. This fulfills the only α and β expansion rules property (LC4c).

A box b is closed if $K2C(b)$ contains $\langle T, \bot\rangle$, $\langle F, \top\rangle$, or two signed formulas $\langle T, \varphi\rangle$ and $\langle F, \varphi\rangle$ for an unsigned formula φ, meeting the closure property (LC5).

Since formulas are not removed from a box, it cannot be reopened once it has been closed. Further, when a rule splits a closed box, since all formulas are copied to the new boxes, these new boxes must also be closed. This meets the non-reopening property (LC6).

The Kripke C-tableau method successfully proves a theorem if only if all of its boxes close. Thus, it meets the success property (LC7).

The box in a Kripke C-tableau meets all of the properties of the container. □

4.4 Generalized Algorithm

In this section, an algorithm is presented that decides theoremhood using only the properties of a logical container. The disjunctive methods (natural deduction, Kripke C-tableau, analytic tableau, and matrix) all use data structures that have the container property, and each method is a different representation of the generalized algorithm.

Algorithm 1 classCheckForClosure(C: Container) returns an empty set or a set with one container

if $\top \in F$ or $\bot \in T$ or a formula appears in both F and T **then**
  return $\emptyset$ { return an empty set, the container is closed}
else
  return $\{C\}$
end if

Algorithm 2 classExpandFormula(E:Signed Formula, C: Container) returns a set of containers

if $SM_r(E) = \alpha$ **then**
  $C \leftarrow C \cup SM_c(E)$
  $\mathcal{S} \leftarrow$ CheckForClosure(C)
else { $SM_r(E) = \beta$ }
  $\{\beta_1, \beta_2\} \leftarrow SM_c(E)$
  $C_1 \leftarrow C \cup \{\beta_1\}$
  $C_2 \leftarrow C \cup \{\beta_2\}$
  $\mathcal{S} \leftarrow$ CheckForClosure(C_1) $\cup$ CheckForClosure(C_2)
end if
return $\mathcal{S}$

Algorithm 3 classProver($\mathcal{S}$: a set of containers) returns theorem or non-theorem

while at least one container C in $\mathcal{S}$ is open **do**
  if all formulas in C have been expanded or atomic **then**
    return non-theorem
  else { Choose a formula and expand it}
    Remove C from $\mathcal{S}$
    Let E be an unexpanded, non-atomic formula in C
    $\mathcal{S} \leftarrow (\mathcal{S} \setminus C) \cup \mathrm{classExpandFormula}(\ \mathrm{E}\ ,\ \mathrm{C}\ )$
  end if
end while
if all of the containers are closed **then**
  return theorem
end if

Algorithm 4 classProofStart(φ: Logical Formula) returns theorem or non-theorem. This algorithm working with algorithms to determine if φ is a theorem.

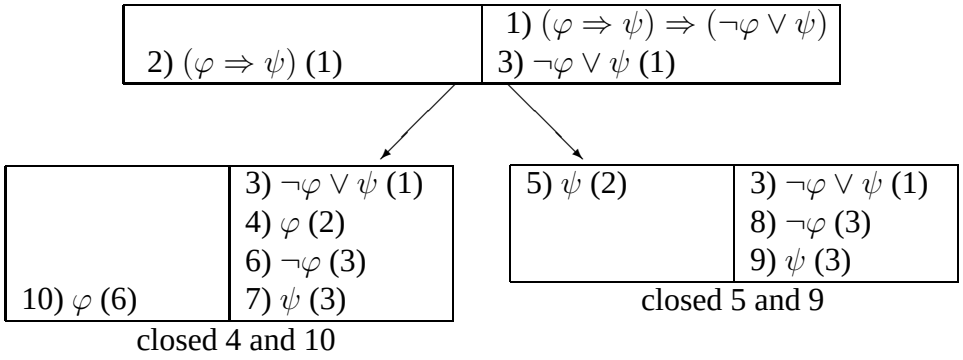
$C \leftarrow \langle \emptyset, \{\varphi\} \rangle$ {Construct a container C with its T set empty and its F set containing φ.}
$\mathcal{S} \leftarrow classCheckForClosure(\{C\})$
return classProver($\mathcal{S}$)

The generalized algorithm starts when `classProofStart` is called. This algorithm is passed the proposed theorem φ and returns either theorem or non-theorem. This algorithm initializes the data structures for the other algorithms. It creates the initial container and the set of open containers $\mathcal{S}$ containing only this initial container, and then calls `classProver`. The algorithm `classProver` contains the main loop of the proof. This loop consists of choosing an open container with an unexpanded formula E, calling `classExpandFormula` to expand E and checking if a termination condition has been met. If all containers are closed, then the algorithm terminates successfully, having proven the theorem. However, if there is an open container with no unexpanded formulas, then the algorithm terminates unsuccessfully (the proof has failed). The algorithm `classExpandFormula` expands a formula, adding formulas to the container and splitting it if necessary. After the expansion, `classExandFormula` returns only those containers that are still open. The final algorithm, `classCheckforClosure` is passed a container C and returns a set of containers; if C is open, it returns a set containing C, but if C is closed, it returns an empty set indicating that C is closed and can be removed from the set of open containers $\mathcal{S}$.

Four proofs for the formula $(\varphi \Rightarrow \psi) \Rightarrow (\neg\varphi \vee \psi)$, each constructed by one of the classical logic methods (resolution is excluded), are shown in Figures 4.2 and 4.3. The Kripke C-tableau and natural deduction proofs both use five expansions, while the analytic tableau and matrix proofs both use four expansions.

These methods use different numbers of expansions. Since these methods all share the same algorithm, it is necessary to explore the reasons for the differences and add rules to establish numerical parity between the methods. In the next three subsections the causes of these numerical differences are explained, and rules are added that lead to a numerical parity between the expansions of these four methods.

Kripke Tableaux



Natural Deduction

$$\dfrac{\dfrac{\dfrac{\varphi \longrightarrow \varphi, \psi}{\longrightarrow \varphi, \neg\varphi, \psi}\, r\neg}{\longrightarrow \varphi, \neg\varphi \vee \psi}\, r\vee \qquad \dfrac{\dfrac{\psi \longrightarrow \neg\varphi, \psi}{\psi \longrightarrow \neg\varphi \vee \psi}\, r\vee}{\varphi \Rightarrow \psi \longrightarrow \neg\varphi \vee \psi}\, l\Rightarrow}{\longrightarrow (\varphi \Rightarrow \psi) \Rightarrow (\neg\varphi \vee \psi)}\, r\Rightarrow$$

Figure 4.2: First of two figures showing the five methods used to prove the theorem $(\varphi \Rightarrow \psi) \Rightarrow (\neg\varphi \vee \psi)$ in classical logic.

4.4.1 Maximum Expansion

When most methods find a contradiction, they stop, possibly leaving formulas unexpanded. If these formulas were expanded, they may create subproofs (containers) and would require expansions to be repeated in each container. Since containers are used in different ways by different methods, it is important to require *maximum expansion*, that is to expand every non-atomic formula possible. This often means performing expansions in a closed container and possibly splitting one.

Maximum expansion was not used in Figures 4.2 and 4.3. In the natural deduction proof, another expansion could have been applied, on the right branch, reducing $\psi \longrightarrow \neg\varphi, \psi$ to $\psi, \varphi \longrightarrow \psi$. This step was not taken because the sequent was already

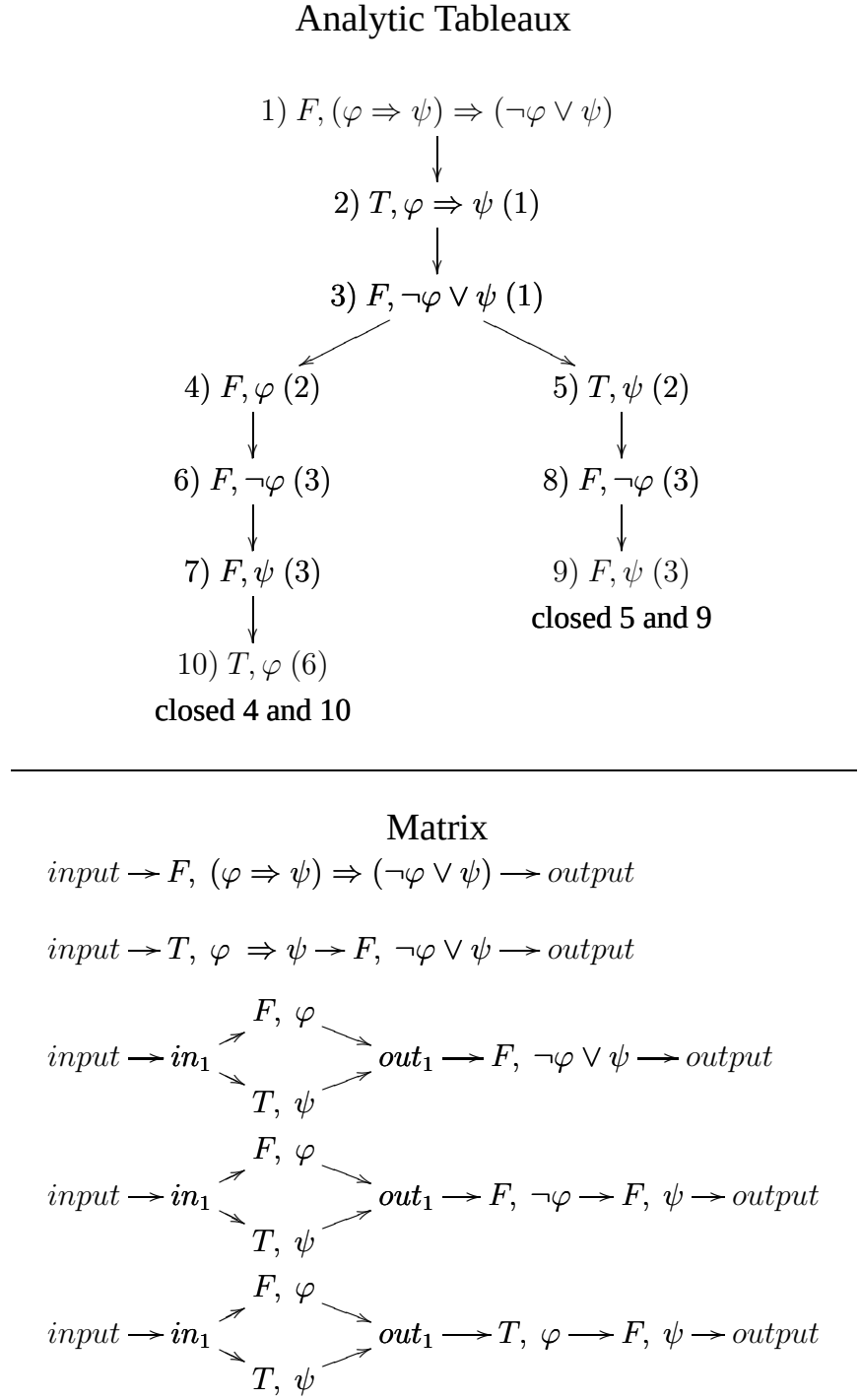


Figure 4.3: Second of two figures showing the different methods used to prove the theorem $(\varphi \Rightarrow \psi) \Rightarrow (\neg\varphi \vee \psi)$ in classical logic.

an axiom. However, it is important to count it. Similarly in the Kripke C-tableau proof, Formula 8, $\neg\varphi$, in the right column of the right box could also have been expanded, adding φ to the left column of the same box. In the analytic tableau, the right branch closed, leaving the non-atomic formula $\langle F, \neg\varphi \rangle$ on Vertex 8 unexpanded. Again, it is important to include these unexpanded formulas in the count. These adjustments bring the number of expansions in the analytic tableau to five and in natural deduction and Kripke C-tableau to six. Using the adjusted counts, the matrix method used four steps, while all of the other methods used five or six steps.

4.4.2 Different Expansion Orders

There are often multiple choices for the expansion order of a given formula. Some expansion orders lead to the same formula being expanded in separate containers. This occurs when multiple formulas can be expanded at the same time as E, but one or more β-formulas are chosen before E. Each application of a β-rule splits the container, creating another copy of E. Finally, when E is expanded, there are several copies present in multiple containers. However, if E is expanded before the β-formulas, then E's children are copied each time a β-rule is applied. Subsection 4.4.3 introduces a grouping rule that eliminates these multiple expansions.

A *formula tree* [Wallen 1990] is a tree of signed formulas for the proposed theorem φ. The root vertex is labeled with $\langle F, \varphi \rangle$. A vertex labeled with the formula E has no children if E is atomic. However, if E is non-atomic, the vertex's children are labeled with the formulas in $SM_c(E)$. A signed formula cannot appear in a proof unless its parent has already appeared.

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{M)\ C,E,F \longrightarrow A,B}{K)\ C,\neg B,E,F \longrightarrow A}\,l\neg}{I)\ C,\neg A,\neg B,E,F \longrightarrow}\,l\neg}{G)\ C,\neg A\wedge\neg B,E,F \longrightarrow}\,l\wedge}{E)\ C,\neg A\wedge\neg B,E\wedge F \longrightarrow}\,l\wedge \qquad \dfrac{\dfrac{\dfrac{\dfrac{N)\ D,E,F \longrightarrow A,B}{L)\ D,E\wedge F \longrightarrow A,B}\,l\wedge}{J)\ D,\neg B,E\wedge F \longrightarrow A}\,l\neg}{H)\ D,\neg A,\neg B,E\wedge F \longrightarrow}\,l\neg}{F)\ D,\neg A\wedge\neg B,E\wedge F \longrightarrow}\,l\wedge}{D)\ \neg A\wedge\neg B, C\vee D, E\wedge F \longrightarrow}\,l\vee}{C)\ (\neg A\wedge\neg B)\wedge(C\vee D), E\wedge F \longrightarrow}\,l\wedge}{B)\ ((\neg A\wedge\neg B)\wedge(C\vee D))\wedge(E\wedge F) \longrightarrow}\,l\wedge}{A)\ \longrightarrow \neg(((\neg A\wedge\neg B)\wedge(C\vee D))\wedge(E\wedge F))}\,l\neg
$$

$$
\begin{array}{c}
1)\ F,\ \neg((((\neg A)\wedge(\neg B))\wedge(C\vee D))\wedge(E\wedge F))\\
\downarrow\\
2)\ T,\ (((\neg A)\wedge(\neg B))\wedge(C\vee D))\wedge(E\wedge F)\ (1)\\
\downarrow\\
3)\ T,\ ((\neg A)\wedge(\neg B))\wedge(C\vee D)\ (2)\\
\downarrow\\
4)\ T,\ (E\wedge F)\ (2)\\
\downarrow\\
5)\ T,\ (\neg A)\wedge(\neg B)\ (3)\\
\downarrow\\
6)\ T,\ (C\vee D)\ (3)\\
\swarrow \qquad\qquad \searrow\\
\begin{array}{cc}
7)\ T, C\ (6) & 8)\ T, D\ (6)\\
\downarrow & \downarrow\\
9)\ T,\ E\ (4) & 11)\ T,\ \neg A\ (5)\\
\downarrow & \downarrow\\
10)\ T,\ F\ (4) & 12)\ T,\ \neg B\ (5)\\
\downarrow & \downarrow\\
13)\ T,\ \neg A\ (5) & 15)\ F,\ A\ (11)\\
\downarrow & \downarrow\\
14)\ T,\ \neg B\ (5) & 16)\ F,\ B\ (12)\\
\downarrow & \downarrow\\
17)\ F,\ A\ (13) & 19)\ T,\ E\ (4)\\
\downarrow & \downarrow\\
18)\ F,\ B\ (14) & 20)\ T,\ F\ (4)
\end{array}
\end{array}
$$

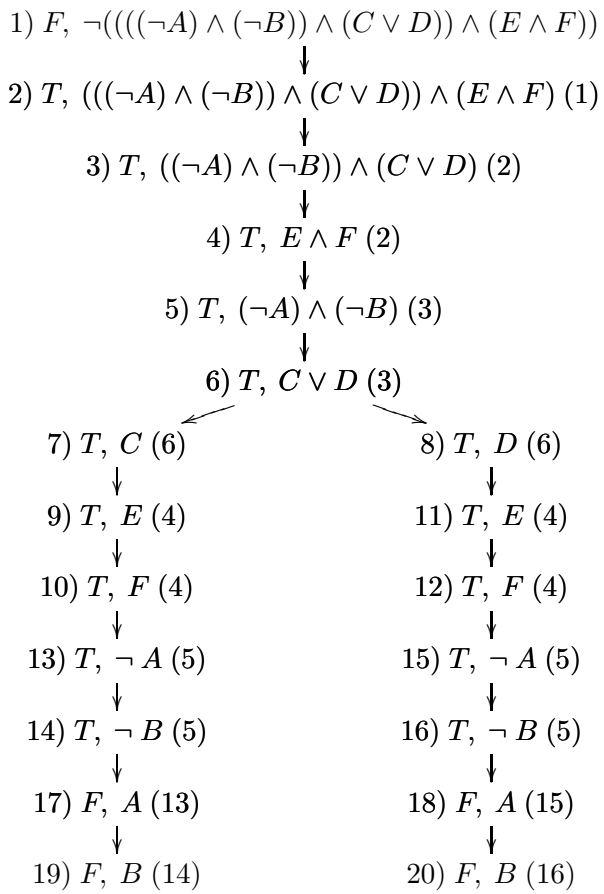
This is **not** a proper analytic tableau. The order was chosen to match the natural deduction proof but violates the rules of tableau construction. Vertices 4 and 5 are on both branches but are expanded in different orders on the two branches.

Sequent	Analytic Tableau Vertex
A	1
B	2
C	3 and 4
D	5 and 6
E	7
F	8
G	9 and 10
H	11 and 12
I	13 and 14
J	15
K	17
L	16
M	18
N	19 and 20

Figure 4.4: An analytic tableau proof violates its construction rules by trying to use the same expansion order used in the a natural deduction proof. Vertices 4 and 5 are on both branches; on the left branch Vertex 4 is expanded before Vertex 5, while on the right branch they are expanded in the opposite order.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{W)\ C, E, F \longrightarrow A, B}{U)\ C, \neg B, E, F \longrightarrow A}\, l\neg}{S)\ C, \neg A, \neg B, E, F \longrightarrow}\, l\neg}{Q)\ C, \neg A \wedge \neg B, E, F \longrightarrow}\, l\wedge \qquad \dfrac{\dfrac{\dfrac{X)\ D, E, F \longrightarrow A, B}{V)\ D, \neg B, E, F \longrightarrow A}\, l\neg}{T)\ D, \neg A, \neg B, E, F \longrightarrow}\, l\neg}{R)\ D, \neg A \wedge \neg B, E, F \longrightarrow}\, l\wedge}{O)\ C, \neg A \wedge \neg B, E \wedge F \longrightarrow}\, l\wedge \qquad P)\ D, \neg A \wedge \neg B, E \wedge F \longrightarrow}{D)\ \neg A \wedge \neg B, C \vee D, E \wedge F \longrightarrow}\, l\vee$$

$$\dfrac{\dfrac{\dfrac{C)\ (\neg A \wedge \neg B) \wedge (C \vee D), E \wedge F \longrightarrow}{B)\ ((\neg A \wedge \neg B) \wedge (C \vee D)) \wedge (E \wedge F) \longrightarrow}\, l\wedge}{A)\ \longrightarrow \neg(((\neg A \wedge \neg B) \wedge (C \vee D)) \wedge (E \wedge F))}\, l\neg$$



This is a proper analytic tableau. The order for the natural deduction proof above was chosen to allow it to be mirrored in the tableau expansion.

Sequent	**Corresponding Analytic Tableau Vertex**
A	1
B	2
C	3 and 4
D	5 and 6
O	7
P	8
Q	9 and 10
R	11 and 12
S	13 and 14
T	15 and 16
U	17
V	18
W	19
X	20

Figure 4.5: The tableau at the bottom and the natural deduction at the top use the same expansion order. In contrast to Figure 4.4, this tableau does **not** violate the construction rules for analytic tableau.

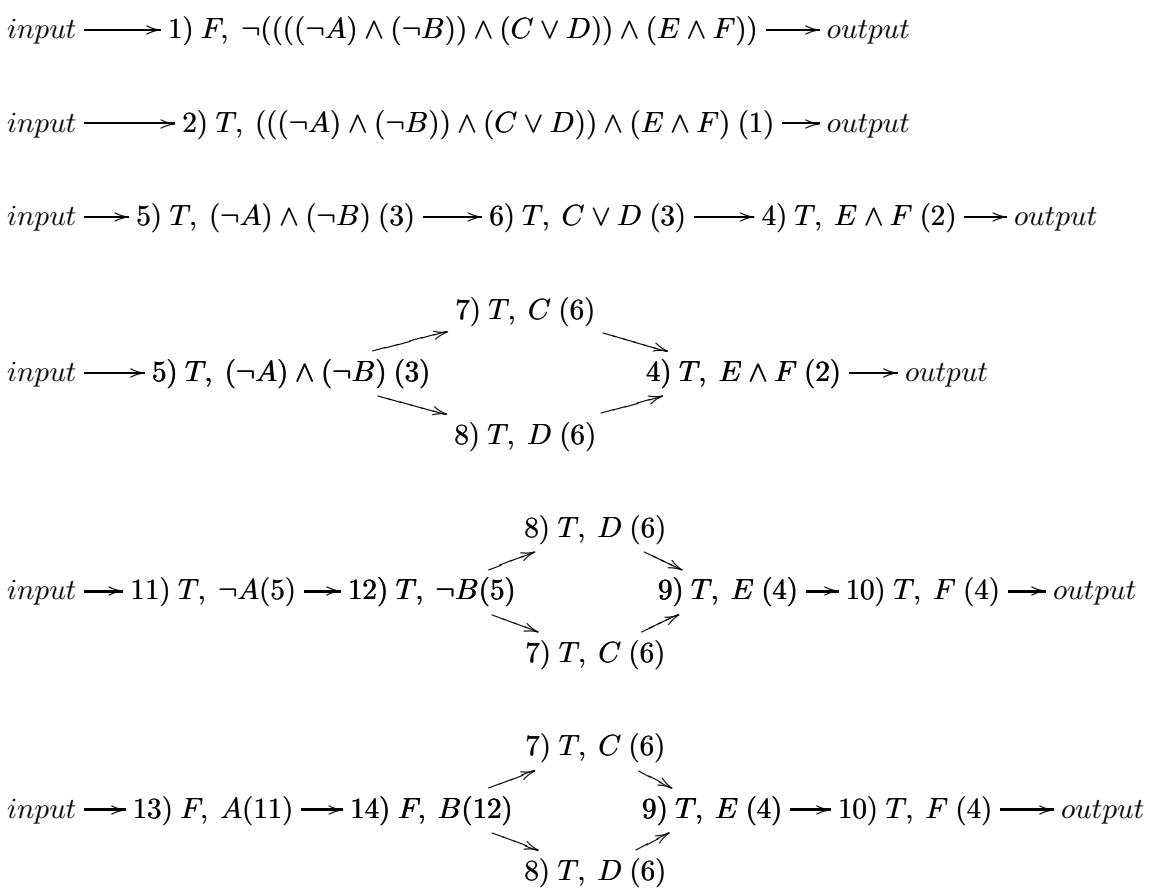


Figure 4.6: The construction of a matrix using the same expansion order as natural deduction and analytic tableau proofs in Figure 4.5.

4.4.3 Grouping in Expansion Orders

In some methods, a formula labeling a vertex in the formula tree appears in more than one container. This formula is expanded separately in each container. To illustrate this, the sequence of actions taken to expand $\neg\varphi \vee \psi$ in each of the methods is examined. In the analytic tableau proof in Figure 4.3, when Vertex 3, labeled with $\langle F, \neg\varphi \vee \psi\rangle$, is expanded, vertices 6 and 7 are added under Vertex 4, and vertices 8 and 9 are added under Vertex 5. This single expansion adds four new vertices since Vertex 3 is on two branches; i.e there are two leaf vertices beneath it. In the Kripke C-tableau (shown in Figure 4.2), Formula 3 ($\neg\varphi \vee \psi$) is expanded twice since it appears in two containers. In one expansion, expanding

Formula 3 in the left box adds formulas 6 and 7. In another expansion, expanding Formula 3 in the right box adds formulas 8 and 9. The natural deduction proof also exhibits this behavior as it expands $\neg\varphi \vee \psi$ twice, once in the left subtree and again in the right subtree. The analytic tableau and matrix methods both expand $\neg\varphi \vee \psi$ once.

Also in natural deduction and Kripke C-tableau, expansions in different containers may occur in different orders. The natural deduction proof in Figure 4.4 shows a situation where $E \wedge F$ is expanded before $\neg A \wedge \neg B$ in the left subtree and the opposite order is used in the right subtree. The Kripke C-tableau (not shown in the figure) is able to use different expansion orders in each box, but the analytic tableau cannot without violating its construction rules. However, in Figure 4.5 where natural deduction uses the same expansion order in both subtrees, a proper analytic tableau using this expansion order can be constructed.

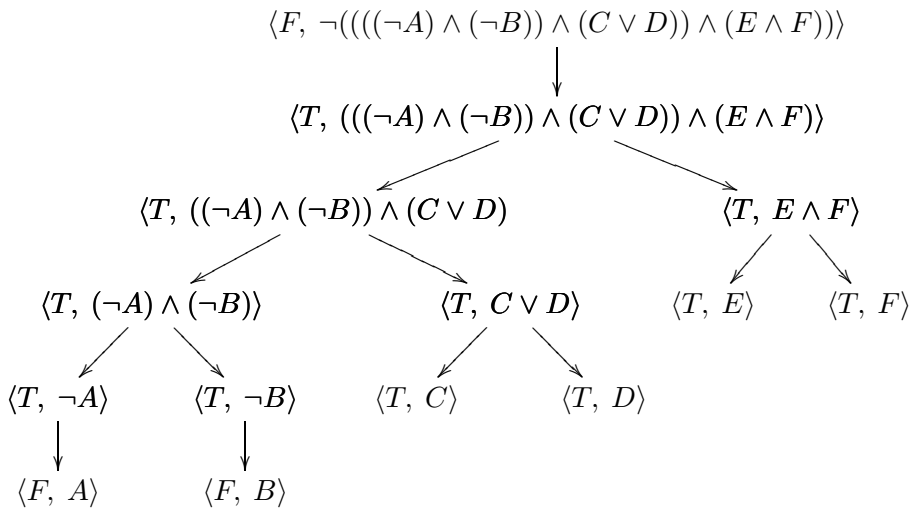


Figure 4.7: Formula tree for $\langle F,\ \neg((((\neg A) \wedge (\neg B)) \wedge (C \vee D)) \wedge (E \wedge F))\rangle$.

The only restriction on the choice of expansion orders is the partial order exhibited by the formula tree in Figure 4.7. A formula may be expanded only after its parent has been expanded. Thus, a formula tree may be viewed as a partial order of the expansions of

the formulas. After a container splits natural deduction, Kripke C-tableau, and sometimes in analytic tableau, the expansion order may differ between containers.

To eliminate different expansion orders in different containers, a rule called grouping is introduced. A *grouping* is the set of expansions that come from a single vertex in the formula tree. Under this rule each expansion step consists of expanding all formulas that belong to the same grouping. Some vertices in the formula tree, notably the root, are always in a grouping that has only one expansion.

By adding the grouping rule, the number of expansion steps required by each method is now the same. An expansion consists of expanding a formula on one sequent, box, branch, or in another container, but an expansion step consists of one or more expansions, one for each formula that shares the same vertex in the formula tree. Returning to the proofs in Figures 4.2 and 4.3 and comparing the number of expansions and expansion steps, using the maximum expansion rule. The number of expansions in natural deduction and Kripke C-tableau proofs is six, in analytic tableau five, and in the matrix method four. However, the number of expansion steps needed for natural deduction, analytic tableau and Kripke C-tableau is four. This change results from the combining of the duplicate expansions of $\neg\varphi \vee \psi$ and $\neg\varphi$. By its nature, the matrix method does not have duplicate expansions; thus, in the matrix, each grouping consists of the expansion of a single vertex. The number of groupings used by each method is the same since every proof attempts to prove the same formula, which has a fixed formula tree.

Since the grouping rule requires that all formulas in a grouping be expanded at the same time, the natural deduction and analytic tableau proofs in Figure 4.4 are not permitted; however, the natural deduction proof and analytic tableaux proofs in Figure 4.5 are permitted. Table 4.1 shows the expansion order and the groupings it creates during the construction of the natural deduction, analytic tableaux, and matrix proofs in Figures 4.5 and 4.6.

Step Function	**Matrix Vertices**	**Analytic Tableau Vertices**	**Sequents**
Initial Step	$\rightarrow 1$	$\rightarrow 1$	$\rightarrow A$
Expand $\langle F, \neg((((\neg A) \wedge (\neg B)) \wedge (C \vee D)) \wedge (E \wedge F))\rangle$	$1 \rightarrow 2$	$1 \rightarrow 2$	$A \rightarrow B$
Expand $\langle T, (((\neg A) \wedge (\neg B)) \wedge (C \vee D)) \wedge (E \wedge F)\rangle$	$2 \rightarrow 3, 4$	$2 \rightarrow 3, 4$	$B \rightarrow C$
Expand $\langle T, ((\neg A) \wedge (\neg B)) \wedge (C \vee D))\rangle$	$3 \rightarrow 5, 6$	$3 \rightarrow 5, 6$	$C \rightarrow D$
Expand $\langle T, C \vee D\rangle$	$6 \rightarrow 7, 8$	$6 \rightarrow 7, 8$	$D \rightarrow O, P$
Expand $\langle T, E \wedge F\rangle$	$4 \rightarrow 9, 10$	$4 \rightarrow 9, 10$ $4 \rightarrow 11, 12$	$O \rightarrow Q$ $P \rightarrow R$
Expand $\langle T, (\neg A) \wedge (\neg B)\rangle$	$5 \rightarrow 11, 12$	$5 \rightarrow 13, 14$ $5 \rightarrow 15, 16$	$Q \rightarrow S$ $R \rightarrow T$
Expand $\langle T, \neg A\rangle$	$11 \rightarrow 13$	$13 \rightarrow 17$ $15 \rightarrow 18$	$S \rightarrow U$ $T \rightarrow V$
Expand $\langle T, \neg B\rangle$	$12 \rightarrow 14$	$14 \rightarrow 19$ $16 \rightarrow 20$	$U \rightarrow W$ $V \rightarrow X$

Table 4.1: A table showing the correspondence between expansion steps in natural deduction, analytic tableau, and matrix proofs as shown in Figures 4.5 and 4.6. Each row in the table represents a single expansion step. The number/letter on the left of the arrow represents the vertex or sequent expanded to generate the number/letter of the vertex or sequent on the right of the arrow. Thus, a single expansion order can be used by each of these methods.

Chapter 5

Intuitionistic Logic

5.1 Some Definitions for Intuitionistic Logic

In [van Dalen 1994], intuitionistic logic is likened to the work of a human mathematician beginning with various postulates and objects and deriving theorems and objects from this initial state, then repeatedly using these previously proven theorems and the existence of these objects to prove new theorems and construct new objects. After a period of time, the mathematician stops.

The state of knowledge in an intuitionistic investigation is called a *world*. The investigation starts in an initial world. This world contains axioms and known objects and theorems that can be immediately derivable from these. As new evidence is gathered, the logical investigation may take different paths. This process is modeled by a DAG linking the current world with other possible worlds. This DAG linking worlds together is called a frame. A *frame* is the collection of worlds that are accessible from the initial world. As new evidence is gained, the investigation moves between worlds within the DAG. The successor relation $\geq$ indicates which worlds are accessible from other worlds. If w_i and w_j are worlds, then the statement $w_i \geq w_j$ indicates that w_i is a successor of w_j.

Instead of true and false, intuitionistic logic is defined using forced and unforced formulas in a frame consisting of possible worlds. Intuitionistic logic has the *monotonicity* property: once a theorem is proven or an individual is found to exist, then it exists in all possible future worlds (theorems and the existence of individuals are never retracted).

In a world a formula can be forced, unforced or neither forced or unforced. Forced formulas are known to be true, while unforced formulas are those formulas which are false or whose truth value is unknown.

The signs T and F, which denoted true and false in classical logic, are used to denote forced and unforced in intuitionistic logic. If the formula φ is forced, it is denoted by $\langle T, \varphi \rangle$ and if φ is unforced, it is denoted by $\langle F, \varphi \rangle$. This notation may seem confusing since formulas that are unforced have the F flag. However, this is standard notation as forced formulas have similar semantics to true formulas in classical logic.

The semantics of forced and unforced formulas are different from true and false in classical logic. They are described using Kripke's semantics. If a formula is forced in a world, it must also be forced in all successor worlds. A formula that is unforced in a world maybe forced in a successor world, but not the converse.

A formula φ is a theorem in intuitionistic logic if it is forced in all frames. Therefore, refutation creates an initial world where φ is unforced and attempts to construct a frame that is consistent with this initial world.

5.2 Philosophical Differences with Classical Logic

The four philosophical differences between classical and intuitionistic logic are:

- constructive dilemma
- an existential quantifier, which requires a witness

- a universal quantifier, which applies to current and future individuals
- the constructive semantics of the connective implication ($\Rightarrow$).

To demonstrate the first difference, the constructive dilemma, consider the proposition G, "Goldbach's conjecture can be proven." Goldbach conjecture states that every even integer greater than two is the sum of two prime numbers. Since G either has a proof or does not have a proof, the formula $G \vee \neg G$ can be constructed to represent this proposition. In classical logic, this is a theorem, but not in intuitionistic logic.

In classical logic this is accepted as a theorem as a result of the law of the excluded middle (*Tertium Non Datur* or there is no third value), which expresses the principle that all propositions are either true or false. Intuitionistic logic rejects this principle, recognizing that the truth value of a proposition may not be known but may become known in the future. In intuitionistic logic, the formula $G \vee \neg G$ means that there is either a proof of G or a proof of $\neg G$. Since G has not be proven nor has it be proven that G is false, $G \vee \neg G$ is not a theorem.

The second difference between classical and intuitionistic logic is the witness requirement for the existential quantifier ($\exists$). In intuitionistic logic, it cannot asserted $\exists x P(x)$ until a y can be found such that $P(y)$ is true. This prevents making assertions about empty domains (e.g. Pegasuses and unicorns).

A third difference between classical and intuitionistic logic deals with the universal quantifier ($\forall$). As evidence is gathered in the reasoning process, new individuals may be discovered. For $\forall x P(x)$ to be forced in world w, one must be able to say that for all x known in w, $P(x)$ is forced; in addition, for any y discovered in a possible successor world, $P(y)$ is also forced.

The fourth difference is in the semantics (meaning) of implication ($\Rightarrow$). In intuitionistic logic $P \Rightarrow Q$ means that there is an algorithm to construct Q from P. In classical

logic $P \Rightarrow Q$ is false only if P is true and Q is false For example, let P be "Lassie is a dog" and Q be "Lassie is a mammal." In classical logic, this statement is false only if Lassie is a dog but not a mammal. Symbolically, $(P \Rightarrow Q) \Rightarrow (\neg P \vee Q)$. Since all dogs are mammals, this statement is true. In intuitionistic logic, if it is known that Lassie is a dog, then it can be concluded that Lassie is a mammal $(P \Rightarrow Q)$. But if the truth value is not known for either proposition: "Lassie is not a dog" or "Lassie is a mammal" $(\neg P \vee Q)$, then the compound statement, "Lassie is not a dog or Lassie is a mammal," does not have a proof, because a proof that "Lassie is not a dog" ($\neg P$) or a proof that "Lassie is a mammal" (Q) is needed (the constructive dilemma comes into play). Hence, $(P \Rightarrow Q) \Rightarrow (\neg P \vee Q)$ is not a theorem in intuitionistic logic.

Statements like the ones below are not known to be true or false, in the fall of 2008:

- The complexity of primality testing is exponential.
- The Goldbach conjecture has a proof.
- $P = NP$.

At some point in the future, the truth value of some of these statements will become known. But for some statements, the truth value may never be known.

In science, experiments are performed in order to answer questions. For instance, if the police are investigating a murder where the victim was stabbed and a suspect has a bloody knife, the blood on the knife is analyzed to determine if it matches the victim's blood type. In classical logic, the assumption is made that the blood type found on the knife is known; intuitionistic logic allows for the test result to be initially unknown and perhaps never known as the knife could be washed before a sample is taken or the test results could be inconclusive. Classical logic assumes that all propositions are known to be true or false, while intuitionistic logic allows for reasoning with incomplete information

and the introduction of information that does not contradict previous information. Variables and formulas can be added to classical logic to allow it to deal with inconclusive or partial information, but intuitionistic logic was designed the work with these kinds of information.

5.3 Kripke Semantics

The semantics of the connectives are defined in terms of forced and unforced formulas in a set of worlds that make up a frame. A frame $\mathcal{F}$ is a three tuple: $\mathcal{F} = (P, \geq, \Phi)$, where P is a set of worlds, $\geq$ is a reflexive and transitive successor relation on the set P, and Φ is a partial function from a world and a formula to the set $\{T, F\}$, indicating if a formula is forced or unforced in a world.

If the formula φ is forced in world p, then $\Phi(p, \varphi) = T$, denoted as $p \Vdash \varphi$, and read as p forces φ. Similarly, if a formula ψ is unforced in world p, then $\Phi(p, \psi) = F$, denoted as $p \nVdash \psi$, and read as p does not force ψ.

The successor relation $\geq$ defines a DAG of worlds within a frame. For example, $q \geq p$ indicates that q is a possible future world reachable from p.

The deductive definitions of the primary connectives shown below are adapted from [Nerode and Shore 1993][page 266]:

- If $p \Vdash \varphi$, then for all $q \geq p$, $q \Vdash \varphi$ (monotonicity property).
- If $p \Vdash \varphi \wedge \psi$, then $p \Vdash \varphi$ and $p \Vdash \psi$.
- If $p \nVdash \varphi \wedge \psi$, then $p \nVdash \varphi$ or $p \nVdash \psi$.
- If $p \Vdash \varphi \vee \psi$, then $p \Vdash \varphi$ or $p \Vdash \psi$.
- If $p \nVdash \varphi \vee \psi$, then $p \nVdash \varphi$ and $p \nVdash \psi$.
- If $p \Vdash \varphi \Rightarrow \psi$, then for all $q \geq p$, $q \Vdash \varphi$ implies $q \Vdash \psi$.

- If $p \Vdash\!\!\!\!/\ (\varphi \Rightarrow \psi)$, then there exists a $q \geq p$ where $q \Vdash\!\!\!\!/\ \varphi$ and $q \Vdash \psi$.
- If $p \Vdash \neg\varphi$, then for all $q \geq p,\ q \Vdash\!\!\!\!/\ \varphi$.
- If $p \Vdash\!\!\!\!/\ \neg\varphi$, then for some $q \geq p,\ q \Vdash \varphi$.

When working in intuitionistic logic, conjunction ($\wedge$) and disjunction ($\vee$) behave very much like their classical logic counterparts, while implication ($\Rightarrow$) and negation ($\neg$) behave differently. The semantics described above affect not only the current world but also a future world or all future worlds. Unforced implication and negation require that there exists a future world with a certain property.

In a given world, formulas that are true are said to be *forced* in that world. If a formula is forced in one world w, then it is forced in all possible future worlds. If a formula is both **unforced** and **unforced** in the same world w, then there is a contradiction. However, it is not a contradiction if a formula is unforced in one world but forced in a successor world.

5.4 Refutation in Intuitionistic Logic

Refutation in classical logic depends on the tautology $\neg\neg\varphi \Leftrightarrow \varphi$, but this does not hold in intuitionistic logic. In intuitionistic logic, a formula is a theorem if it is forced in all frames. In intuitionistic logic, a refutation proof creates a frame with an initial world assuming that the proposed theorem is unforced and then builds the rest of the frame consistent with this assumption. The proof succeeds if and only if a frame cannot be constructed consistent with this assumption.

Signed Formula E	**Expansion Rule Type $SM_r(E)$**	**Child Formulas $SM_c(E)$**	**Intuitionistic Semantics**	**$J(E)$ Function**
$\langle T, \varphi \wedge \psi \rangle$	α	$\{\langle T, \varphi \rangle, \langle T, \psi \rangle\}$	current world	present
$\langle F, \varphi \wedge \psi \rangle$	β	$\{\langle F, \varphi \rangle, \langle F, \psi \rangle\}$	current world	present
$\langle T, \varphi \vee \psi \rangle$	β	$\{\langle T, \varphi \rangle, \langle T, \psi \rangle\}$	current world	present
$\langle F, \varphi \vee \psi \rangle$	α	$\{\langle F, \varphi \rangle, \langle F, \psi \rangle\}$	current world	present
$\langle T, \varphi \Rightarrow \psi \rangle$	β	$\{\langle F, \varphi \rangle, \langle T, \psi \rangle\}$	current and all future worlds	all_future
$\langle F, \varphi \Rightarrow \psi \rangle$	α	$\{\langle T, \varphi \rangle, \langle F, \psi \rangle\}$	at least one future world	some_future
$\langle T, \neg\varphi \rangle$	α	$\{\langle F, \varphi \rangle\}$	current and all future worlds	all_future
$\langle F, \neg\varphi \rangle$	α	$\{\langle T, \varphi \rangle\}$	at least one future world	some_future

Table 5.1: J Function definition. The J function is used in addition to SM_r and SM_c to describe the semantics of intuitionistic logic.

5.5 The J Function

In the description of the intuitionistic algorithm, the function J is introduced. This function modifies the actions taken when expanding some formulas. J maps a signed formula E to one of three values: $present$, all_future, and $some\_future$.

- If $J(E) = present$, then the expansion of the formula is carried out in the current world, much like in classical logic.
- If $J(E) = all\_future$, then the expansion takes place in the current world as well as in all possible future worlds.
- If $J(E) = some\_future$, then the expansion only applies to at least one future world.

In intuitionistic logic, all of the formulas E where $J(E) = all\_future$ have a T sign. Since the monotonicity property of the logic and the methods moves all formulas with a T sign to future worlds, these formulas can be handled as if $J(E) = current$. If another logic was

implemented that required some formulas with a F sign to be advanced to future worlds, then this function would have to be modified.

Definition 16 *The* N set *consists of the unforced formulas that have either implication or negation as their primary connective. A formula E is in this set if and only if $J(E) = some\_future$.*

The formulas in the N set require special handling in each of the methods. If a formula is in the N set, then the Kripke semantics require a future world to meet some forcing condition.

5.6 Intuitionistic Methods

The intuitionistic proof methods presented in this section are modifications of their classical logic counterparts. Each method is modified to reflect the intuitionistic semantics of the connectives. Analytic tableau and the matrix method build their graphs and then analyze them to see if each branch or path can be closed simultaneously. Natural deduction and Kripke tableau purge the the unforced formulas when expanding an N set formula. During a proof there is often more than one formula in the N set. After expanding a formula in the N set, future expansions may create another N set, and another choice has to be made. If one series of choices fails to produce a proof, another set of choices is tried until either one sequence of choices succeeds or all possible sequences have failed. If sequence of choices produces a successful proof, then the theorem has been proven, but if no sequence can be found, then the proof attempt has failed.

$$\Gamma, \varphi \longrightarrow \varphi, \Delta \quad (Axiom)$$

$$\Gamma, \bot \longrightarrow \Delta \quad (Axiom) \qquad \Gamma \longrightarrow \top, \Delta \quad (Axiom)$$

$$\frac{\Gamma, \varphi, \psi \longrightarrow \Delta}{\Gamma, \varphi \wedge \psi \longrightarrow \Delta} l\wedge \qquad \frac{\Gamma \longrightarrow \varphi, \Delta \quad \Gamma \longrightarrow \psi, \Delta}{\Gamma \longrightarrow \varphi \wedge \psi, \Delta} r\wedge$$

$$\frac{\Gamma, \varphi \longrightarrow \Delta \quad \Gamma, \psi \longrightarrow \Delta}{\Gamma, \varphi \vee \psi \longrightarrow \Delta} l\vee \qquad \frac{\Gamma \longrightarrow \varphi, \psi, \Delta}{\Gamma \longrightarrow \varphi \vee \psi, \Delta} r\vee$$

$$\frac{\Gamma \longrightarrow \varphi, \Delta \quad \Gamma, \psi \longrightarrow \Delta}{\Gamma, \varphi \Rightarrow \psi \longrightarrow \Delta} l\Rightarrow \qquad \frac{\Gamma, \varphi \longrightarrow \psi}{\Gamma \longrightarrow \varphi \Rightarrow \psi, \Delta} r\Rightarrow$$

$$\frac{\Gamma \longrightarrow \varphi, \Delta}{\Gamma, \neg\varphi \longrightarrow \Delta} l\neg \qquad \frac{\Gamma, \varphi \longrightarrow}{\Gamma \longrightarrow \neg\varphi, \Delta} r\neg$$

Figure 5.1: Intuitionistic sequent rules. This figure contains the natural deduction rewrite rules and axioms for intuitionistic logic. Γ and Δ represent the sets of formulas that are not involved in the current rule. The $r \Rightarrow$ and $r\neg$ rules differ from their counterparts in classical logic. This set of sequent rules is a modification of the rules in [Wallen 1990].

5.6.1 Natural Deduction

As in classical logic, natural deduction attempts to construct a tree of sequents rooted with an endsequent, having axioms at its leaves. The sequent rules for intuitionistic logic, shown in Figure 5.1, are similar to the ones used in classical logic. They have the same axiom rules, but two of the rewrite rules are different. The two rules that differ are $r \Rightarrow$ and $r\neg$; these rewrite rules expand a formula in the N set. They are both missing Δ from the premise, indicating that formulas in the succedent are purged except for the formulas generated by the expansion.

The order in which formulas are expanded can determine whether or not a proof is found. Unlike classical logic where the expansion order does not affect the outcome of the proof, the search for an expansion order that leads to a successful proof is important

in intuitionistic logic. A formula is a theorem if there is at least one expansion order that creates a tree that has axiom sequents at its leaves.

$$\dfrac{\dfrac{\dfrac{\dfrac{A \longrightarrow}{\longrightarrow \neg A, A}\, r\neg}{\neg A \longrightarrow \neg A}\, l\neg}{\longrightarrow \neg A \Rightarrow \neg A}\, r\Rightarrow} \qquad \dfrac{\dfrac{\dfrac{\dfrac{A \longrightarrow A}{A, \neg A \longrightarrow}\, l\neg}{\neg A \longrightarrow \neg A}\, r\neg}{\longrightarrow \neg A \Rightarrow \neg A}\, r\Rightarrow}$$

Figure 5.2: Two intuitionistic natural deduction proof attempts of $\neg A \Rightarrow \neg A$. The proof on the left applies the $l\neg$ rule before the $r\neg$ rule and fails, while the proof on the right applies these rules in the opposite order and succeeds.

Two proof attempts for the theorem $\neg A \Rightarrow \neg A$ are shown in Figure 5.2; because they use different expansion orders, the one on the left fails, while the one on the right succeeds. After applying the $r \Rightarrow$ rule, there is a choice of either applying a $l\neg$ or a $r\neg$ rule; the left proof applies the $l\neg$ rule, while the right proof applies the $r\neg$ rule. At this point, the left proof has the sequent $\longrightarrow \neg A, A$ as its leaf. The $r\neg$ rule is the only one that can be applied to this sequent, creating the sequent $A \longrightarrow$, which is not an axiom, and since no rewrite rules can be applied, the proof has failed. Returning to the proof tree on the right after the $r\neg$ is applied, it has the leaf sequent $A, \neg A \longrightarrow$. The only rewrite rule that can be applied is $l\neg$, adding the sequent $A \longrightarrow A$; since this is an axiom, the proof is successful. Since the $r\neg$ rule purges the contents of the succedent, it is a good heuristic to apply it when there are as few formulas as possible in the succedent.

In classical logic the formula $(A \Rightarrow B) \Rightarrow (\neg A \vee B)$ is a theorem, but it is not a theorem in intuitionistic logic. Figure 5.3 shows two different unsuccessful proof attempts for this formula. Regardless of the expansion order used and, hence, the sequence of choices made to expand formulas in the N set, this formula cannot be proven.

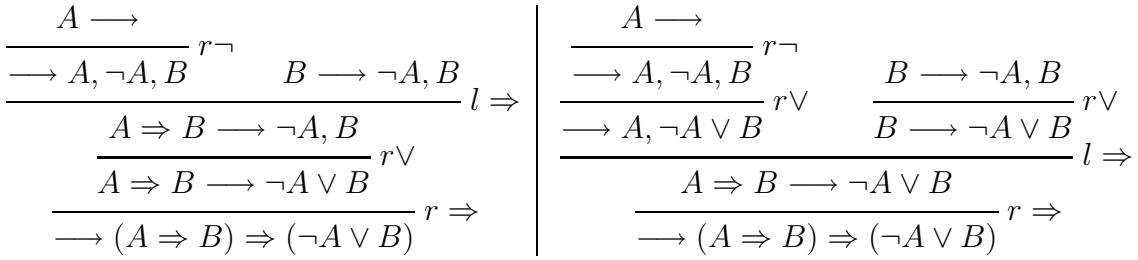


Figure 5.3: Two intuitionistic natural deduction proof attempts for the formula $(A \Rightarrow B) \Rightarrow (\neg A \vee B)$. The two proof attempts in this figure use different expansion orders to prove the same formula. The proof on the left applies the $r\vee$ rule before the $l \Rightarrow$ rule, while the proof on the right applies these rules in the opposite order. Both expansion orders lead to failed proofs for the same formula.

5.6.2 Kripke Tableau

The Kripke tableau method was designed for intuitionistic logic. A simplified version for classical logic, called the Kripke C-tableau, was described in Section 3.2. The rules for both the Kripke tableau and the C-tableau were derived from the sequent rules used for their respective logics.

The functions KR_r and KR_c defined in Table 5.2 are the same as in classical logic, but intuitionistic logic also includes the J function. This J function modifies the actions taken when expanding a formula E in the N set. When $J(E) = some\_future$, the intuitionistic method creates a "new" box from the current one by copying only the formula(s) in the left column, while the formulas in right column are lost, and then adding the formula(s) generated by the expansion.

Since the Kripke tableau is just another representation of natural deduction, the special action taken when expanding an implication or negation in the right column corresponds to the two sequent rewrite rules $r\neg$ and $r \Rightarrow$. In these two sequent rules, the formulas in the succedent of the conclusion are lost. This loss is represented by the absence of Δ from the premise.

Kripke Column Formula E	**Expansion Rule Type $KR_r(E)$**	**Child Formulas $KR_c(E)$**	**Intuitionistic Semantics**	**$J(E)$ Function**
$\langle left, \varphi \wedge \psi \rangle$	$no\ split$	$\{\langle left, \varphi \rangle, \langle left, \psi \rangle\}$	current world	present
$\langle right, \varphi \wedge \psi \rangle$	$split$	$\{\langle right, \varphi \rangle, \langle right, \psi \rangle\}$	current world	present
$\langle left, \varphi \vee \psi \rangle$	$split$	$\{\langle left, \varphi \rangle, \langle left, \psi \rangle\}$	current world	present
$\langle right, \varphi \vee \psi \rangle$	$no\ split$	$\{\langle right, \varphi \rangle, \langle right, \psi \rangle\}$	current world	present
$\langle left, \varphi \Rightarrow \psi \rangle$	$split$	$\{\langle right, \varphi \rangle, \} \langle left, \psi \rangle\}$	current and all future worlds	all_future
$\langle right, \varphi \Rightarrow \psi \rangle$	$no\ split$	$\{\langle left, \varphi \rangle, \langle right, \psi \rangle\}$	at least one future world	some_future
$\langle left, \neg\varphi \rangle$	$no\ split$	$\{\langle right, \varphi \rangle\}$	current and all future worlds	all_future
$\langle right, \neg\varphi \rangle$	$no\ split$	$\{\langle left, \varphi \rangle\}$	at least one future world	some_future

Table 5.2: Kripke and J Functions. The J function is used with the KR_r and KR_c functions to describe the actions taken to expand a formula in a Kripke tableau.

Figure 5.4 shows an unsuccessful Kripke proof attempt for the formula $(A \Rightarrow B) \Rightarrow (\neg A \vee B)$. As in natural deduction, finding a proof becomes a search for an expansion order that causes all of the leaf boxes to close.

5.6.3 Analytic Tableau

This subsection discusses how Nerode and Shore [Nerode and Shore 1993] modified the analytic tableau method to prove theorems in intuitionistic logic. As in classical logic, each tableau expansion uses one of Smullyan's rules to expand a vertex labeled with a formula. The formulas generated by each expansion are added to the tableau. In addition, the intuitionistic method constructs a partial order. Each vertex in an intuitionistic tableau is labeled with both a signed formula and an element from the partial order.

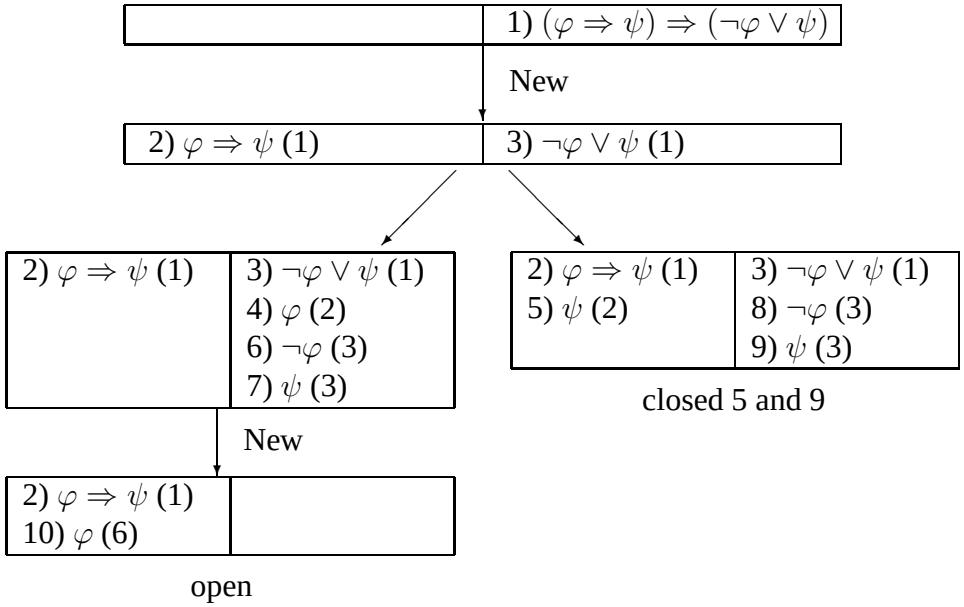


Figure 5.4: Intuitionistic Kripke tableau proof attempt for $(\varphi \Rightarrow \psi) \Rightarrow (\neg\varphi \vee \psi)$. This is a theorem in classical logic, but it is not a theorem in intuitionistic logic.

The first step in constructing a proof for the formula φ consists of creating a partial order with one element p_0 and creating the root vertex of the tableau. This vertex is labeled with the partial order element p_0 and the signed formula $\langle F, \varphi \rangle$. Each subsequent step expands a vertex labeled with the signed formula E and a partial order element p. If $J(E) = some\_future$, then a new element p' of the partial is created as the immediate successor of p and the new vertices are labeled with p'. If $J(E) \neq some\_future$, no new element is added to the partial order, and the vertices added to the tableau are labeled with p.

Recall the monotonicity rule states that if a formula is forced in one world, it is forced in all successor worlds. This rule may add vertices to a tableau without expanding a vertex. Suppose that a vertex v is labeled with the forced formula E and the partial order element p. This rule can be applied if there is a partial order element q that is a successor of p. The application of the rule adds a vertex as the child of a vertex in $L(v)$, this new vertex is labeled with E and q.

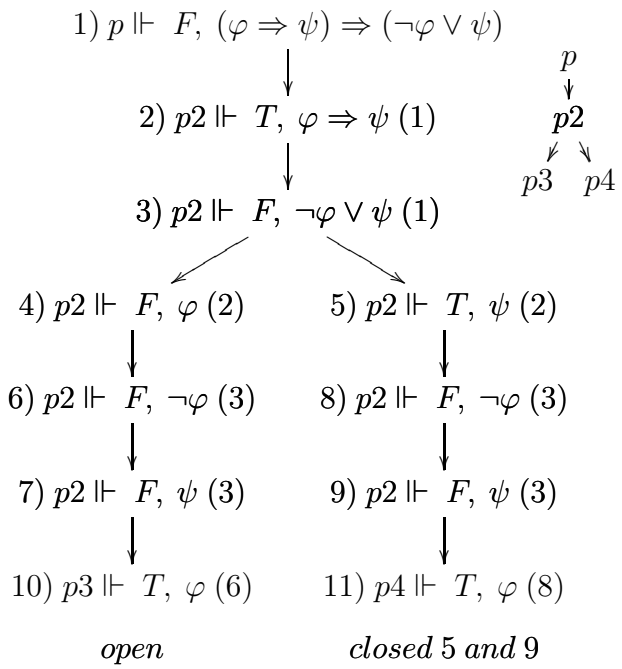


Figure 5.5: Intuitionistic analytic tableau proof attempt of $(\varphi \Rightarrow \psi) \Rightarrow (\neg\varphi \vee \psi)$. Each vertex is labeled with both a signed formula and an element of the partial order. The right branch closes because vertices 5 and 9 contain contradictory formulas in world $p2$. In the left column, while vertices 4 and 10 are labeled with contradictory formulas, they are in different worlds. Further, since $\langle T, \varphi\rangle$ is world $p3$, and $\langle F, \varphi\rangle$ is world $p2$, the monotonicity rule cannot be applied because $p2$ is not a successor of $p3$. Nor can $\langle F, \varphi\rangle$ be moved from $p2$ to $p3$ since it is unforced.

As in classical logic, a branch is closed when it contains vertices labeled with contradictory formulas. In intuitionistic logic, a branch closes if it contains contains a contradiction, either a vertex labeled with a self-contradictory formula or a pair of contradictory formulas. A vertex is self-contradictory if it is labeled with $\langle T, \bot\rangle$ or $\langle F, \top\rangle$. A pair of vertices are contradictory if they both are labeled with the same partial order element, but one is labeled with $\langle F, \varphi\rangle$ and the other labeled with $\langle T, \varphi\rangle$ for some formula φ.

5.6.4 Matrices in Intuitionistic Logic

The construction rules used to build a matrix for intuitionistic logic are the same as those for classical logic; however, each non-structural vertex in the matrix is labeled with both a signed formula and a T-string (described below). As in classical logic, after the matrix is constructed, there is a search for a pair of contradictory formulas on each path. In intuitionistic logic, there is an additional requirement that the T-strings labeling the vertices which have contradictory formulas can be unified with a single substitution.

A good description of T-strings can be found in [Otten and Kreitz 1996]; they serve a similar role as partial orders do in an intuitionistic tableau. In the theorem prover ileanTAP [Otten 1997], T-strings are used in an analytic tableau instead of the partial order used in the previous subsection.

A T-string is composed of constants and variables. An over-bar will be used to mark variables in a T-string. Two T-strings are unified by substituting a string of constants and variables for each variable. For instance, $ca\overline{r}d$ and $candid$ may be unified by substituting ndi for $\overline{r}$. However, $mi\overline{n}or$ and $ma\overline{t}rix$ cannot be unified since one string starts with the constants mi and the other with the constants ma; the variables appear after these initial characters.

Definition 17 *Two strings t and s over the same alphabet have the* T-string *property if 1) there are no repeated symbols within t or s, 2) there is a string r with $0 \leq |r| \leq min(|t|, |s|)$ that is a prefix of both t and s, and 3) once past the common prefix r, they share no symbols in common.*

The T-strings used in intuitionistic matrices are constructed using the formula tree [Wallen 1990] discussed in Section 2.11. Recall for the proposed theorem φ, the root vertex of the formula tree is labeled with the signed formula $\langle F, \varphi \rangle$. A vertex v labeled with the formula E has no children if E is atomic. If $|SM_c(E)| = 1$, then v has one child labeled

with the formula in $SM_c(E)$. If $|SM_c(E)| = 2$, then v has two children, each labeled with a formula in $SM_c(E)$.

For convenience each vertex in the formula tree will be denoted with an identifier a_i, where i is a unique number within the tree; these numbers are assigned using depth-first ordering. A vertex is *special* if it is the root or a leaf, or is labeled with a signed formula that has negation or implication as its primary connective. The T-string for vertex v in the formula tree is constructed by concatenating the symbols associated with each special vertex as one travels from the root to v. Let E be the signed formula labeling the special vertex i in a formula tree. If E is forced, then the symbol for vertex i is the variable $\overline{a_i}$, while if E is unforced, then the symbol for vertex i is the constant a_i. Suppose vertex v has a child w and v is labeled with the T-string s. If w is not special, then it will also be labeled with s. But if w is special, its T-string consists of s followed by the T-string symbol for the vertex w.

A pair of vertices on path p is contradictory if for some formula φ one vertex is labeled with the formula $\langle T, \varphi \rangle$ and the other vertex is labeled with the formula $\langle F, \varphi \rangle$. Equation 5.1 defines R_p as the set of pairs of contradictory vertex pairs on path p. Equation 5.2 defines S_p as the set of pairs of T-strings where each T-string labels one vertex of a contradictory pair of vertices on path p. If for each p there is a T-string pair $(t_1, t_2) \in S_p$ such that $\sigma(t_1) = \sigma(t_2)$, then the proof is successful.

$$R_p = \{(v_1, v_2) | v_1 \text{ and } v_2 \text{ are contradictory vertices on path } p\} \tag{5.1}$$

$$S_p = \{(s_1, s_2) | (v_1, v_2) \in R_p \text{where } s_1 \text{ labels } v_1 \text{ and } s_2 \text{ labels } v_2\} \tag{5.2}$$

Let σ be a substitution mapping each T-string variable to a possibly empty string of T-string constants. For an intuitionistic proof to be successful, each path p must contain

$$input \longrightarrow 1)\ F,\ \varphi \vee \neg\varphi \longrightarrow output$$

$$input \longrightarrow 2)\ F,\ \varphi\ (1) \rightarrow 3)\ F,\ \neg\varphi\ (1) \rightarrow output$$

$$input \longrightarrow 2)\ F,\ \varphi\ (1) \rightarrow 4)\ T,\ \varphi\ (3) \longrightarrow output$$

Figure 5.6: Matrix proof of the excluded middle $\varphi \vee \neg\varphi$. This is a theorem in classical logic but not a theorem in intuitionistic logic. The T-strings labeling vertices 2 and 4 cannot be unified.

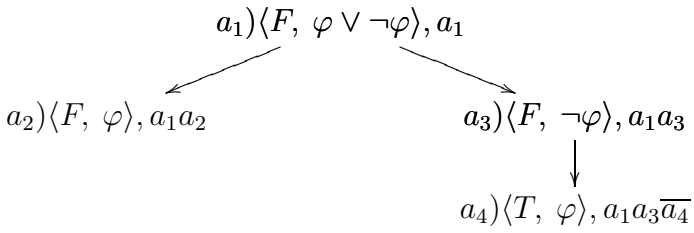


Figure 5.7: The formula tree with T-strings for the proposed theorem $(\varphi \vee \neg\varphi)$. The root vertex is labeled with the signed formula composed of the F sign and the proposed theorem. Each vertex in the formula tree is labeled with one of the signed formulas created by expanding the signed formula labeling its parent and the T-string derived from its parent.

either 1) a vertex with a self-contradictory formula ($\langle T, \bot\rangle$ or $\langle F, \top\rangle$) or 2) a pair of T-strings in S_p that can be unified by a substitution σ; i.e., for each path p there is a pair of T-strings $(s_1, s_2) \in S_p$ such that $\sigma(s_1) = \sigma(s_2)$.

The first example of an intuitionistic matrix proof is for the formula known as the excluded middle, $(\varphi \vee \neg\varphi)$, and is displayed in Figure 5.6 with its formula tree being shown in Figure 5.7. In classical logic, its only path would be closed because it contains vertices labeled with the contradictory formulas $\langle T, \varphi\rangle$ and $\langle F, \varphi\rangle$. But in intuitionistic logic, the two T-strings, $a_1a_3\overline{a_4}$ and a_1a_2, also have to be unified, but this is not possible since the constants a_2 and a_3 cannot be unified, and, thus, the proof fails.

In the second example, an intuitionistic matrix proof for $(\varphi \Rightarrow \psi) \Rightarrow (\neg\varphi \vee \psi)$ is attempted. Figure 5.8 shows the matrix proof and Figure 5.9 its formula tree. There

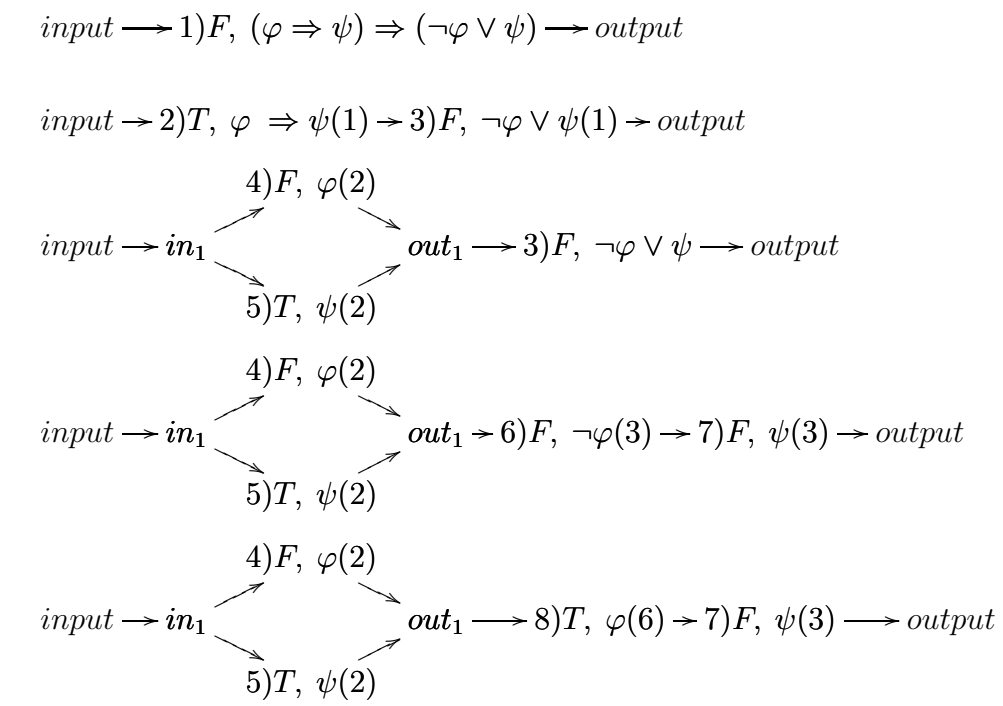


Figure 5.8: Intuitionistic matrix proof of $(\varphi \Rightarrow \psi) \Rightarrow (\neg\varphi \vee \psi)$. The proof proceeds from the graph at the top to the one at the bottom. The proof attempt fails; in the final matrix each path has a pair of contradictory formulas, but there is not a single substitution that unifies both pairs of T-strings.

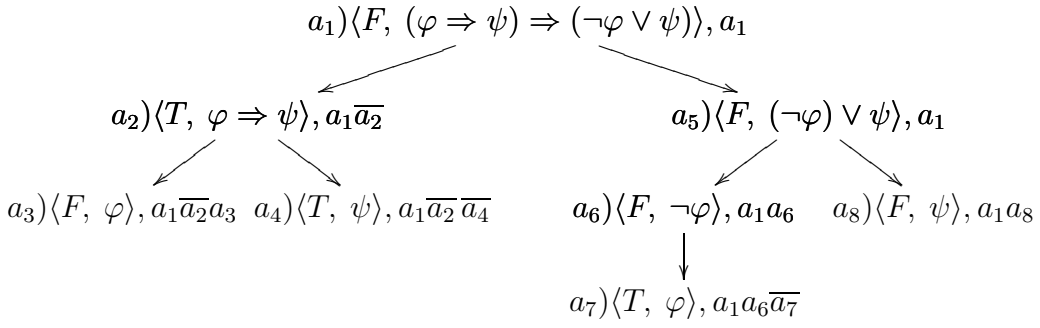


Figure 5.9: Formula tree labeled T-strings for $(\varphi \Rightarrow \psi) \Rightarrow (\neg\varphi \vee \psi)$. The root vertex is labeled with the signed formula composed of the F sign and the proposed theorem. Each vertex in the formula tree is labeled with a formula and a T-string.

are two paths through the matrix; on the upper path there is only one set of contradictory vertices, 4 and 8; Vertex 4 has the T-string $a_1\overline{a_2}a_3$, and Vertex 8 has the T-string $a_1a_6\overline{a_7}$.

On the bottom path there is only one set of contradictory vertices, 5 and 7; Vertex 5 has the T-string $a_1\overline{a_2}\,\overline{a_4}$, and Vertex 7 has the T-string a_1a_8. To unify the top pair, $\sigma(\overline{a_2})$ must start with a_6, but to unify the bottom pair, $\sigma(\overline{a_2}) = a_8$ or $\sigma(\overline{a_2}) = \epsilon$, where ϵ is the empty string. or the empty string. These two requirements of $\sigma(\overline{a_2})$ cannot be simultaneously satisfied. The proof fails since there is no single substitution that unifies both pairs of T-strings.

Recall the monotonicity property states the if a formula is forced in one world, then it is forced in all successor worlds. This property is implemented for a formula E by taking two actions, 1) in the matrix inserting a copy of E as its immediate successor and 2) in the formula tree adding a E as its own sibling, so that the complete formula tree structure of E appears twice. The different copies of E's tree with each have their own identifiers within the tree, and, thus, different T-string constants and variables. Recall each vertex in the formula tree has an identifier.

It may be necessary to create more than two copies of an forced formula. Since using the monotonicity rule creates extra copies of a formula, it should only be done when a proof has failed because of a T-string variable needed to have more than one value. However, care must be taken to find the forced formula farthest from the root suitable for the circumstance to limit to extra vertices added to the matrix and the formula tree. Finally, the extra paths added to the matrix by using the monotonicity rule may add extra paths that cannot be closed.

5.7 Intuitionistic Container

The intuitionistic container is a modification of its classical counterpart. It identifies the commonalities of the intuitionistic methods that will be used in a generalized algorithm for intuitionistic logic. This section begins by introducing the J function and N set formulas. Next the J function is used in specifying the properties of the intuitionistic

container, and then proving that each of the intuitionistic methods uses a data structure and a S function that has the properties of the intuitionistic container.

Classical logic was described using the functions SM_c and SM_r to expand formulas in a container. To describe intuitionistic logic, a third function J, defined in Figure 5.1, is added that modifies the actions taken when expanding an N set formula.

In addition to the set of β child containers created by the application of a β-rule, there is also a set of child containers created when a formula in the N set is expanded. Expanding a formula E in the N set creates a container having only the forced formulas of its parent and the formulas in $SM_c(E)$. Unlike child containers created by a β-rule, if one of the containers created by an N set expansion closes, then its parent also closes. A container is contradictory if it contains a contradictory formula or formula(s), i.e. $\langle T, \bot \rangle$, $\langle F, \top \rangle$ or for a formula φ, both $\langle T, \varphi \rangle$ and $\langle F, \varphi \rangle$. A container closes if one of three conditions are met: 1) it is contradictory (it contains a contradiction), 2) all of its β child containers close, or 3) one of its N child containers close.

An *intuitionistic logical container* $\langle C, S \rangle$ consists is a data structure C that stores logical formulas and a function S that maps the data structure to a set of signed formulas, with the following six properties:

LJ1– Two Sets The logical formulas stored in a container C can be separated into two sets T and F denoted as $T(C)$ and $F(C)$.

LJ2– S Function The S function maps the formulas in C to a set of signed formulas where the formulas with a T sign come from the $T(C)$ set and the formulas with a F sign come from the $F(C)$ set.

LJ3– Initial Container When the initial container is created, it contains only the proposed theorem in its F set.

LJ4– Expansion Rules Let C be a container and E be a signed formula in $S(C)$.

LJ4a– α **and** $\boldsymbol{J(E) \neq some\_future}$ If $SM_r(E) = \alpha$ and $J(E) \neq some\_future$, then the expansion rule adds formula(s) to C. Thus, $S(C') = S(C) \cup SM_c(E)$ where C' is the container after the expansion.

LJ4b– α **and** $\boldsymbol{J(E) = some\_future}$ If $SM_r(E) = \alpha$ and $J(E) = some\_future$, then there is an expansion rule that creates an N child container C'. This container has the forced formulas in C and the formulas in $SM_c(E)$; i.e. $S(C') = T(C) \cup SM_c(E)$. Note that the set of unforced formulas, $F(C)$, is not copied to C' in the creation of the N child container.

LJ4c– β **expansion** If $SM_r(E) = \beta$, then there is an expansion rule that splits C, replacing it with two containers C_1 and C_2. Let E_1 and E_2 be the signed formulas in $SM_c(E)$. The contents of these new containers are $S(C_1) = S(C) \cup \{E_1\}$ and $S(C_2) = S(C) \cup \{E_2\}$.

LJ4d– Only α **and** β **expansion rules** All rules which change the contents of a container or create new containers must fit into one of the three categories above or can be decomposed into steps, each of which belongs to one of the above categories.

Note: In this system whenever $J(E) = some\_future$, E is an α-formula. Thus, there is no formula E where $J(E) = some\_future$ and $SM_r(E) = \beta$.

LJ5– Closure A container closes if 1) it contains a contradiction, 2) all of the β child containers created by a split rule close, or 3) at least of one of its N child containers created by expanding a formula in the N set closes.

LJ6– Success A proof method succeeds if and only if the application of the expansion and closure rules causes its initial container to close.

5.7.1 A Sequent in Natural Deduction as an Intuitionistic Container

Lemma 18 *If E is an SSF formula in a sequent K, $SM_r(SF_n(E)) = \alpha$, and $J(SF_n(E)) \neq some\_future$, then the premise P generated by expanding E can be computed using the equation $S2C(P) = S2C(K) \cup SM_c(E)$.*

Proof:

The sequent rules in which $SM_r(SF_n(E)) = \alpha$ and $J(SF_n(E)) \neq some\_future$ are $l\wedge$, $r\vee$, and $l\neg$. These rules are the same for both classical and intuitionistic logic and thus, have already been proven as part of lemmas 9 and 10.

Thus, the intuitionistic sequent meets the α and $J(E) \neq some\_future$ expansion property (LJ4a). □

Lemma 19 *If E is an SSF formula in the sequent K and $J(SF_n(E)) = some\_future$, then the premise P generated expanding E meets the equality $T(S2C(K)) \cup SM_c(SF_n(E)) = S2C(P)$.*

Proof:

If E is an SSF formula and $J(E) = some\_future$, then the form of E is either $\langle succ, \varphi \Rightarrow \psi \rangle$ or $\langle succ, \neg\varphi \rangle$; the rules used to expand these formulas are $r\Rightarrow$ and $r\neg$, respectively. In both cases $SM_r(SF_n(E)) = \alpha$.

In both cases below in the conclusion of the sequent rule, Γ represents the set of SSF formulas in the antecedent and Δ represents the set of SSF formulas in the succedent. Let $G = S2C(\Gamma)$, the set of signed formulas corresponding to the formulas in the antecedent, and $D = S2C(\Delta)$, the set of signed formulas corresponding to the formulas in the succedent.

The two cases below consider the two types of formulas, where $J(SF_n(E)) = some\_future$.

Case E is of the form $\langle succ, \varphi \Rightarrow \psi \rangle$:

The sequent rule for E is $r \Rightarrow \quad \dfrac{\Gamma, \varphi \longrightarrow \psi}{\Gamma \longrightarrow \varphi \Rightarrow \psi, \Delta}\, r \Rightarrow$.

The premise sequent P is computed below:

$$T(S2C(K)) \cup SM_c(SF_n(E))$$

$$= T(S2C(\Gamma \cup \Delta \cup \{\langle succ, \varphi \Rightarrow \psi \rangle\})) \cup SM_c(SF_n(\langle succ, \varphi \Rightarrow \psi \rangle))$$

$$= T(G \cup D \cup \{\langle F, \varphi \Rightarrow \psi \rangle\}) \cup SM_c(\langle F, \varphi \Rightarrow \psi \rangle)$$

$$= G \cup \{\langle T, \varphi \rangle, \langle F, \psi \rangle\} = S2C(\Gamma, \varphi \longrightarrow \psi) = S2C(P)$$

The sequent computed by this equation matches the premise of the $r \Rightarrow$ rewrite rule.

Case E is of the form $\langle succ, \neg\varphi \rangle$:

The sequent rule for E is $r\neg \quad \dfrac{\Gamma, \varphi \longrightarrow}{\Gamma \longrightarrow \neg\varphi, \Delta}\, r\neg$.

The premise sequent P is computed below:

$$T(S2C(K)) \cup SM_c(SF_n(E))$$

$$= T(S2C(\Gamma \cup \Delta \cup \{\langle succ, \neg\varphi \rangle\})) \cup SM_c(SF_n(\langle succ, \neg\varphi \rangle))$$

$$= T(G \cup D \cup \{\langle F, \neg\varphi \rangle\}) \cup SM_c(\langle F, \neg\varphi \rangle)$$

$$= G \cup \{\langle T, \varphi \rangle\} = S2C(\Gamma, \varphi \longrightarrow) = S2C(P)$$

The sequent computed by this equation matches the premise of the $r\neg$ rewrite rule.

The intuitionistic sequent rewrite rules meet the α and $J(E) = some\_future$ expansion property (LJ4b). □

Lemma 20 *If E is an SSF formula in the sequent K and $SM_r(SF_n(E)) = \beta$, then the two premises P_1 and P_2 can be computed using the equations $S2C(P_1) = S2C(K) \cup \{E_1\}$ and $S2C(P_2) = S2C(K) \cup \{E_2\}$ where $\{E_1, E_2\} = SM_c(E)$.*

Proof:

The sequent rules in which $SM_r(SF_n(E)) = \beta$ are $r\wedge$, $l\vee$, and $l \Rightarrow$. These rules are the same for both classical and intuitionistic logic Thus, they have already been proven as

part of lemmas 9 and 10. Hence, the intuitionistic sequent meets the β expansion property (LJ4c). $\square$

Lemma 21 *A sequent in intuitionistic natural deduction with $S2C$ as its S function is an intuitionistic container.*

Proof:

The sequent is composed of two sets, with the antecedent being the T set and the succedent being the F set, fulfilling the two set property (LJ1).

The S function is $S2C$ as it maps a sequent to a set of signed formulas. The formulas in the antecedent are given a T sign, and the formulas in the succedent are given an F sign, meeting the S function property (LJ2).

Suppose that φ is the proposed theorem. The endsequent k_0 at the root of the natural deduction proof contains only φ in its succedent. Hence, $S2C(k_0) = \{\langle F, \varphi \rangle\}$. The root sequent meets the initial container property (LJ3).

Lemmas 18, 19, and 20 established that the sequent expansion rules meet the expansion properties (LJ4a, LJ4b, and LJ4c). Since these three rule types cover each sequent rewrite rule, this meets the only α and β expansion rule property (LJ4d).

The axiom sequents for intuitionistic logic are the same as the ones used in classic logic; thus, an intuitionistic natural deduction sequent will close if it contains a contradictory formula or formulas. A sequent is an axiom if 1) it has $\bot$ in its antecedent, 2) it has $\top$ in its succedent, or 3) there is a formula that appears in both the antecedent and the succedent. An axiom sequent represents a container that has a contradiction. If the sequent(s) directly above K was created using by expanding a formula not in the N set, then K would also close since its child container(s) have closed. If the sequent directly above K was created by expanding a formula in K's N set, then K would also close since one of its N children has closed. There may have been several formulas in K's N set, but the one that

appears in the proof was chosen because it created a subtree that closed. This meets the closure property (LJ5).

Suppose that K is a sequent in a proof tree with axioms at each of its leaves. For each type of sequent rewrite rule, if the sequent(s) above K close, then K closes also. As one progress away from the leaves, each sequent closes because the sequent(s) above it close. The process ends when the initial sequent at the root of the proof tree closes, meeting the success property (LJ6). □

5.7.2 A Box in a Kripke Tableau as an Intuitionistic Container

Lemma 22 *If E is a formula in a Kripke box b, $SM_r(SF_k(E)) = \alpha$, and $J(SF_k(E)) \neq some\_future$, then the formulas added to b by expanding E meet the equality $K2C(KR_c(E)) = SM_c(SF_k(E))$.*

Proof:

These Kripke expansion rules are the same for both classical and intuitionistic logic. Thus, they have already been proven as part of Lemma 14.

Thus, the Kripke box meets the α and $J(E) \neq some\_future$ expansion property (LJ4a). □

Lemma 23 *If E is a KCF formula in a box b and $J(SF_k(E)) = some\_future$, then a new box b' is created by expanding E. This new box b' contains the forced formulas in b and the formulas in $KR_c(E)$, i.e. $T(K2C(b)) \cup SM_c(SF_k(E)) = K2C(b')$ where $T(C)$ is the set of signed formulas in a container C that are forced, i.e. have a T sign.*

Proof:

Suppose that E is a KCF formula in a Kripke box b and the box b' is created from b by expanding E.

The two Kripke expansion rules where $SM_r(SF_k(E)) = \alpha$ and $J(SF_k(E)) = some\_future$ are both rules for expanding N set formulas; these rules expand formulas that have one of two forms $\langle right, \varphi \Rightarrow \psi \rangle$ or $\langle right, \neg\varphi \rangle$.

In both cases below, suppose that $C = K2C(b)$, and that C is a container with the set of signed formulas that correspond to the set of KCF formulas in b. Let $T(C)$ be the set of forced formulas in container C and $F(b)$ be the set of unforced formulas in container C.

Case E is of the form $\langle right, \varphi \Rightarrow \psi \rangle$:

The contents of the new box b' are computed below:

$$T(K2C(b)) \cup SM_c(SF_k(E)) = T(C) \cup SM_c(SF_k(\langle right, \varphi \Rightarrow \psi \rangle))$$
$$= T(C) \cup SM_c(\langle F, \varphi \Rightarrow \psi \rangle) = T(C) \cup \{\langle T, \varphi \rangle, \langle F, \psi \rangle\} = K2C(b')$$

The contents of this new box computed match the contents of the box generated by expanding E.

Case E is of the form $\langle right, \neg\varphi \rangle$:

The contents of the new box b' are computed below:

$$T(K2C(b)) \cup SM_c(SF_k(E)) = T(C) \cup SM_c(SF_k(\langle right, \neg\varphi \rangle))$$
$$= T(C) \cup SM_c(\langle F, \neg\varphi \rangle) = T(C) \cup \{\langle T, \varphi \rangle\} = K2C(b')$$

The contents of this new box computed matches the contents of the box generated by expanding E.

In both cases when expanding a formula in the N set of a Kripke box, the box constructed meets the α and $J(E) = some\_future$ expansion property (LJ4b). □

Lemma 24 *If E is a KCF formula in the box b and $SM_r(SF_k(E)) = \beta$, then the expansion splits b into two boxes b_1 and b_2. The contents of b_1 and b_2 can be computed using the equations $K2C(b_1) = K2C(b) \cup \{SF_k(E_1)\}$ and $K2C(b_2) = K2C(b) \cup \{SF_k(E_2)\}$ where $\{E_1, E_2\} = KR_c(E)$.*

Proof:

If $KR_r(E) = split$, then E is one of three forms, $\langle left, \varphi \vee \psi \rangle$, $\langle right, \varphi \wedge \psi \rangle$, or $\langle left, \varphi \Rightarrow \psi \rangle$, in each case where $SM_r(SF_k(E)) = \beta$. Expanding E splits the box ($KR_r(E) = split$) if and only if E is in one of these three forms.

When E is expanded, the box b splits, creating boxes b_1 and b_2. One formula in $KR_c(E)$ is added to b_1 and the other to b_2. Thus, WLOG, assume that E_1 is added to b_1 and E_2 is added to b_2, then $K2C(b_1) = K2C(b) \cup \{SF_k(E_1)\}$ and $K2C(b_2) = K2C(b) \cup \{SF_k(E_2)\}$.

Thus, the Kripke box meets the β expansion property (LJ4c). □

Lemma 25 *A box in a Kripke tableau with $K2C$ as its S function is an intuitionistic container.*

Proof:

A box is separated into two columns, with the left column containing the forced formulas and the right containing the unforced formulas, meeting the two set property (LJ1).

The $K2C$ function maps the set of KCF formulas in a box to a set of signed formulas. The formulas in the left column of the box are mapped to the T set, and the formulas in the right column of the box are mapped to the F set, fulfilling the S function property (LJ2).

Suppose that φ is the proposed theorem. The initial box b_0 in a Kripke tableau proof contains only the formula φ in right column, i.e. $K2C(b_0) = \{\langle F, \varphi \rangle\}$. Hence, the initial box meets the initial container property (LJ3).

Lemma 22 proves that the box meets the α and $J(E) \neq some\_future$ expansion property (LJ4a); Lemma 23 proves that the box meets the α and $J(E) = some\_future$ expansion property (LJ4b), and Lemma 24 proves that the Kripke box meets the β expansion

property (LJ4c). All expansion rules in a Kripke tableau fall into one of these three types; thus, the Kripke box meets the only α and β expansion rules property (LJ4d).

If a box b closes because it is contradictory, then the container $K2C(b)$ also closes. There are three contradictory conditions to check:

- If a box b contains $\langle left, \bot \rangle$, then the box closes. Since $\langle T, \bot \rangle \in K2C(b)$, the container also closes.
- If a box b contains $\langle right, \top \rangle$, then the box closes. Since $\langle F, \top \rangle \in K2C(b)$, the container also closes.
- If a box b contains both $\langle left, \varphi \rangle$ and $\langle right, \varphi \rangle$ for some formula φ, then the box closes. Since $\{\langle T, \varphi \rangle, \langle F, \varphi \rangle\} \subseteq K2C(b)$, the container also closes.

Each condition that makes a Kripke box b contradictory causes it and the container $K2C(b)$ to close.

If E is a formula in a box b and $KR_r(E) = split$, then expanding E creates two β child boxes b_1 and b_2. If b_1 and b_2 both close, then b also closes.

In a box b when a formula E in the N set is expanded, an N child box b' is created. If b' closes, then b also closes. Thus, the Kripke box meets the closure property (LJ5).

A proof succeeds if all of the leaf boxes close by the argument above; the leaf boxes cause the ones directly above them to close, and the closing progresses upwards until the initial box at the root closes; thus, the sequent meets the success property (LJ6). □

5.7.3 An Intuitionistic Container for Analytic Tableau

This subsection proves that an ordered pair consisting of a branch and an element of a partial order, abbreviated as BPO, is the container for an intuitionistic analytic tableau. The BPO with b as its branch and p as its partial order element is denoted as $\langle b, p \rangle$. For

a vertex v in an analytic tableau, let $P(v)$ be a function the maps a vertex v to the partial order element labeling it.

The function $IB2C$ (short for intuitionistic branch to container) maps a branch b and an element of the partial order p (BPO) to a set of formulas. A formula E is in $IB2C(b, p)$ if it labels a vertex v on b and either 1) $P(v) = p$ or 2) E is forced and $P(v)$ proceeds p in the partial order:

$$IB2C(b, p) = \{E \mid E \in \langle b, p\rangle \text{ or } E \text{ is a forced and } E \in \langle q, b\rangle \text{ where } q \leq p\} \qquad (5.3)$$

Let B be the set of branches that pass through the vertex v that are labeled with the formula E and the partial order element p. When E is expanded, if $J(E) \neq some\_future$, then p' is p; otherwise a new element p' of the partial order is created as the immediate successor of p. Expanding E adds the formulas in $SM_c(E)$ to $\langle b, p'\rangle$ for each $b \in B$. The monotonicity rule also adds the forced formulas generated by each expansion to BPOs $\langle b, q\rangle$ where $b \in B$ and $q > p'$, i.e. to all BPOs which have a branch that passes through v and have a partial order element that is a successor of p'. An expansion that adds forced formulas updates all containers in known successor worlds of p' as well as successor worlds of p' that might be created by future expansions. This is the reason that the definition of $IB2C$ includes forced formulas created in predecessor worlds.

Theorem 26 *Using $IB2C$ as its S function, a BPO in an intuitionistic analytic tableau meets the expansion properties (LJ4a, LJ4b, LJ4c, and LJ4d) of the intuitionistic container.*

Proof:

At each step, a vertex v is expanded. Let b be a branch which contains v, E be the signed formula that labels v, p be the partial order element that labels v, and v_ℓ be the leaf vertex of b.

Case $SM_r(E) = \alpha$ and $J(E) \neq some\_future$:

This case behaves in the same way as an analytic tableau in classical logic. If $|SM_c(E)| = 1$, one vertex is added as the child of v_ℓ. On the other hand, if $|SM_c(E)| = 2$, one vertex is added as the child of v_ℓ and the other as the child of the first child (the grandchild of v_ℓ). The new vertices are labeled with formula(s) in $SM_c(E)$ and the partial order element p. If b' is the branch after the expansion, then $IB2C(b', p) = IB2C(b, p) \cup SM_c(E)$. This meets the α and $J(E) \neq some\_future$ property (LJ4a).

Case $SM_r(E) = \alpha$ and $J(E) = some\_future$:

The vertices added to the tableau are labeled with the same formulas as in the previous case. The difference is that these new vertices are labeled with a new element of the partial order. This new element p' is added as an immediate successor of p.

Suppose that b' is the branch after the expansion, then $IB2C(b', p') = T(IB2C(b, p)) \cup SM_c(E)$. This meets the α and $J(E) = some\_future$ expansion property (LJ4b).

Case $SM_r(E) = \beta$:

This case proceeds similarly to the classical version. The expansion of E adds vertices as the two children of v_ℓ, causing b to split into two branches b_1 and b_2, each labeled with a formula in $SM_c(E)$ and the partial order element p. Thus, $IB2C(b_1, p) = IB2C(b, p) \cup \{E_1\}$ and $IB2C(b_2, p) = IB2C(b, p) \cup \{E_2\}$ where E_1 and E_2 are the formulas in $SM_c(E)$. This meets the β expansion property (LJ4c).

The monotonicity rule allows a vertex v labeled with a forced formula E and the partial element p to be added as the child of a leaf vertex. The new vertex is labeled with E

and a partial order element q where $q > p$. The application of the monotonicity rule does not affect the contents of $IB2C(b, q)$ since the definition of $IB2C$ already includes E.

Since these expansion rules are the only way to create or modify the contents of a BPO and all the expansion rules fit into one of the three rule types, each handled be one of the three cases above. Therefore the BPO meets the only α and β-rule property (LJ4d). □

Lemma 27 *If a BPO $\langle b, p\rangle$ contains a contradiction, then the container $IB2C(b, p)$ also contains a contradiction.*

Proof:

In an analytic tableau, a BPO $\langle b, p\rangle$ closes under these conditions. 1) It contains a vertex on b labeled with partial order element p and either the formula $\langle T, \bot\rangle$ or $\langle F, \top\rangle$, then $\langle T, \bot\rangle \in IB2C(b, p)$ or $\langle F, \top\rangle \in IB2C(b, p)$; in either case $IB2C(b, p)$ contains a contradiction; 2) Let p be a partial element such that $p \leq q$ and φ is an unsigned formula. If a vertex on a branch b is labeled with $\langle T, \varphi\rangle$ and p and another vertex is labeled with $\langle F, \varphi\rangle$ and q, then $\{\langle T, \varphi\rangle, \langle F, \varphi\rangle\} \subseteq IB2C(b, q)$, and, hence, $IB2C(b, q)$ is contradictory. □

Theorem 28 *A BPO in an analytic tableau with $IB2C$ as its S function is an intuitionistic container.*

Proof:

Let $\langle b, p\rangle$ be a BPO containing the formulas labeling vertices on a branch b and labeled with the partial element p as well as the forced formulas labeling a vertex v on b where $P(v)$ proceeds p. The formulas in $\langle b, p\rangle$ can be separated into two sets T and F with the forced formulas in the T set and the unforced formulas in the F set. This meets the two set property (LJ1).

For a branch b and an element of the partial order p, the function $IB2C(b, p)$ is the set of signed formulas in $\langle b, p\rangle$. The signed formulas in $IB2C(b, p)$ can then be partitioned

into a T and an F set matching their signs. The $IB2C$ meets the requirements of the S function property (LJ2).

Suppose φ is the proposed theorem. The initial analytic tableau consists only of the root vertex labeled with $\langle F, \varphi \rangle$ and the partial element p_0. Since there is only one branch b_0 and only one element in the partial order p_0, then there is only one BPO $\langle b_0, p_0 \rangle$. Since $IB2C(b_0, p_0) = \{\langle F, \varphi \rangle\}$, the initial BPO meets the initial container property (LJ3).

Lemma 26 proved that a BPO meets the expansion properties (LJ4a-d).

Lemma 27 proved that if the BPO $\langle b, p \rangle$ contains a contradiction, then $IB2C(b, p)$ also contains a contradiction.

Let b_1 and b_2 be the two branches created by expanding a β-formula E on branch b. The split creates two branches, each considering one of two formulas that would satisfy the semantics of E. If both branches close, then both of the formulas in $SM_c(E)$ lead to contradictions, and, hence, branch b also has a contradiction.

If the BPO $\langle b, p' \rangle$ is found to have a contradiction and it was created by expanding an N set formula E in the BPO $\langle b, p \rangle$, then the BPO $\langle b, p \rangle$ also has a contradiction. The Kripke semantics of N set formulas guarantee that there exists a world where the Kripke semantics of the formula(s) in $SM_c(E)$ are satisfied. Since there is a contradiction in $\langle b, p' \rangle$, there is a violation of the Kripke semantics for an N set formula, creating a contradiction in $\langle b, p \rangle$.

When a BPO contains a contradiction, it closes; the three causes for this closure are 1) the BPO itself contains a contradiction, 2) all of the BPOs β children have contradictions, or 3) at least one of the BPOs N children has a contradiction. Together these three conditions meet the closure property (LJ5).

The closure process works from the BPOs on each branch where the contradictory formulas are found, closing BPOs as it progresses, until the initial BPO is closed, satisfying the success property (LJ6). □

5.7.4 A Path and T-String Pair a Matrix as an Intuitionistic Container

There is high confidence that a ordered pair consisting of a path and a T-string, similar to a BPO, can be proven to be an intuitionistic container. But the proof has not been found at this time, and thus it has been moved to future work.

5.8 Generalized Algorithm for Intuitionistic Logic

The generalized algorithm for intuitionistic logic, like its classical counterpart, determines if a formula is a theorem using only operations provided by the intuitionistic container; thus, it shows the commonalities of the different intuitionistic methods.

Each container in the intuitionistic generalized algorithm stores formulas in one of five sets:

- T_0 for forced formulas that have not been expanded
- T_1 for forced formulas that have been expanded
- F_0 for unforced formulas that have not been expanded
- F_1 for unforced formulas that have been expanded
- N for unforced formulas, where their expansion would create a new container. A formula E is placed in this set when $J(E) = some\_future$.

All formulas not in the N set are processed first; if the container closes when expanding an non-N formula, then the formulas in the N set do not have to be expanded. If this rule is violated, expanding a formula that creates an N child container before formulas not in the N set may cause a proof attempt to fail, while waiting could lead to a successful proof.

If a container C has formulas in its N set, a new container is created where each of these formulas is expanded. Since the Kripke semantics of these formulas affect only a non-empty subset of future worlds, it has to be assumed that these sets are disjoint. In contrast to the β rules, only one of the containers created by an expansion of an N set formula has to close for its parent container to close.

The generalized algorithm keeps track of which formulas are expanded by moving formulas between sets; when a forced formula is expanded, it is moved from T_0 to T_1, and when an unforced formula is expanded, it is moved from F_0 to F_1.

During each iteration of the main loop, the algorithm selects a formula E in $T_0 \cup F_0$ and processes it. If $J(E) \neq some\_future$, then the formulas in $SM_c(E)$ are processed in the same way as it in the classical generalized algorithm. However, when $J(E) = some\_future$, the formula is moved to the N set. After all formulas in T_0 and F_0 are expanded, then each formula in the N set is expanded separately.

The generalized algorithm starts when `intProofStart` is called. This algorithm is passed the proposed theorem φ and returns either `proven` or `notProven` or non-theorem. This algorithm creates the initial container C_0 with only φ in its F_0 set. The set of open containers $\mathcal{S}$ is created with C_0 as its only member. Then, `intProof` is called which contains the main loop of the generalized algorithm. Each trip through the loop consists of choosing an open container with an unexpanded formula E, calling `intExpandFormula` to expand E and checking if a termination condition has been met. If S is the empty set, then the algorithm terminates successfully, having proven the theorem. However, if S has an open container with no unexpanded formulas, then the algorithm terminates unsuccessfully (the proof has failed). In the loop, if a formula E is chosen for expansion and $J(E) = some\_future$, then E is placed in the N set. After both the T_0 and F_0 sets are emptied, then the formulas in the N set are processed by `intProcessNSet`. For each formula in N, `intProcessNSet` creates a new N child container. This N child

Comment	$\mathbf{T_0}$	$\mathbf{T_1}$	$\mathbf{F_0}$	$\mathbf{F_1}$	N
Initial state			1) $\varphi \vee \neg\varphi$		
Expand 1			2) φ (1) 3) $\neg\varphi$ (1)	1) $\varphi \vee \neg\varphi$	
Move 3 to N			2) φ (1)	1) $\varphi \vee \neg\varphi$	3) $\neg\varphi$ (1)
Expand 3 in N	4) φ (3)				
Open					

Table 5.3: Trace of the generalized algorithm for the formula $\varphi \vee \neg\varphi$ known as the excluded middle. A characteristic of intuitionistic logic is that the excluded middle is not a theorem.

container has a T_0 set consisting of all forced formulas, both expanded and unexpanded, from its parent and the formulas in $SM_c(E)$, but none of the unforced formulas from its parent. The reason is that expanded formulas are placed in the T_0 set is that formulas of the form $\langle T, \varphi \Rightarrow \psi\rangle$ and $\langle T, \neg\varphi\rangle$ generate unforced formulas. Marking all forced formulas as unexpanded allows the unforced formulas derived from forced formulas to be derived again. This step in the generalized algorithm implements the monotonicity property. If at least one of these N child containers closes, then the parent container can also be closed.

The algorithm `intExpandFormula` expands one formula, adding formulas to the container and splitting it if necessary. After the expansion, this algorithm returns only those containers that remain open. The final algorithm, `intCheckforClosure`, is passed a container C; if C is open, a set containing C is returned, but if C is closed, the empty set is returned so that C can be removed from the set of open containers $\mathcal{S}$.

Algorithm 5 intCheckForClosure(C: Container) returns an empty set or a set with one container.

if $\top \in F_1$ or $\bot \in T_1$ or a formula appears in both F_1 and T_1 **then**
  return $\emptyset$ {Return an empty set, indicating that the container is closed.}
else
  return $\{C\}$
end if

Algorithm 6 intExpandFormula(E:Signed Formula, C: Container) returns a set of containers.

if E is atomic **then**
  if E is forced **then**
    Add E to T_1.
  else { E is unforced.}
    Add E to F_1.
  end if
  $\mathcal{S} \leftarrow$ intCheckForClosure(C)
end if
if $SM_r(E) = \alpha$ **then**
  Add forced elements in $SM_c(E)$ to T_0.
  Add unforced elements in $SM_c(E)$ to F_0.
  $\mathcal{S} \leftarrow$ intCheckForClosure(C)
else { $SM_r(E) = \beta$ }
  Copy C to C_1 and C_2.
  for all $ch_i \in SM_c(E)$ **do**
    if ch_i is Forced **then**
      Add ch_i to T_0 in C_i
    else {ch_i is Unforced}
      Add ch_i to F_0 in C_i
    end if
  end for
  $\mathcal{S} \leftarrow$ intCheckForClosure(C_1) $\cup$ intCheckForClosure(C_2)
end if
return $\mathcal{S}$

Algorithm 7 intProof($\mathcal{S}$: set of containers) returns proven or notProven.

if all containers in $\mathcal{S}$ are closed **then**
  return proven
end if
while at least one container is open **do**
  Choose an open container C.
  Remove C from $\mathcal{S}$ {Let $T_0, T_1, F_0, F_1,$ and$N$ be the sets of formulas in C.}
  if $T_0 \cup F_0 \neq \emptyset$ **then**
    Choose a formula E in $T_0 \cup F_0$
    if $E \in T_0$ **then**
      Remove E from T_0.
      Add E to T_1.
    else { $E \in F_0$ }
      Remove E from F_0.
      Add E to F_1.
    end if
    if $J(E) = some\_future$ **then**
      Add E to N.
    else
      $\mathcal{S} \leftarrow \mathcal{S} \cup$ intExpandFormula(E, C)
    end if
  else { $F_0 \cup T_0 = \emptyset$ }
    return $intProcessNSet(C)$
  end if
end while

Algorithm 8 intProcessNSet(C: Container) returns proven or notProven.

for all $E \in N$ **do**
  Remove a formula E from the N set of C
  Create a new container C'.
  $T'_0 \leftarrow T_0 \cup T_1$ { T_0 may contain atomic formulas}
  $T'_1 \leftarrow \emptyset$
  $F'_0 \leftarrow \emptyset$
  $F'_1 \leftarrow \emptyset$
  $N' \leftarrow \emptyset$
  branchStatus $\leftarrow$ intProof($\{C'\}$)
  if branchStatus = Proven **then**
    return Proven
  end if
end for
return notProven

Algorithm 9 intStart(φ: Logical Formula) returns proven or notProven.

Construct a container C with all sets empty except for F_0 which contains φ. {where φ is the proposed theorem}
Let $\mathcal{S}$ be an empty set of containers.
$C \leftarrow \langle \emptyset, \emptyset, \{\varphi\}, \emptyset, \emptyset \rangle$ {Construct a container C with φ as the only formula in set F_0 and the other sets are empty.}
return intProof($\mathcal{S}$)

In traces of the algorithm, multiple containers are handled; there is a column for each set and a row for each step in the algorithm. Each container is identified by a sequence of numbers, with the empty sequence representing the root container. If a container is identified by the sequence of numbers K, then when a β-rule is applied to this container, two containers are created, one with the sequence of numbers $K, 1$ and the other with the sequence $K, 2$. This numbering system is designed to keep track of the multiple containers.

Comment	$\mathbf{T_0}$	$\mathbf{T_1}$	$\mathbf{F_0}$	$\mathbf{F_1}$	**N**
Initial state			1) $\varphi \Rightarrow (\varphi \vee \psi) \vee \neg\psi$		
Move 1 to N					1) $\varphi \Rightarrow (\varphi \vee \psi) \vee \neg\psi$
Expand 1	2) φ (1)		3) $(\varphi \vee \psi) \vee \neg\psi$ (1)		
Expand 3	2) φ (1)		4) $\varphi \vee \psi$(3)	3) $(\varphi \vee \psi) \vee \neg\psi$ (1)	
	2) φ (1)		5) $\neg\psi$ (3)		
Move 5 to N	2) φ (1)		4) $\varphi \vee \psi$(3)	3) $(\varphi \vee \psi) \vee \neg\psi$ (1)	5) $\neg\psi$ (3)
	2) φ (1)				
Expand 5 in N	2) φ (1)				
	6) ψ (5)				
Open					

Table 5.4: Trace of proof attempt for $\varphi \Rightarrow (\varphi \vee \psi) \vee \neg\psi$ that violates the expansion order of the generalized algorithm. This trace shows how the algorithm gives the wrong result by expanding Formula 5, which creates a new tableau, before expanding Formula 4, which does not create a new tableau.

The proof attempt for $\varphi \Rightarrow (\varphi \vee \psi) \vee \neg\psi$ shown in Table 5.4 fails because the expansion of Formula 5, $\neg\psi$, in N causes the formulas in F_0 and F_1 to be purged. Specifically, F_0 contains Formula 4, $\varphi \vee \psi$, so the expansion of Formula 5 purges Formula 4. If Formula 4 were expanded before Formula 5, then φ and ψ would be added to F_0, raising a contradiction with Formula 2. Since φ is in T_0, this contradiction closes the container.

Comment	$\mathbf{T_0}$	$\mathbf{T_1}$	$\mathbf{F_0}$	$\mathbf{F_1}$	$\mathbf{N}$
Initial state			1) $\varphi \Rightarrow (\varphi \vee \psi) \vee \neg\psi$		
Move 1 to N					1) $\varphi \Rightarrow (\varphi \vee \psi) \vee \neg\psi$
Expand 1	2) φ (1)		3) $(\varphi \vee \psi) \vee \neg\psi$ (1)		
Expand 3	2) φ (1)		4) $\varphi \vee \psi$(3) 5) $\neg\psi$ (3)	3) $(\varphi \vee \psi) \vee \neg\psi$ (1)	
Move 5 to N	2) φ (1)		4) $\varphi \vee \psi$(3)	3) $(\varphi \vee \psi) \vee \neg\psi$ (1)	5) $\neg\psi$ (3)
Expand 4	2) φ (1)		6) φ (4) 7) ψ (4)	3) $(\varphi \vee \psi) \vee \neg\psi$ (1) 4) $\varphi \vee \psi$(3)	5) $\neg\psi$ (3)
Closed 2 *and* 6					

Table 5.5: Trace of the generalized algorithm using an expansion order that proves the theorem $\varphi \Rightarrow (\varphi \vee \psi) \vee \neg\psi$. Unlike the above trace, in this trace the expansion of Formula 5 is deferred and the expansion of 4 generates formulas allowing the container to close.

However, the trace in Table 5.5 shows another proof attempt to prove this formula that expands all formulas in $T_0 \cup F_0$ before it expands an N set formula; thus, the generalized algorithm is successful.

The proof attempt for $(\varphi \Rightarrow \psi) \Rightarrow (\neg\varphi \vee \psi)$ shown in Table 5.6 fails because there is an open container in which all formulas have been expanded. This formula is not a theorem in intuitionistic logic.

5.9 Concluding Remarks on Intuitionistic Logic

This chapter began by describing the philosophical differences between intuitionistic and classical logic. Next, Kripke semantics defined the semantics of the connectives in terms of forcing. The J function was introduced, together with SM_r and SM_c. These functions were used to describe the actions taken for each expansion in the intuitionistic versions of the proof methods. Using these functions, the properties of the intuitionistic container were specified. The data structures used in each of the intuitionistic methods were shown to meet the properties of the container. Finally, the container played a central role in the intuitionistic generalized algorithm.

Comment	$\mathbf{T_0}$	$\mathbf{T_1}$	$\mathbf{F_0}$	$\mathbf{F_1}$	**N**
Initial state			1) $(\varphi \Rightarrow \psi) \Rightarrow (\neg\varphi \vee \psi)$		
Move 1 to N					1) $(\varphi \Rightarrow \psi) \Rightarrow (\neg\varphi \vee \psi)$
Expand 1	2)$\varphi \Rightarrow \psi$ (1)		3) $\neg\varphi \vee \psi$ (1)		
Expand 2 Container 1		2)$\varphi \Rightarrow \psi$ (1)	3)$\neg\varphi \vee \psi$ (1) 4) φ (2)		
Expand 3 Container 1		2)$\varphi \Rightarrow \psi$ (1)	4) φ (2) 6) $\neg\varphi$ (3) 7) ψ (3)	3)$\neg\varphi \vee \psi$ (1)	
Move 6 to N Container 1		2)$\varphi \Rightarrow \psi$ (1)	4) φ (2) 7) ψ (3)	3)$\neg\varphi \vee \psi$ (1)	6) $\neg\varphi$ (3)
Expand 6 Container 1	2)$\varphi \Rightarrow \psi$ (1) 8) φ (3)				
Expand 2 Container 1, 1	8) φ (6)	2)$\varphi \Rightarrow \psi$ (1)	9) φ (2)		
Closed 8 *and* 9					
Expand 6 Container 1, 2	10) ψ (2) 8) φ (6)	2)$\varphi \Rightarrow \psi$ (1)			
Open					
Expand 2 Container 2	5) ψ (2)	2) $\varphi \Rightarrow \psi$ (1)	3) $\neg\varphi \vee \psi$ (1)		
Expand 3 Container 2	5) ψ (2)	2) $\varphi \Rightarrow \psi$ (1)	11) $\neg\varphi$ (3) 12) ψ (3)	3) $\neg\varphi \vee \psi$ (1)	
Closed 5 *and* 12					

Table 5.6: Proof trace of the formula $(\varphi \Rightarrow \psi) \Rightarrow (\neg\varphi \vee \psi)$

The container is concerned with a single expansion within it and checking the closure conditions of a single container, while the generalized algorithm manages the overall proof. The container and generalized algorithm expose the commonalities of each method. It also shows that the order in which formulas in the N set are expanded is critical to the success of the proof attempt.

Chapter 6

Conclusion

This research has explored the commonalities of four well-established methods used in classical and intuitionistic logic: natural deduction, Kripke tableau, analytic tableau, matrix, and resolution (classical logic only). Classical logic, followed by intuitionistic logic, were discussed separately, but each had a similar three-step development. First, several proof methods for that logic were described. Second, a container property was introduced that identified common aspects of these methods. Third, a generalized algorithm was introduced that used only the operations in the container property. The common features captured by the container and used the generalized algorithm include: how formulas expansion modified and created new containers, the closure conditions, and success requirement.

6.1 Containers and Their Generalized Algorithms

The container property specifies requirements for a single expansion, while the generalized algorithm manages the proof itself. The container property 1) requires that formulas can be identified as belonging to one of two sets T and F, 2) describes how Smullyan's

rules are used to expand them, adding formulas to the containers and splitting them, and 3) specifies the closure conditions for a container. The generalized algorithm manages the proof by initializing the first container, managing the set of containers, using the expansion rules of the container to modify it, detecting if it has closed, and determining when a proof has terminated (successfully or unsuccessfully).

6.2 The Benefits of Identifying Commonalities

The reason for introducing containers and the generalized algorithms was to reveal the commonalities of these methods. Doing so makes it is easier to compare how these methods construct proofs.

In classical logic, the properties of a container employ Smullyan's rules. For each method there was a proof that demonstrated the expansion rules met the container's expansion property. Together these proofs illustrate that each formula is expanded either explicitly using Smullyan's rules or rewrite rules that can be shown to be equivalent to them. To facilitate the discussion of Smullyan's rules, this research introduced two functions SM_r and SM_c. The definitions of the functions KR_r and KR_c for C-Kripke tableau were chosen both to match the semantics of Kripke's rewrite rules and to facilitate comparisons with Smullyan's functions.

In moving from classical logic to intuitionistic logic, a new function, J, was introduced. This function indicates the special processing required to expand a formula. J, together with SM_r and SM_c, describes the actions that need to be taken to construct an intuitionistic proof. The intuitionistic container property uses these three functions to describe its requirements.

The properties of a container codify the common closure and rewrite rules of the various methods. If other methods can be proven to have the container property, then they

become full members with the other container methods. A switch can be made between any two container methods. In addition, when a theorem or heuristic is established for a container and/or its associated algorithm, then that theorem or heuristic applies to all container methods.

6.3 Commonalities Permitting Switching

Research [Tammet 1996] has shown that some theorems can be proved in less than a tenth of a second using resolution but require several seconds or minutes using an analytic tableau; for some theorems this situation is reversed: analytic tableau is faster than resolution. One solution is to run two or more methods as separate tasks and use the answer of the one that terminates first. Alternatively switching between methods during a proof may allow for better results than any single method.

The commonalities between these methods are sufficient to allow a switch between methods during a proof. A proof begins using one method, performing some expansion steps; then its data structures are transformed into a second method's data structures. The proof continues using this second method. The containers and generalized algorithms introduced in this research for classical and intuitionistic logic are robust enough to support this switching between methods. A switch which is made between expansion steps, can be made as often as desired without requiring any expansions to be repeated in the proof.

6.4 The Importance of the N Set in Intuitionistic Logic

The importance of the N set formulas in proving intuitionistic logic theorems became apparent during the writing and debugging of a program to implement the Kripke

method. Learning how to handle N set formulas in Kripke tableaux assisted in the understanding of the other intuitionistic methods.

The β-rule applications increase the number of branches in a tree, making a proof more difficult to complete since every branch must close for the proof to be successful. Expanding a formula in the N set creates a different kind of split; where only one branch needs to close for the proof to be successful.

The order in which formulas in the N set are expanded is critically important to finding a proof. If a formula in the N set is expanded, the F set of that container is purged. However, in classical logic formulas are never purged. They accumulate as expansions are performed. Therefore, if a proof exists, it will always be found regardless of the expansion order used. In intuitionistic logic each formula in a container's N set is expanded separately. Expanding a formula E in the N creates a new container. This new container includes the forced formulas from its parent and the formulas in $SM_c(E)$; the unforced formulas are purged, including the other formulas in the N set, requiring a choice to be made. If one choice does not lead to a proof, then the program backtracks to make another choice.

A good heuristic for choosing which formula to expand in an N set would be useful and is an area for future work. Here two heuristics are proposed. First, if an unforced formula E can be derived from a formula E' in the T set, then the monotonicity rule allows E to be derived again after another member of the N set has been expanded. The second heuristic examines the T set for formulas that contain variables that appear in formula E; if few or none of the variables are shared, then expanding E is not likely to generate a contradiction.

6.5 Future Work

The research presented in this research builds upon the research of others and hopefully is a foundation that will be used for future work. This subsection discusses some areas that should be investigated based on the research presented here. Some of these areas include: 1) Proving that other proof methods for classical or intuitionistic logic can be described using containers. 2) Using the insights gained from the existing container methods to investigate how optimizations and heuristics for one method can be adapted to the others. 3) Identifying the conditions when a switch between methods should be made and the method to switch to. 4) Adapting the container for use with logics other than classical and intuitionistic logic. 5) Modifying the containers to support first order logic. 6) Proving that the intuitionistic matrix uses a container, deferred from Subsection 5.7.4.

The identification of the similarities between the methods allows optimizations and heuristics to be potentially shared between the different methods. What does an optimization or heuristic look like when it is translated to another method? It might not make sense, not apply, be another well-known optimization or heuristic, or perhaps be something new. With the identification of the similarities, new heuristics and optimizations can be adapted for other methods more easily.

Since β-rules split containers, these methods might be well-suited to a distributed computing environment, where each task can be given a different part of the proof tree to solve. After each task completes, its results can be combined with the results of the other tasks.

The switching feature of the generalized algorithm raises many questions including: When should one method be switched to another? What is the overhead required to make a switch? What properties of a formula make it best suited to a given method? The answers

to these questions would improve the decisions for the following questions: Which method should be used to begin a proof? When a switch should be made? Which method should the switch be made to?

Classical and intuitionistic logic are only two of many types of logic. This research defined classical logic using the Smullyan functions SM_r and SM_c. Intuitionistic logic was characterized by adding the J function to the Smullyan functions. The J function modifies the actions taken when certain formulas are expanded. In describing the semantics of other logics, perhaps a function similar to the intuitionistic logic's J function could be used to correctly implement the semantics of that logic. There is a close relationship between modal-S4 logic and intuitionistic logic, thus modal-S4 should be one of the first logics for which a new container should be designed. Modal logics have two extra connectives; therefore, by extending SM_r and SM_c to handle the new connectives and modifying the J function, it is likely that these functions could be used in an algorithm for modal-S4 logic. There are other modal logic types for which rules could be developed.

It is also desirable to modify both the classical and intuitionistic containers and generalized algorithms to support first-order logic. First-order logic supports the quantifiers for all ($\forall$) and there exists ($\exists$), predicates, and functions. These extra elements add succinctness and expressiveness to the language. For instance, the statement "Given a positive integer x there exists a positive integer y such that $x + y = 15$." can be expressed using the first-order formula $\forall x \exists y A(x, y, 15) \wedge P(x) \wedge P(y)$ where P is a predicate that is true only when its argument is a positive integer and A is a predicate that is true only when the the sum of its first two arguments equals the third. Assuming the x and y are both less than fifteen, this can be encoded in propositional logic by combining two sets of formulas, the first indicating that there are two integers, each having a single value between one and fourteen, and the second writing formulas indicating which combinations of the values of the two integers equal fifteen. This propositional formula would be long and complex

and would have to be drastically modified if the question is even slightly changed, such as changing the sum to sixteen.

6.6 Concluding Summary

This research has shown that many common methods used in classical and intuitionistic logic share a common algorithm. This research defined a container, proved several methods had a data structure that has the properties of a container, and created a generalized algorithm. The container provides a simpler solution revealing the important commonalities shared by the methods. It allows proofs about the container and generalized algorithm to apply to all methods that share these properties.

Instead of constructing an ordinary spanning tree of equivalences between the methods, this research created a star-like spanning tree by adding a new vertex with the container and generalized algorithm. Each time a proof method was shown to have a data structure and a S function met the properties of a container, an edge was added from that method to the generalized algorithm. This equivalence argument was repeated for intuitionistic logic. The vertex at the center of this star graphs is a generalized algorithm exposing the commonalities of the methods.

Bibliography

AHO, A. V., LAM, M. S., SETHI, R., AND ULLMAN, J. D. 2006. *Compilers, Principles, Techniques, and Tools*, second ed. Addison-Wesley, Reading, MA.

ANDREWS, P. B. 1981. Theorem proving via general mappings. *Journal of the ACM 28,* 2, 193–214.

BACHMAIR, L. AND GANZINGER, H. 2001. Resolution theorem proving. In *Handbook of Automated Reasoning*, A. Robinson and A. Voronkov, Eds. Vol. 1. Elsevier Science B. V., Amsterdam, The Netherlands, Chapter 2, 19–178. Co-published by MIT Press of Cambridge, MA.

BETH, E. W. 1956. Semantic construction of intuitionistic logic. *Mededelingen der Koninklijke Nederlandse Akademie van Wentenschappen, Afd. Letterkunde, Nieuwe Reeks 19,* 11, 357–388.

BETH, E. W. 1966. *The Foundations of Mathematics*, Harper Torchbook ed. Harper & Row, New York.

BIBEL, W. 1981. On matrices with connections. *Journal of the ACM 28,* 4, 633–645.

BIBEL, W. 1987. *Automated Theorem Proving*, second ed. Vieweg, Braunschweig, FDR.

BOOLE, G. 1854. *An Investigation of the Laws of Thought on Which are Founded the Mathematical Theories of Logic and Probabilities.* Macmillan, New York, NY. Reprinted with corrections, Dover Publications, New York, NY, 1958.

DAVIS, M. 2001. The early history of automated deduction. In *Handbook of Automated Reasoning*, A. Robinson and A. Voronkov, Eds. Vol. 1. Elsevier Science B. V., Amsterdam, The Netherlands, Chapter 1, 3–15. Co-published by MIT Press of Cambridge, MA.

DAVIS, M. AND PUTNAM, H. 1960. A computing procedure for quantification theory. *Journal of the ACM 7,* 3, 201–215.

DEGTYAREV, A. AND VORONKOV, A. 2001. The inverse method. In *Handbook of Automated Reasoning*, A. Robinson and A. Voronkov, Eds. Vol. 1. Elsevier Science B. V., Amsterdam, The Netherlands, Chapter 4, 179–272. Co-published by MIT Press of Cambridge, MA.

DOYLE, E. 2003. A comparison of tableaux and resolution theorem proving procedures. In *ACMSE '03 - ACM Southeast Conference*, M. Burge, Ed. ACM, ACM, New York, NY, 230–234.

DUFFIN, R. J. 1965. Topology of series-parallel networks. *Journal of Mathematical Analysis and Applications 10*, 303–318.

ENDERTON, H. B. 2001. *A Mathematical Introduction to Logic*, second ed. Harcourt/Academic Press, Burlington, MA.

FITTING, M. 1996. *First-Order Logic and Automated Theorem Proving*, second ed. Springer-Verlag, New York.

FITTING, M. C. 1969. *Intuitionistic Logic Model Theory and Forcing*. Studies in Logic and the Foundations of Mathematics. North-Holland, Amsterdam.

GAREY, M. R. AND JOHNSON, D. S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York.

GENTZEN, G. 1935. Untersuchungen über das logische schliessen. *Mathematische Zeitschrift 39*, 176 – 210, 405–431. Translation in Szabo M.E., editor, The collected papers of Gerhard Gentzen, Chapter 3, 68-131, North-Holland, Amsterdam, 1969.

GÖDEL, K. 1967. On formally undecidable propositions of *Principia Mathematica* and related systems I. In *From Frege to Gödel: A Source Book in Mathematical Logic, 1879–1931*, J. van Heijenoort, Ed. Harvard University Press, Cambridge, MA, 596–616. Originally published in 1931, this version is an English translation with commentary.

HÄHNLE, R. 2001. Tableaux and related methods. In *Handbook of Automated Reasoning*, A. Robinson and A. Voronkov, Eds. Vol. 1. Elsevier Science, Amsterdam, The Netherlands, Chapter 3, 100–178. Co-published by MIT Press of Cambridge, MA.

KNEALE, W. AND KNEALE, M. 1962. *The Development of Logic*. Oxford University Press, Oxford, UK.

KRIPKE, S. 1965. Semantical analysis of intuitionistic logic I. In *Formal Systems and Recursive Functions*, J. Crossley and M. A. E. Dummett, Eds. North-Holland Publishing Company, Amsterdam, 92–130. (Proceedings of the Eighth Logic Colloquium, Oxford, July 1963).

LIS, Z. 1960. Wynikanie semantyczne a wynikanie formalne. *Studia Logica 10*, 39–60. Logical consequence, semantic and formal.

MARQUIS, J.-P. Fall 2007. Category theory. In *The Stanford Encyclopedia of Philosophy*, E. N. Zalta, Ed.

MINTS, G. 1994. Resolution strategies for the intuitionistic logic. In *Constraint Programming*, B. Mayoh, E. Tyugu, and J. Penjam, Eds. NATO ASI Series F, vol. 131. Springer Verlag, Berlin, 289–311.

MOSCHOVAKIS, J. Spring 2007. Intuitionistic logic. In *The Stanford Encyclopedia of Philosophy*, E. N. Zalta, Ed.

MURRAY, N. V. AND ROSENTHAL, E. 1990. Reexamining intractability of tableau methods. In *Proceedings of the international symposium on Symbolic and algebraic computation*. ACM, New York, NY, 52–59.

NERODE, A. AND SHORE, R. A. 1993. *Logic For Applications*. Springer-Verlag, New York.

OTTEN, J. 1997. ileanTAP: An intuitionistic theorem prover. In *Proceedings of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*. Springer-Verlag, London, UK, 307–312.

OTTEN, J. AND KREITZ, C. 1996. T-string-unification: Unifying prefixes in non-classical proof methods. In *5^{th} Workshop on Theorem Proving with Analytic Tableaux and Related Methods*, U. Moscato, Ed. Lecture Notes in Artificial Intelligence, vol. 1071. Springer Verlag, Berlin, 244–260.

PIERCE, B. 1991. *Basic Category theory for Computer Scientists*. Massachusetts Institute of Technology, Cambridge Massachusetts.

PRAWITZ, D., PRAWITZ, H., AND VOGHERA, N. 1960. A mechanical proof procedure and its realization in an electronic computer. *Journal of the ACM 7*, 2, 102–128.

ROBINSON, J. A. 1965. A machine-oriented logic based on the resolution principle. *Journal of the ACM 12*, 1 (Jan), 23 – 41.

SAHLIN, D., FRANZÉN, T., AND HARIDI, S. 1992. An intuitionistic predicate logic theorem prover. *Journal Logic Computation 2*, 5, 619–656.

SMITH, R. Winter 2007. Aristotle's logic. In *The Stanford Encyclopedia of Philosophy*, E. N. Zalta, Ed.

SMULLYAN, R. M. 1995. *First-order Logic*. Dover, Mineola, NY. (The Dover edition is an unabridged, corrected republication of this work first published by Springer-Verlag, New York, 1968.).

TAMMET, T. 1996. A resolution theorem prover for intuitionistic logic. *Lecture Notes in Computer Science 1104*, 2–16.

ULLMAN, J. D. 1998. *Elements of ML Programming*, ML97 ed. Prentice-Hall, Upper Saddle River, New Jersey.

VAN DALEN, D. 1994. *Logic and Structure*, third ed. Springer-Verlag, Berlin.

VON PLATO, J. Summer 2008. The development of proof theory. In *The Stanford Encyclopedia of Philosophy*, E. N. Zalta, Ed.

WAALER, A. 2001. Connections in nonclassical logics. In *Handbook of Automated Reasoning*, A. Robinson and A. Voronkov, Eds. Vol. 2. Elsevier Science B. V., Amsterdam, The Netherlands, Chapter 22, 1487–1578. Co-published by MIT Press of Cambridge, MA.

WALLEN, L. 1990. *Automated Proof Search in Non-classical Logics.* MIT Press, Cambridge, MA.

WEST, D. B. 1996. *Introduction to Graph Theory*. Prentice-Hall, Inc., Upper Saddle River, New Jersey.

WOS, L., OVERBEEK, R., LUSK, E., AND BOYLE, J. 1984. *Automated Reasoning: Introduction and Applications.* Prentice-Hall, Englewood Cliffs, New Jersey.