Implementing Tableaux by Decision Diagrams

Jean Goubault-Larrecq
Institut für Logik, Komplexität und Deduktionssysteme
Universität Karlsruhe, D-76128 Karlsruhe*†
Jean.Goubault@frcl.bull.fr

August 29, 1996

Abstract

Binary Decision Diagrams (BDDs) are usually thought of as devices engineered specially for classical propositional logic. We show that we can build on one of their variants, Minato's zero-suppressed BDDs, to build compact data structures that encode whole tableaux. We call these structures tableaux decision diagrams (TDDs), and show how tableaux proof search is implemented in this framework. For this to be efficient, we have to restrict to canonical proof formats (in the sense of Galmiche et al.) to be able to take advantage of sharing in TDDs. Sharing is fundamental, not because it reduces memory consumption, but because it allows us to expand or close many tableaux paths in parallel, with corresponding gains in efficiency. We provide some empirical evidence that this is indeed efficient, by illustrating the method on a well-chosen system for propositional intuitionistic logic.

1 Introduction

Binary Decision Diagrams, or *BDDs*, are simple data structures that represent both the sets of models and of counter-models to a given propositional formula. They have enjoyed great popularity in hardware verification circles since their (re-)invention by Bryant in 1986 [1], and in other domains as well, including automated deduction [19, 10, 11]. However, BDDs only apply to classical propositional logic, excluding important logics like intuitionistic logic or modal logics.

On the other hand, tableau and sequent calculi offer a flurry of methods for solving the satisfiability problem in classical and non-classical logics [5]. But it is unclear how to adapt BDDs to the latter. It is precisely the purpose of this paper to show one possible solution: we use Shin-Ichi Minato's zero-suppressed BDDs [16], augmented by a new reduction rule that closes paths, to represent whole tableaux in memory.

The plan of this paper is as follows. In Section 2, we recap a few basic notions about zero-suppressed binary decision diagrams, and introduce our approach to implement tableaux by using similar structures, which we call Tableau Decision Diagrams (TDDs). In Section 3, we then proceed to develop a method based on TDDs for finding proofs in propositional intuitionistic logic. We take the latter as a case study, because it is one of the most important non-classical logics. This method is in fact applicable to many other logics, but crucially relies on the existence of a Gentzen system for the logic with many nice combinatorial properties, i.e. permutabilities and invertibility of rules, to make full use of the potential for parallelism offered by TDDs. We validate our approach in Section 4 by testing it on several benchmark problems, compare our approach to other works in Section 5, and conclude in Section 6.

*Research partially funded by the HCM grant 7532.7-06 from the European Union. Most of the research done in this paper was done while I was still at Bull S.A.

†On leave from Bull Corporate Research Center, rue Jean Jaurès, F-78340 Les Clayes sous Bois.

2 Zero-Suppressed Binary Decision Diagrams

2.1 Formulas, Paths, Tableaux

We consider a vocabulary of *propositional formulae* Φ, Ψ, ... built from *propositional variables* or *atoms* A, B, C, ... using operators & (conjunction), $\vee$ (disjunction), $\bot$ (false), $\supset$ (implication). We let $\sim\Phi$ stand for $\Phi \supset \bot$.

A *signed formula* $s\Phi$ is either $+\Phi$ or $-\Phi$, where Φ is a formula and $s \in \{+, -\}$ is a sign. A *path* p is a finite set of signed formulae. A *tableau* t is a finite set of paths.

A *tableau calculus* is a finite set of rules that transform paths into zero, one or more paths. To prove a formula Φ, we start with the tableau $\{\{-\Phi\}\}$, and rewrite paths in the tableau until we get the empty tableau (in which case Φ is proved), or until we have explored all tableaux that are reachable from the initial tableau without obtaining the empty tableau (in which case Φ is unprovable); otherwise, the process does not terminate.

Searching for a proof is a non-deterministic process: we have to find whether there exists a derivation of the empty tableau from the initial tableau. This non-determinism may be implemented in several ways, but the most usual one is by *backtracking*: if we can rewrite a tableau T into several possible tableaux T_1, ..., T_n, then try to reach the empty tableau from T_1, then from T_2 if we did not succeed, then ... from T_n if we did not succeed earlier. This kind of implementation does not require that we keep whole tableaux in memory, as only paths are changed from one tableau to the next. Therefore, we only keep paths (even only signed formulas in classical logic) during the whole tableau search process. This uses very little memory, but it is also the reason why tableaux seem to redo the same proof work over and over on different branches of the search tree.

Another way of implementing tableaux would be to keep whole tableaux in memory, and to share their paths in some way, so as to be able to reason on several paths *in parallel* even on a sequential machine. That is, if several paths share a common subpath, and this subpath is provable (we can reach the empty tableau from it), then closing this one subpath will close all the paths that contain it at once. Moreover, if there is enough sharing among paths, the penalty for keeping whole tableaux in memory will be offset somewhat.

The latter approach is quite similar to that of binary decision diagrams [1], or BDDs. In classical propositional logic, we can try to prove a formula by non-deterministically searching for an invalidating assignment: either by backtracking (as in the Davis-Logemann-Loveland, better known as the Davis-Putnam method [2]), or by representing all assignments in memory, and mapping each to the value of the given formula under the assignment. The latter is precisely what BDDs do.

2.2 Zero-Suppressed BDDs

Zero-suppressed BDDs, or 0-sup-BDDs, are a slight modification of the binary decision diagram data structure, and are due to Minato [16]. Their use is not in representing formulas (up to propositional equivalence), but in representing sets of cubes, where a cube is a conjunction of Boolean variables. They provide a compact representation of sets of cubes, which have successfully been used in hardware circles to compute sets of prime implicants of or essential prime implicants of impressive sizes (op.cit.)

Our aim here is to use 0-sup-BDDs to represent tableaux, i.e. we trade propositional variables for signed formulas. We present 0-sup-BDDs in this framework. Most definitions are due to Minato, and the only new technique in this section is the closing reduction rule (2).

Let **1** (truth) and **0** (falsehood) be two new symbols. A *binary decision diagram*, or *BDD* F, is defined inductively as either **1**, **0** or a triple $(s\Phi, F_0, F_1)$. Graphically, we write the latter as in Figure 1, where the second component F_0 (the **0** *branch*) is linked to the root by a dotted line, and the third component F_1 (the **1** *branch*) is linked to the root by a solid line. In plain text, we shall write $s\Phi \longrightarrow F_1/F_0$ for this triple. We shall also call $s\Phi$ a *kernel* of the BDD — we won't use kernels as propositional variables, as is usual, hence the new name.

Any BDD represents a tableau, as follows:

Definition 1 *Given a BDD F, the* tableau $T(F)$ associated with F *is defined as follows:* $T(\mathbf{1})$ *is the tableau* $\{\emptyset\}$ *that only contains the empty path,* $T(\mathbf{0})$ *is the empty tableau* $\emptyset$*, and* $T(s\Phi \longrightarrow F_1/F_0) = \{\{s\Phi\} \cup p \mid p \in T(F_1)\} \cup T(F_0)$.

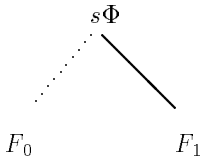

Figure 1: A node in a BDD

BDDs are *shared*, that is, they are implemented not as trees, but as directed acyclic graphs. We use a global hash-table *uniq* recording all previously allocated BDD nodes. Then, to allocate a BDD node $s\Phi \longrightarrow F_1/F_0$, we first look *uniq* up to see whether there is a node with all three matching components; if so, we return this node, otherwise, we generate a new node containing $s\Phi$, F_0 and F_1, add it to the *uniq* hash-table, and return it.

Ordered BDDs, or *OBDDs*, are BDDs such that on any path from the root to the leaves, kernels are sorted in increasing order with respect to a given total ordering $<$. Apart from sharing, the choice of ordering is the most important factor for getting small BDDs for large tableaux. We shall delay the question of finding good orderings until Section 4, and shall assume that all our BDDs are ordered by a given ordering.

A *zero-suppressed BDD* is a BDD that is reduced by the following reduction rule:

$$(s\Phi \longrightarrow \mathbf{0}/F_0) \longrightarrow F_0 \tag{1}$$

Observe that the tableaux of the left-hand and right-hand sides are identical.

Because the kernels of BDDs are signed formulas, we wish to find another reduction rule that would automatically eliminate closed paths from a BDD's tableau. To this end, we shall assume that the BDD ordering is of the form: $\ldots < +\Phi_0 < -\Phi_0 < +\Phi_1 < -\Phi_1 < \ldots < +\Phi_k < -\Phi_k < \ldots$, i.e. that $<$ is the lexicographic ordering on signed formulas that compares the formulas first (with respect to some ordering), and then the signs. That $+ < -$ is not essential, and we might have chosen $- < +$ instead. What is important is that the kernels $+\Phi_k$ and $-\Phi_k$ are next to each other. This allows us to make sense of the following reduction rule:

$$(+\Phi \longrightarrow (-\Phi \longrightarrow F_2/F_1)/F_0) \longrightarrow (+\Phi \longrightarrow F_1/F_0) \tag{2}$$

which eliminates all paths that contain both $+\Phi$ and $-\Phi$. Therefore, a BDD which is irreducible by rule (2) has a tableau that contains no closed path. We call this rule the *closing reduction rule*. This leads to our final notion of BDD:

Definition 2 *Let $<$ be a given total strict ordering on formulas. Write again $<$ the strict ordering on signed formulas defined by: $s\Phi < s'\Phi'$ iff $\Phi < \Phi'$, or $\Phi = \Phi'$, $s = +$ and $s' = -$.*

A zero-suppressed BDD, *or* 0-sup-BDD, *is a BDD that is shared, ordered by $<$ and reduced by rule (1).*

A tableau decision diagram, *or* TDD *F, G, H, ... is a 0-sup-BDD that is reduced by rule (2) as well.*

The crucial observation here is that the closing reduction rule closes several, possibly many paths in exactly one step: imagine that F_2 contains, say, 1000 unclosed paths (which is perfectly reasonable), then instead of closing all paths $p, -\Phi, +\Phi$ where p ranges over the 1000 paths in F_2, the closing reduction rule closes them all in parallel. This parallelism in the implementation of tableau rules in TDDs is exactly what we are after: TDDs automatically close paths in parallel, and we shall see how we can apply most other tableau rules in parallel as well in Section 3, on a case study on propositional intuitionistic logic.

To make precise the relation between TDDs and tableaux, we define an inverse to T as follows:

Definition 3 *Let D be the function from tableaux to 0-sup-BDDs defined as follows: $D(\emptyset) = \mathbf{0}$, $D(\{\emptyset\}) = \mathbf{1}$, and if t is a tableau containing at least one signed formula, let $D(t)$ be $s\Phi \longrightarrow D(t_1)/D(t_0)$, where $s\Phi$ is the least (for $<$) signed formula occurring in t, t_0 is the set of all paths in t not containing $s\Phi$, and t_1 is the set of all paths $p \setminus \{s\Phi\}$, where p ranges over all paths in t containing $s\Phi$.*

Observe that the definition is well-founded, as it proceeds by recursion over the number of signed formulas that occur in the tableau.

Lemma 1 *For any tableau t, $T(D(t)) = t$.*

Proof: By induction on the number of signed formulas that occur in t. If $t = \emptyset$, then $T(D(t)) = T(\mathbf{0}) = t$; if $t = \{\emptyset\}$, then $T(D(t)) = T(\mathbf{1}) = t$. Otherwise, let $s\Phi$ be the least signed formula occurring in t. Then $D(t) = s\Phi \longrightarrow D(t_1)/D(t_0)$ as in Definition 3, and $T(D(t)) = \{\{s\Phi\} \cup p \mid p \in T(D(t_1))\} \cup T(D(t_0))$ by Definition 1. By induction hypothesis, $T(D(t)) = \{\{s\Phi\} \cup p \mid p \in t_1\} \cup t_0 = t$ by definition of t_0 and t_1. □

Lemma 2 *For any 0-sup-BDD F, $D(T(F)) = F$.*

Proof: By induction on the number of kernels in F. If $F = \mathbf{0}$, then $D(T(F)) = D(\emptyset) = F$; if $F = \mathbf{1}$, then $D(T(F)) = D(\{\emptyset\}) = F$. Otherwise, F is of the form $s\Phi \longrightarrow F_1/F_0$, with $s\Phi$ less than any kernel occurring in F_0 or F_1. Then $T(F) = \{\{s\Phi\} \cup p \mid p \in T(F_1)\} \cup T(F_0)$ by Definition 1. Because F is reduced by rule 1, F_1 is not $\mathbf{0}$, hence $T(F_1)$ is not empty. So $s\Phi$ occurs in $T(F)$, and is necessarily the least signed formula occurring in $T(F)$. Therefore $D(T(F)) = s\Phi \longrightarrow D(T(F_1))/D(T(F_0))$, which is equal to $s\Phi \longrightarrow F_1/F_0$ by induction hypothesis, that is to F. □

We may conclude:

Theorem 3 *The pair D, T defines a bijection between tableaux and 0-sup-BDDs. Moreover, it defines a bijection between tableaux with no closed paths and TDDs.*

Proof: The first part of the claim follows directly from Lemmas 1 and 2. As for the second part, let F be an arbitrary 0-sup-BDD. If $T(F)$ has a non-closed path $p, +\Phi, -\Phi$, then there is a subBDD of F with root kernel $+\Phi$ through which this path goes: this BDD has the form $+\Phi \longrightarrow (-\Phi \longrightarrow F_2/F_1)/F_0$ with $F_2 \neq \mathbf{0}$, hence F is reducible by rule (2). Conversely, if F is reducible by this rule, then there is a subBDD of F of the form $+\Phi \longrightarrow (-\Phi \longrightarrow F_2/F_1)/F_0$, and $F_2 \neq \mathbf{0}$ since F is reduced by (1): it follows that there is a path p in F_2, and therefore that there is a path $p', +\Phi, -\Phi, p$ in F, which is not closed. □

2.3 Implementing TDDs

Assume that we have a function `BDDalloc` that takes a signed formula $s\Phi$ and two BDDs F_0 and F_1, and returns a shared node $s\Phi \longrightarrow F_1/F_0$. Then the following function (written in pseudo-Standard ML):

```
fun TDDnode (sΦ, F0, 0) = F0 (* reduction by (1) *)
  | TDDnode (sΦ, F0, F1) =
    if s = + andalso F1 = −Φ ⟶ F3/F2 for some F2, F3
       then TDDnode (sΦ, F0, F2) (* reduction by (2) *)
    else BDDalloc (sΦ, F0, F1)
```

builds the reduced form of the BDD $s\Phi \longrightarrow F_1/F_0$, when F_0 and F_1 are already reduced, and $s\Phi < s'\Phi'$ for any $s'\Phi'$ occurring in F_0 or F_1. Check also that it terminates. (In fact the only recursive call could be expanded by unfolding the definition once, and then the recursion disappears, since F_2 cannot have $-\Phi$ as topmost kernel.)

The tableau $\{\{s\Phi\}\}$ is built by:

```
fun TDDmake (sΦ) = BDDalloc (sΦ, 0, 1)
```

The following functions `TDDsubset1`, `TDDsubset0`, `TDDunion` and `TDDinter` were already defined by Minato.

The subset of paths $p \setminus \{s\Phi\}$ such that $p \in T(F)$ contains a given signed formula $s\Phi$ is given by `TDDsubset1` $(F, s\Phi)$, where `TDDsubset1` is defined by:

```
memofun TDDsubset1 (0, _) = 0
      | TDDsubset1 (1, _) = 0
      | TDDsubset1 (s'Φ' ⟶ F1/F0, sΦ) =
```

```
        if s'Φ' = sΦ
           then F1
        else if s'Φ' < sΦ
           then TDDnode (s'Φ', TDDsubset1 (F0, sΦ), TDDsubset1 (F1, sΦ))
        else 0
```

where the keyword `memofun` denotes the declaration of a *memoizing function*, that is, a function that keeps a private hash-table mapping previous arguments to the result of the computation on them. On each call to a memoizing function, the function first checks whether its private hash-table contains an entry for the supplied arguments; if so, it returns the previous result, and otherwise it computes the result, stores it into the hash-table, and returns it.

Similarly, the subset of paths in $T(F)$ that do not contain a given signed formula $s\Phi$ is given by `TDDsubset0` $(F, s\Phi)$, where `TDDsubset0` is defined by:

```
memofun TDDsubset0 (0, _) = 0
      | TDDsubset0 (1, _) = 1
      | TDDsubset0 (F as s'Φ' ⟶ F1/F0, sΦ) =
        if s'Φ' = sΦ
           then F0
        else if s'Φ' < sΦ
           then TDDnode (s'Φ', TDDsubset0 (F0, sΦ), TDDsubset0 (F1, sΦ))
        else F
```

Computing the union of two tableaux is done as follows:

```
memofun TDDunion (0, F') = F'
      | TDDunion (F, 0) = F
      | TDDunion (1, 1) = 1
      | TDDunion (1, s'Φ' ⟶ F1'/F0') =
        TDDnode (s'Φ', TDDunion (1, F0'), F1')
      | TDDunion (sΦ ⟶ F1/F0, 1) =
        TDDnode (sΦ, TDDunion (F0, 1), F1)
      | TDDunion (F as sΦ ⟶ F1/F0, F' as s'Φ' ⟶ F1'/F0') =
        if sΦ = s'Φ'
           then TDDnode (sΦ, TDDunion (F0, F0'), TDDunion (F1, F1'))
        else if sΦ < s'Φ'
           then TDDnode (sΦ, TDDunion (F0, F'), F1)
        else TDDnode (s'Φ', TDDunion (F, F0'), F1')
```

Computing the intersection of two tableaux is done as follows:

```
memofun TDDinter (0, F') = 0
      | TDDinter (F, 0) = 0
      | TDDinter (1, s'Φ' ⟶ F1'/F0') = TDDinter (1, F0')
      | TDDinter (sΦ ⟶ F1/F0, 1) = TDDinter (F0, 1)
      | TDDinter (F as sΦ ⟶ F1/F0, F' as s'Φ' ⟶ F1'/F0') =
        if sΦ = s'Φ'
           then TDDnode (sΦ, TDDinter (F0, F0'), TDDinter (F1, F1'))
        else if sΦ < s'Φ'
           then TDDinter (F0, F')
        else TDDinter (F, F0')
```

We shall also need to compute the *shuffle* $t \mathbin{@} t'$ of two tableaux, where $t \mathbin{@} t'$ is defined as $\{p \cup p' \mid p \in t, p' \in t'\}$. Computing the shuffle of $T(F)$ and $T(F')$ is intuitively done by the following function:

```
memofun TDDshuffle (0, F') = 0
      | TDDshuffle (F, 0) = 0
```

```
  | TDDshuffle (1, F') = F'
  | TDDshuffle (F, 1) = F
  | TDDshuffle (F as sΦ ⟶ F1/F0, F' as s'Φ' ⟶ F1'/F0') =
    if sΦ = s'Φ'
       then TDDnode (sΦ,TDDshuffle (F0, F0'),
                       TDDunion (TDDshuffle (F0, F1'),
                       TDDunion (TDDshuffle (F1, F0'),
                                 TDDshuffle (F1, F1'))))
    else if sΦ < s'Φ'
       then TDDnode (sΦ, TDDshuffle (F0, F'), TDDshuffle (F1, F'))
    else TDDnode (s'Φ', TDDshuffle (F, F0'), TDDshuffle (F, F1'))
```

However, observe that `TDDshuffle` only computes the shuffle of two decision diagrams when `TDDnode` builds zero-suppressed BDDs, i.e. when the additional closing reduction rule is not implemented. Otherwise, when given two TDDs in argument, `TDDshuffle` computes their shuffle *minus all resulting closed paths*.

Tableau calculi need another auxiliary function for replacing a signed formula $s\Phi$ in a TDD F by a TDD F'. More precisely, this operation replaces every path p in $T(F)$ that contains $s\Phi$ by a set of paths $(p \setminus \{s\Phi\}) \cup p'$, where p' ranges over the paths in $T(F')$. For example, consider the following tableau rule of classical propositional logic, where comma denotes union:

$$\frac{p, -(\Phi_1 \ \& \ \Phi_2)}{p, -\Phi_1 \quad | \quad p, -\Phi_2}$$

We can implement it on TDDs by replacing $-(\Phi_1 \ \& \ \Phi_2)$ in F by the tableau $\{\{-\Phi_1\}, \{-\Phi_2\}\}$, and this will effect it at once on all paths containing $-(\Phi_1 \ \& \ \Phi_2)$ in the whole tableau. This operation is defined as:

```
fun TDDreplace (F, sΦ, F') =
  TDDunion (TDDsubset0 (F, sΦ),
            TDDshuffle (TDDsubset1 (F, sΦ), F'))
```

Iterating through all paths in F, and applying a function f to each can be done in several different ways. Assuming that f returns a result, or raises an exception if it fails, then calling `TDDiter` $(f, F, [])$, where $[]$ is the empty list will either return the first result $f(p)$ where f does not fail, or raise an exception. The third argument to `TDDiter` is a representation of the path that will be submitted to f, represented as a list of signed formulas in decreasing $<$ order.

```
exception TDDITER (* declare an exception for failure *)
fun TDDiter (f, 1, p) = f(p)
  | TDDiter (f, 0, p) = raise TDDITER (* fail *)
  | TDDiter (f, (sΦ ⟶ F1/F0), p) =
    TDDiter (f, F0, p) handle _ => TDDiter (f, F1, sΦ :: p)
```

where `raise` raises an exception, e `handle _ =>` e' evaluates e, and if an exception is raised, evaluates e'; and $a::l$ builds the list whose first element is a and whose rest is l.

The `TDDiter` functional, as it is, sweeps through all paths from left to right in the TDD F. There are clearly many other, including more clever, possibilities.

Note that, in the case of TDDs with lots of paths but few nodes (hence highly shared), iterating through all paths destroys most of the interest of TDDs. In the sequel, the reader should keep in mind that applying `TDDiter` is likely to be costly, while other operations are not if TDDs remain of manageable size (which is likely).

3 Intuitionistic Logic

3.1 A Tableau System

Consider the tableau calculus of Figure 2. The rules have a numerator (above the bar) and possibly several denominators (below the bar), which are paths. Read as a set, the path p, p' denotes the union of p and p', and p, Φ where Φ is a signed formula denotes $p \cup \{\Phi\}$.

$$(Ax)\quad \frac{}{p, +\Phi, -\Phi}$$

$$(-\bot)\quad \frac{p, -\bot}{p} \qquad (+\bot)\quad \frac{p, +\bot}{}$$

$$(-\,\&\,)\quad \frac{p, -(\Phi_1\ \&\ \Phi_2)}{p, -\Phi_1 \ \mid\ p, -\Phi_2} \qquad (+\,\&\,)\quad \frac{p, +(\Phi_1\ \&\ \Phi_2)}{p, +\Phi_1, +\Phi_2}$$

$$(-\vee)\quad \frac{p, -(\Phi_1 \vee \Phi_2)}{p, -\Phi_1, -\Phi_2} \qquad (+\vee)\quad \frac{p, +(\Phi_1 \vee \Phi_2)}{p, +\Phi_1 \ \mid\ p, +\Phi_2}$$

$$(-\supset)\quad \frac{p, -(\Phi_1 \supset \Phi_2)}{p^+, +\Phi_1, -\Phi_2} \qquad (+\supset)\quad \frac{p, +(\Phi_1 \supset \Phi_2)}{p, +(\Phi_1 \supset \Phi_2), -\Phi_1 \ \mid\ p, +\Phi_2}$$

Figure 2: Tableau rules for intuitionistic logic

$$(Ax)\quad \frac{}{,\ ,\ \Phi \triangleright \Phi, \Delta}$$

$$(\bot R)\quad \frac{,\ \triangleright \Delta}{,\ \triangleright \bot, \Delta} \qquad (\bot L)\quad \frac{}{,\ ,\ \bot \triangleright \Delta}$$

$$(\&\,R)\quad \frac{,\ \triangleright \Phi_1, \Delta \qquad ,\ \triangleright \Phi_2, \Delta}{,\ \triangleright \Phi_1\ \&\ \Phi_2, \Delta} \qquad (\&\,L)\quad \frac{,\ ,\ \Phi_1, \Phi_2 \triangleright \Delta}{,\ ,\ \Phi_1\ \&\ \Phi_2 \triangleright \Delta}$$

$$(\vee R)\quad \frac{,\ \triangleright \Phi_1, \Phi_2, \Delta}{,\ \triangleright \Phi_1 \vee \Phi_2} \qquad (\vee L)\quad \frac{,\ ,\ \Phi_1 \triangleright \Delta \qquad ,\ ,\ \Phi_2 \triangleright \Delta}{,\ ,\ \Phi_1 \vee \Phi_2 \triangleright \Delta}$$

$$(\supset R)\quad \frac{,\ ,\ \Phi_1 \triangleright \Phi_2}{,\ \triangleright \Phi_1 \supset \Phi_2, \Delta} \qquad (\supset L)\quad \frac{,\ ,\ \Phi_1 \supset \Phi_2 \triangleright \Phi_1, \Delta \qquad ,\ ,\ \Phi_2 \triangleright \Delta}{,\ ,\ \Phi_1 \supset \Phi_2 \triangleright \Delta}$$

Figure 3: A sequent system for intuitionistic logic

Because sets are hard to work with, we shall consider paths as *multisets* of signed formulas and tableaux as multisets of paths. The comma will then denote multiset union instead of set union. In practice, the multiplicity of signed formulas in paths, or of paths in tableaux, will be irrelevant, and we shall erase them, i.e. we shall consider paths and tableaux as sets again in the implementation; this is because the rule of contraction will be admissible (this will be Lemma 9).

In each rule, the distinguished occurrences of formulas above the line are called the *principal formulas*; symmetrically, the distinguished occurrences of formulas below the line are called the *active formulas*. Moreover, for any path p, we write p^+ the set of formulas with sign $+$ in p.

Of these rules, only those concerned with implication are peculiar to intuitionistic logic. The $(+\supset)$ rule duplicates its principal formula $+(\Phi_1 \supset \Phi_2)$ on its left denominator, while $(-\supset)$ erases all non-principal negative formulas by the $p \mapsto p^+$ operation.

This system is a translation of the Gentzen system of Figure 3, where sequents $\Phi_1, \Phi_2, \ldots, \Phi_m \triangleright \Phi_{m+1}, \ldots, \Phi_{m+n}$ are translated to paths $+\Phi_1, +\Phi_2, \ldots, +\Phi_m, -\Phi_{m+1}, \ldots, -\Phi_{m+n}$, and the direction of inference rules is reversed. This Gentzen system is a variant of Dummett's system ([3], p.228), and is easily seen to be equivalent to it. See [4] for a discussion of various sequent systems for intuitionistic logic.

To prove a formula Φ, set up the initial tableau $\{\{-\Phi\}\}$, and apply the rules of Figure 2 to paths non-deterministically, until we get the empty tableau (then Φ is proved). If we cannot get the empty tableau, then Φ is unprovable. We shall also say that a path p or a tableau t is provable if and only if the empty tableau is reachable from $\{p\}$, resp. t.

We can reduce the amount of non-determinism by noticing that most of the rules are *invertible*: a rule is invertible if, whenever its numerator is provable, then all its denominators are provable as well. As noticed by Dyckhoff ([4], pp.801–802), all the rules but $(-\supset)$ are invertible.

Moreover, the search for a proof from the goal may go on forever, because the left denominator of the $(+\supset)$ rule contains its numerator. For instance, the search for a proof of $((A \supset B) \supset A) \supset A$ (Pierce's formula) goes on like this:

$$\dfrac{\dfrac{-((A \supset B) \supset A) \supset A}{+(A \supset B) \supset A, -A}}{+(A \supset B) \supset A, -A, -A \supset B \quad | \quad -A, +A}$$

which then goes on expanding $+(A \supset B) \supset A$ forever. (In fact, this formula is not provable in intuitionistic logic.) We can forbid this by making the search fail whenever it generates a path twice in the same sequence of expansions. This *loop-check* [6] is enough to ensure that the search for a proof terminates, since there are only finitely many possible paths — when paths are viewed as sets, whence the importance of showing that contraction is eliminable. (There are still at most $4^{|\Phi|}$ paths that can arise from considering all sets of signed subformulas of a given formula Φ — $2^{|\Phi|}$ by making a more careful analysis, and this is quite large.)

We would like to use a tableau expansion strategy like the following: because all rules but $(-\supset)$ are invertible, apply them eagerly on the current tableau until none is applicable, then try to apply $(-\supset)$ non-deterministically, and continue. However, because $(+\supset)$ may loop, we don't know whether we have to apply it once, twice or more before applying $(-\supset)$. Moreover, although applying any rule except $(-\supset)$ preserves provability, we are not sure that this draws us closer to an actual proof. For all these reasons, we also have to consider the combinatorics of proof permutabilities in addition to invertibility properties.

3.2 Invertibility, Permutabilities and Canonical Proofs

We develop the theory of permutabilities in our deduction system formally. This seems not to have been done at this level of detail before. To simplify matters, we first observe the following:

Lemma 4 *Call an instance of (Ax) on principal formula Φ* atomic *if and only if Φ is an atomic formula other than $\bot$.*

Then, every provable sequent has a proof in which all instances of (Ax) are atomic.

Proof: We claim that every sequent $, , \Phi \triangleright \Phi, \Delta$ has such a proof. We prove the claim by structural induction on Φ. If Φ is atomic and different from $\bot$, this is clear. This is also clear if Φ is $\bot$, since then $, , \Phi \triangleright \Phi, \Delta$ is an instance of $\bot L$. Otherwise, if Φ is an implication $\Phi_1 \supset \Phi_2$, by induction hypothesis there is a proof π_1 of $, , \Phi_1 \supset \Phi_2, \Phi_1 \triangleright \Phi_1, \Phi_2$ and a proof π_2 of $, , \Phi_1, \Phi_2 \triangleright \Phi_2$ with only axioms operating on atomic formulas, so:

$$\begin{array}{ll} & \begin{array}{cc} \pi_1 & \pi_2 \\ \vdots & \vdots \\ , , \Phi_1 \supset \Phi_2, \Phi_1 \triangleright \Phi_1, \Phi_2 & , , \Phi_1, \Phi_2 \triangleright \Phi_2 \end{array} \\ (\supset L) & \dfrac{\qquad\qquad\qquad\qquad\qquad\qquad}{, , \Phi_1 \supset \Phi_2, \Phi_1 \triangleright \Phi_2} \\ (\supset R) & \dfrac{\qquad\qquad\qquad\qquad\qquad\qquad}{, , \Phi_1 \supset \Phi_2 \triangleright \Phi_1 \supset \Phi_2, \Delta} \end{array}$$

is a proof as claimed. If Φ is a conjunction or a disjunction, the cases are similar.

Now, if $, \triangleright \Delta$ is a provable sequent, let π be some proof: the lemma follows by structural induction on π, using the previous claim when coming to instances of (Ax). □

Because of Lemma 4, we may without loss of generality assume that (Ax) is restricted so that its principal formula Φ is atomic and not $\bot$. We shall tacitly assume this in the rest of this section.

We also observe that we can define some quite benign transformations on proofs, that is, some transformations that do not alter in an essential way the skeleton of the proof. In particular, they do not increase the height of the proof, and do not introduce new instances of proof rules in the middle of the proof. Alternatively, we can see most of the following definitions as refinements of the fact that all rules in the intuitionistic calculus above, except $(\supset R)$, are invertible.

Definition 4 (π, Φ) *For any proof π of $, \triangleright \Delta$, and any formula Φ, let the* weakening on the right π, Φ *of π by Φ be the proof of $, \triangleright \Delta, \Phi$ defined by adding Φ consistently on the right of every sequent until we come up*

to an instance of (Ax) *or* $(\supset R)$*. More formally, if* π *is:*

$$(\supset R)\ \dfrac{\begin{array}{c}\pi'\\ \vdots \\ ,',\Phi_1 \triangleright \Phi_2\end{array}}{,' \triangleright \Phi_1 \supset \Phi_2, \Delta'}$$

then π, Φ *is:*

$$(\supset R)\ \dfrac{\begin{array}{c}\pi'\\ \vdots \\ ,',\Phi_1 \triangleright \Phi_2\end{array}}{,' \triangleright \Phi_1 \supset \Phi_2, \Delta', \Phi}$$

And if π *ends with any other rule* R*, i.e. it is of the form:*

$$(R)\ \dfrac{\begin{array}{ccc}\pi_1 & \ldots & \pi_n \\ \vdots & & \vdots \\ ,_1 \triangleright \Delta_1 & \ldots & ,_n \triangleright \Delta_n\end{array}}{,' \triangleright \Delta'}$$

then π, Φ *is:*

$$(R)\ \dfrac{\begin{array}{ccc}\pi_1, \Phi & \ldots & \pi_n, \Phi \\ \vdots & & \vdots \\ ,_1 \triangleright \Delta_1, \Phi & \ldots & ,_n \triangleright \Delta_n, \Phi\end{array}}{,' \triangleright \Delta', \Phi}$$

Similarly, we define the weakening on the left *of* π *by* Φ *as the proof* Φ, π *obtained by consistently adding* Φ *to the left of every sequent.*

Definition 5 $([\Phi_1 \supset]\pi)$ *If* π *is a proof of a sequent of the form* $, ,\Phi_1 \supset \Phi_2 \triangleright \Delta$*, then we let* $[\Phi_1 \supset]\pi$ *be the proof of* $, ,\Phi_2 \triangleright \Delta$ *defined as follows. If* π *is of the form:*

$$(\supset L)\ \dfrac{\begin{array}{cc}\pi_1 & \pi_2 \\ \vdots & \vdots \\ ,',\Phi_1 \supset \Phi_2 \triangleright \Phi_1, \Delta' & ,',\Phi_2 \triangleright \Delta'\end{array}}{,',\Phi_1 \supset \Phi_2 \triangleright \Delta'}$$

then $[\Phi_1 \supset]\pi$ *is just* π_2*, and if* π *is:*

$$(R)\ \dfrac{\begin{array}{ccc}\pi_1 & \ldots & \pi_n \\ \vdots & & \vdots \\ ,_1,\Phi_1 \supset \Phi_2 \triangleright \Delta_1 & \ldots & ,_n,\Phi_1 \supset \Phi_2 \triangleright \Delta_n\end{array}}{,',\Phi_1 \supset \Phi_2 \triangleright \Delta'}$$

where R *is not* $(\supset L)$ *or the principal formula is not* $\Phi_1 \supset \Phi_2$ *on the left, then* $[\Phi_1 \supset]\pi$ *is:*

$$(R)\ \dfrac{\begin{array}{ccc}[\Phi_1 \supset]\pi_1 & \ldots & [\Phi_1 \supset]\pi_n \\ \vdots & & \vdots \\ ,_1,\Phi_2 \triangleright \Delta_1 & \ldots & ,_n,\Phi_2 \triangleright \Delta_n\end{array}}{,',\Phi_2 \triangleright \Delta'}$$

Definition 6 ($[\Phi_1\vee]\pi$, $[\vee\Phi_2]\pi$) *If π is a proof of a sequent of the form $, , \Phi_1 \vee \Phi_2 \rhd \Delta$, then we let $[\Phi_1\vee]\pi$ (resp. $[\vee\Phi_2]\pi$) be the proof of $, , \Phi_2 \rhd \Delta$ (resp. $, , \Phi_1 \rhd \Delta$) defined as follows. If π is of the form:*

$$(\vee L)\ \dfrac{\begin{array}{cc}\pi_1 & \pi_2\\ \vdots & \vdots\\ , ', \Phi_1 \rhd \Delta' & , ', \Phi_2 \rhd \Delta'\end{array}}{, ', \Phi_1 \vee \Phi_2 \rhd \Delta'}$$

then $[\Phi_1\vee]\pi$ is just π_2 (resp. $[\vee\Phi_2]\pi$ is π_1), and if π is:

$$(R)\ \dfrac{\begin{array}{ccc}\pi_1 & \ldots & \pi_n\\ \vdots & & \vdots\\ , _1, \Phi_1 \vee \Phi_2 \rhd \Delta_1 & \ldots & , _n, \Phi_1 \vee \Phi_2 \rhd \Delta_n\end{array}}{, ', \Phi_1 \vee \Phi_2 \rhd \Delta'}$$

where R is not $(\vee L)$ or the principal formula is not $\Phi_1 \vee \Phi_2$ on the left, then $[\Phi_1\vee]\pi$ is:

$$(R)\ \dfrac{\begin{array}{ccc}[\Phi_1\vee]\pi_1 & \ldots & [\Phi_1\vee]\pi_n\\ \vdots & & \vdots\\ , _1, \Phi_2 \rhd \Delta_1 & \ldots & , _n, \Phi_2 \rhd \Delta_n\end{array}}{, ', \Phi_2 \rhd \Delta'}$$

and similarly for $[\vee\Phi_2]\pi$.

Definition 7 ($\pi[\Phi_1 \vee \Phi_2]$) *If π is a proof of a sequent of the form $, \rhd \Phi_1 \vee \Phi_2, \Delta$, then we let $\pi[\Phi_1 \vee \Phi_2]$ be the proof of $, \rhd \Phi_1, \Phi_2, \Delta$ defined as follows. If π is of the form:*

$$(\vee R)\ \dfrac{\begin{array}{c}\pi_1\\ \vdots\\ , ' \rhd \Phi_1, \Phi_2, \Delta'\end{array}}{, \rhd \Phi_1 \vee \Phi_2, \Delta'}$$

then $\pi[\Phi_1 \vee \Phi_2]$ is π_1; if π is of the form:

$$(\supset R)\ \dfrac{\begin{array}{c}\pi_1\\ \vdots\\ , ', \Phi_3 \rhd \Phi_4\end{array}}{, ' \rhd \Phi_3 \supset \Phi_4, \Phi_1 \vee \Phi_2, \Delta'}$$

then $\pi[\Phi_1 \vee \Phi_2]$ is:

$$(\supset R)\ \dfrac{\begin{array}{c}\pi_1\\ \vdots\\ , ', \Phi_3 \rhd \Phi_4\end{array}}{, ' \rhd \Phi_3 \supset \Phi_4, \Phi_1, \Phi_2, \Delta'}$$

and if π is:

$$(R)\ \dfrac{\begin{array}{ccc}\pi_1 & \ldots & \pi_n\\ \vdots & & \vdots\\ , _1 \rhd \Phi_1 \vee \Phi_2, \Delta_1 & \ldots & , _n \rhd \Phi_1 \vee \Phi_2, \Delta_n\end{array}}{, ' \rhd \Phi_1 \vee \Phi_2, \Delta'}$$

where R is not $(\supset R)$, and R is not $(\vee R)$ unless the principal formula is not $\Phi_1 \vee \Phi_2$ on the right, then $\pi[\Phi_1 \vee \Phi_2]$ is:

$$(R)\ \dfrac{\begin{array}{ccc}\pi_1[\Phi_1 \vee \Phi_2] & \ldots & \pi_n[\Phi_1 \vee \Phi_2]\\ \vdots & & \vdots\\ , _1 \rhd \Phi_1, \Phi_2, \Delta_1 & \ldots & , _n \rhd \Phi_1, \Phi_2, \Delta_n\end{array}}{, ' \rhd \Phi_1, \Phi_2, \Delta'}$$

Definition 8 ($[\Phi_1 \ \& \ \Phi_2]\pi$) *If π is a proof of a sequent of the form $, , \Phi_1 \ \& \ \Phi_2 \rhd \Delta$, then we let $[\Phi_1 \ \& \ \Phi_2]\pi$ be the proof of $, , \Phi_1, \Phi_2 \rhd \Delta$ defined as follows. If π is of the form:*

$$(\ \& \ L)\ \dfrac{\begin{array}{c}\pi_1\\ \vdots \\ , \ ', \Phi_1, \Phi_2 \rhd \Delta'\end{array}}{, \ ', \Phi_1 \ \& \ \Phi_2 \rhd \Delta'}$$

then $[\Phi_1 \ \& \ \Phi_2]\pi$ is just π_1, and if π is:

$$(R)\ \dfrac{\begin{array}{ccc}\pi_1 & \ldots & \pi_n\\ \vdots & & \vdots \\ , \ _1, \Phi_1 \ \& \ \Phi_2 \rhd \Delta_1 & \ldots & , \ _n, \Phi_1 \ \& \ \Phi_2 \rhd \Delta_n\end{array}}{, \ ', \Phi_1 \ \& \ \Phi_2 \rhd \Delta'}$$

where R is not $(\ \& \ L)$ or the principal formula is not $\Phi_1 \ \& \ \Phi_2$ on the left, then $[\Phi_1 \ \& \ \Phi_2]\pi$ is:

$$(R)\ \dfrac{\begin{array}{ccc}[\Phi_1 \ \& \ \Phi_2]\pi_1 & \ldots & [\Phi_1 \ \& \ \Phi_2]\pi_n\\ \vdots & & \vdots \\ , \ _1, \Phi_1, \Phi_2 \rhd \Delta_1 & \ldots & , \ _n, \Phi_1, \Phi_2 \rhd \Delta_n\end{array}}{, \ ', \Phi_1, \Phi_2 \rhd \Delta'}$$

Definition 9 ($\pi[\Phi_1 \ \&\ ]$, $\pi[\ \& \ \Phi_2]$) *If π is a proof of a sequent of the form $, \rhd \Phi_1 \ \& \ \Phi_2, \Delta$, then we let $\pi[\Phi_1 \ \&\ ]$ (resp. $\pi[\ \& \ \Phi_2]$) be the proof of $, \rhd \Phi_2, \Delta$ (resp. $, \rhd \Phi_1, \Delta$) defined as follows. If π is of the form:*

$$(\ \& \ R)\ \dfrac{\begin{array}{cc}\pi_1 & \pi_2\\ \vdots & \vdots \\ , \ ' \rhd \Phi_1, \Delta' & , \ ' \rhd \Phi_2, \Delta'\end{array}}{, \ ' \rhd \Phi_1 \ \& \ \Phi_2, \Delta'}$$

then $\pi[\Phi_1 \ \&\ ]$ is π_2 and $\pi[\ \& \ \Phi_2]$ is π_1; if π is of the form:

$$(\supset R)\ \dfrac{\begin{array}{c}\pi_1\\ \vdots \\ , \ ', \Phi_3 \rhd \Phi_4\end{array}}{, \ ' \rhd \Phi_3 \supset \Phi_4, \Phi_1 \ \& \ \Phi_2, \Delta'}$$

then $\pi[\Phi_1 \ \&\ ]$ is:

$$(\supset R)\ \dfrac{\begin{array}{c}\pi_1\\ \vdots \\ , \ ', \Phi_3 \rhd \Phi_4\end{array}}{, \ \rhd \Phi_3 \supset \Phi_4, \Phi_2, \Delta'}$$

and similarly for $\pi[\ \& \ \Phi_2]$; and if π is:

$$(R)\ \dfrac{\begin{array}{ccc}\pi_1 & \ldots & \pi_n\\ \vdots & & \vdots \\ , \ _1 \rhd \Phi_1 \ \& \ \Phi_2, \Delta_1 & \ldots & , \ _n \rhd \Phi_1 \ \& \ \Phi_2, \Delta_n\end{array}}{, \ ' \rhd \Phi_1 \ \& \ \Phi_2, \Delta'}$$

where R is not $(\supset R)$, and R is not $(\ \& \ R)$ unless the principal formula is not $\Phi_1 \ \& \ \Phi_2$ on the right, then $\pi[\Phi_1 \ \&\ ]$ is:

$$(R)\ \dfrac{\begin{array}{ccc}\pi_1[\Phi_1 \ \&\ ] & \ldots & \pi_n[\Phi_1 \ \&\ ]\\ \vdots & & \vdots \\ , \ _1 \rhd \Phi_2, \Delta_1 & \ldots & , \ _n \rhd \Phi_2, \Delta_n\end{array}}{, \ ' \rhd \Phi_2, \Delta'}$$

and similarly for $\pi[\,\&\,\Phi_2]$.

Definition 10 ($\pi[\bot]$) *If* π *is a proof of a sequent of the form* $, \rhd \bot, \Delta$, *then we let* $\pi[\bot]$ *be the proof of* $, \rhd \Delta$ *defined as follows. If* π *is of the form:*

$$(\bot R)\ \dfrac{\begin{array}{c}\pi_1\\ \vdots\\ , ' \rhd \Delta'\end{array}}{, ' \rhd \bot, \Delta'}$$

then $\pi[\bot]$ *is* π_1*; if* π *is of the form:*

$$(\supset R)\ \dfrac{\begin{array}{c}\pi_1\\ \vdots\\ , ', \Phi_1 \rhd \Phi_2\end{array}}{, ' \rhd \Phi_1 \supset \Phi_2, \bot, \Delta'}$$

then $\pi[\bot]$ *is:*

$$(\supset R)\ \dfrac{\begin{array}{c}\pi_1\\ \vdots\\ , ', \Phi_1 \rhd \Phi_2\end{array}}{, \rhd \Phi_1 \supset \Phi_2, \Delta'}$$

and if π *is:*

$$(R)\ \dfrac{\begin{array}{ccc}\pi_1 & \ldots & \pi_n\\ \vdots & & \vdots\\ , _1 \rhd \bot, \Delta_1 & \ldots & , _n \rhd \bot, \Delta_n\end{array}}{, ' \rhd \bot, \Delta'}$$

where R *is not* $(\supset R)$, *and* R *is not* $(\bot R)$ *unless the principal formula is not* $\bot$ *on the right, then* $\pi[\bot]$ *is:*

$$(R)\ \dfrac{\begin{array}{ccc}\pi_1[\bot] & \ldots & \pi_n[\bot]\\ \vdots & & \vdots\\ , _1 \rhd \Delta_1 & \ldots & , _n \rhd \Delta_n\end{array}}{, ' \rhd \Delta'}$$

Definition 11 *Any of the transformations of Definitions 4, 5, 6, 7, 8, 9, 10 is called an* elementary benign *transformation.*

A benign transformation *is any composition of elementary benign transformations.*

Observe that $[\Phi_1 \supset]\pi$, $[\Phi_1 \vee]\pi$, $[\vee \Phi_2]\pi$, $\pi[\Phi_1 \vee \Phi_2]$, etc. are well-defined only because π was assumed to contain only atomic instances of (Ax). Otherwise, doing these transformations might well turn an instance of (Ax) into something else.

One reason why these transformations are benign is the following:

Definition 12 *The* size $|\pi|$ *of a proof* π *is the number of occurrences of rule instances in it. More formally, the size of an instance of* (Ax) *or* $(\bot L)$ *is 1, and the size of a proof:*

$$(R)\ \dfrac{\begin{array}{ccc}\pi_1 & \ldots & \pi_n\\ \vdots & & \vdots\\ , _1 \rhd \Delta_1 & \ldots & , _n \rhd \Delta_n\end{array}}{, \rhd \Delta}$$

is $|\pi_1| + \ldots + |\pi_n| + 1$.

R_1 \ R_2		$\supset L$ $(+\supset)$	$\supset R$ $(-\supset)$	$\vee L$ $(+\vee)$	$\vee R$ $(-\vee)$	$\&\, L$ $(+\&)$	$\&\, R$ $(-\&)$	$\bot L$ $(+\bot)$	$\bot R$ $(-\bot)$
$\supset L$	$(+\supset)$	p	np^*	p	p	p	p	$\times$	p
$\supset R$	$(-\supset)$	np	$\times$	np	c	p	c	$\times$	c
$\vee L$	$(+\vee)$	p	p	p	p	p	p	$\times$	p
$\vee R$	$(-\vee)$	p	$\times$	p	p	p	p	$\times$	p
$\&\, L$	$(+\&)$	p	p	p	p	p	p	$\times$	p
$\&\, R$	$(-\&)$	p	$\times$	p	p	p	p	$\times$	p
$\bot L$	$(+\bot)$	c	c	c	c	c	c	$\times$	c
$\bot R$	$(-\bot)$	p	$\times$	p	p	p	p	$\times$	p

Figure 4: Permutabilities in the intuitionistic calculus

Lemma 5 *Let π be a proof and π' be a benign transformation of π. Then $|\pi'| \leq |\pi|$.*

Proof: This holds for the empty benign transformation ($\pi' = \pi$) and for every elementary benign transformation, as shown by an easy structural induction on the proof π. The lemma then follows by induction on the number of elementary benign transformations that compose the given transformation, $\Box$

Following Kleene [13], in spirit at least, we say that a sequent rule R_1 *permutes under* a rule R_2, or that R_2 *permutes over* R_1, if and only if, whenever we have a proof of the form:

$$
\begin{array}{cc}
 & \begin{array}{ccccc} \pi_1 & & \ldots & & \pi_n \\ \vdots & & & & \vdots \end{array} \quad \begin{array}{ccccc} & & & & \\ \pi'_2 & & \ldots & & \pi'_m \\ \vdots & & & & \vdots \end{array} \\
(R_1) & \dfrac{,_1 \triangleright \Delta_1 \quad \ldots \quad ,_n \triangleright \Delta_n}{,'_1 \triangleright \Delta'_1} \qquad \dfrac{,'_2 \triangleright \Delta'_2 \quad \ldots \quad ,'_m \triangleright \Delta'_m}{, \triangleright \Delta} \\
(R_2) &
\end{array}
$$

where the active formulas of R_2 are not the principal formulas of R_1, then we can transform it to a proof of the same sequent with R_1 below R_2, i.e. to a proof of the form:

$$
\begin{array}{cc}
 & \begin{array}{ccccc} \vdots & & & & \vdots \end{array} \\
(R_1) & \dfrac{,''_1 \triangleright \Delta''_1 \quad \ldots \quad ,''_n \triangleright \Delta''_n}{, \triangleright \Delta}
\end{array}
$$

where each $,''_i \triangleright \Delta''_i$, $1 \leq i \leq n$, is either the conclusion of an instance of R_2 whose premises are among $\pi_1, \ldots, \pi_n, \pi'_2, \ldots, \pi'_m$ or their benign transformations, or is directly the conclusion of one of the latter subproofs, or a benign transformation of one of these proofs.

When all the $,''_i \triangleright \Delta''_i$, $1 \leq i \leq n$, are conclusions of $\pi_1, \ldots, \pi_n, \pi'_2, \ldots, \pi'_m$, or of benign transformations, i.e. when R_2 simply disappears during the permutation, we say that R_1 *cancels* R_2.

Then, the table of permutabilities for the system of Figure 3 is given in Figure 4. Entries in row R_1 and column R_2 are to be read as follows: p means that R_1 permutes under R_2, np means that R_1 does not permute under R_2, c means that R_1 cancels R_2, and $\times$ means that R_1 is never in permutation position over R_2. Entries marked np^* are special cases that have some nice properties not directly apparent from permutabilities. The proof is given in Appendix A.

The important observation is that we may read off permutabilities from the table and deduce efficient strategies for proof-search in the proof system, as noticed by Galmiche and Perrier [7]. In this paper, we consider only tableaux-style proof search, that is, proof search that proceeds backwards from the goal to the axioms.

Then, any rule R whose row does not contain np can be freely permuted down in the sequent proof (up in the tableau proof), which is to say that we may apply it eagerly. This is the case with all non-implicational rules, so we shall expand every non-atomic formula that is not an implication as soon as we can.

Rule $(\supset L)$ can also be permuted down, except that it cannot be permuted past $(\supset R)$. However, whenever we have a proof of the form:

$$\dfrac{\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ , , \Phi_1 \supset \Phi_2, \Phi_3 \triangleright \Phi_1, \Phi_4\end{array} \quad \begin{array}{c}\pi_2\\ \vdots\\ , , \Phi_2, \Phi_3 \triangleright \Phi_4\end{array}}{, , \Phi_1 \supset \Phi_2, \Phi_3 \triangleright \Phi_4}\,(\supset L)}{, , \Phi_1 \supset \Phi_2 \triangleright \Phi_3 \supset \Phi_4, \Delta}\,(\supset R)$$

we can complicate the proof by adding an instance of $(\supset L)$ on the same principal formula $\Phi_1 \supset \Phi_2$ on the left *below* the $(\supset R)$ instance:

$$\dfrac{\dfrac{\dfrac{\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ , , \Phi_1 \supset \Phi_2, \Phi_3 \triangleright \Phi_1, \Phi_4\end{array} \quad \begin{array}{c}\pi_2\\ \vdots\\ , , \Phi_2, \Phi_3 \triangleright \Phi_4\end{array}}{, , \Phi_1 \supset \Phi_2, \Phi_3 \triangleright \Phi_4}\,(\supset L)}{, , \Phi_1 \supset \Phi_2 \triangleright \Phi_1, \Phi_3 \supset \Phi_4, \Delta}\,(\supset R) \quad \dfrac{\begin{array}{c}\pi_2\\ \vdots\\ , , \Phi_2, \Phi_3 \triangleright \Phi_4\end{array}}{, , \Phi_2 \triangleright \Phi_3 \supset \Phi_4, \Delta}\,(\supset R)}{, , \Phi_1 \supset \Phi_2 \triangleright \Phi_3 \supset \Phi_4, \Delta}\,(\supset L)$$

Notice that this also duplicates the subproof π_2, and demands that we now apply $(\supset R)$ twice as well. While this would be detrimental to a tableau theorem prover that expands paths sequentially, this will be less so in our method, where we may expand several paths at the same time. Notice also that we could have just added the new instance of $(\supset L)$ below $(\supset R)$ simply because $(\supset L)$ is invertible.

The other difficulty with this is that there is no upper bound on the number of times that we might duplicate the instance of $(\supset L)$ from above to below the instance of $(\supset R)$. I.e., we might want to apply $(\supset L)$ eagerly, but this leads to non-termination. In fact, in a sequence of expansions by rules other than $(\supset R)$ (i.e., $(- \supset)$), we never need to expand the same left implication twice. Indeed, otherwise, we might permute them with other rules so that they come next to each other, producing the following configuration:

$$\dfrac{\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ , , \Phi_1 \supset \Phi_2 \triangleright \Phi_1, \Phi_1, \Delta\end{array} \quad \begin{array}{c}\pi_2\\ \vdots\\ , , \Phi_2 \triangleright \Phi_1, \Delta\end{array}}{, , \Phi_1 \supset \Phi_2 \triangleright \Phi_1, \Delta}\,(\supset L) \quad \begin{array}{c}\pi_3\\ \vdots\\ , , \Phi_2 \triangleright \Delta\end{array}}{, , \Phi_1 \supset \Phi_2 \triangleright \Delta}\,(\supset L)$$

But then, look at the subproof π_1: its end sequent is $, , \Phi_1 \supset \Phi_2 \triangleright \Phi_1, \Phi_1, \Delta$. Since contraction is admissible in this system, we may transform π_1 into a proof of $, , \Phi_1 \supset \Phi_2 \triangleright \Phi_1, \Delta$. We only have to check that eliminating contraction does not reintroduce another instance of $(\supset L)$ above $(\supset R)$, or anything more complicated. In fact, we have to show that we can eliminate contraction from canonical proofs, yielding yet other canonical proofs.

It is now time to define an appropriate notion of canonical proofs, taking into account the permutabilities of Figure 4 and the above remarks on $(\supset L)$ and $(\supset R)$. (Our definition is probably not as general as we could make it, but it is simple and sufficient.)

Definition 13 *Let $\succ$ be the smallest transitive relation such that, for every sign s in $+, -$, for every formulas Φ, Φ_1, Φ_2:*

- $s(\Phi_1 \ \& \ \Phi_2) \succ s\Phi_1$, $s(\Phi_1 \ \& \ \Phi_2) \succ s\Phi_2$,
- $s(\Phi_1 \vee \Phi_2) \succ s\Phi_1$, $s(\Phi_1 \vee \Phi_2) \succ s\Phi_2$,
- $+(\Phi_1 \supset \Phi_2) \succ +\Phi_2$, $+(\Phi_1 \supset \Phi_2) \succ -\Phi_1$,
- $s\Phi \succ -(\Phi_1 \supset \Phi_2)$, *provided that $s\Phi$ is not a negative implication.*

Observe that $\succ$ is a strict ordering. Let $>$ be any total strict ordering extending $\succ$.

For any sequent $, \triangleright \Delta$, write $+, $ for the multiset of formulas $+\Phi$ with $\Phi \in ,$ and $-\Phi$ with $\Phi \in \Delta$. Denote by $p_1 \setminus p_2$ the multiset difference of p_1 and p_2.

We say that a proof π of $, \triangleright \Delta$ in the system of Figure 3 is $>$-canonical *if and only if:*

- *either $, \triangleright \Delta$ is an instance of (Ax);*
- *or π ends in an instance of $(\supset R)$:*

$$(\supset R)\ \frac{\begin{array}{c}\pi_1\\ \vdots \\ , , \Phi_1 \triangleright \Phi_2\end{array}}{, \triangleright \Phi_1 \supset \Phi_2, \Delta}$$

  and π_1 is canonical.

- *or π ends in an instance of some other rule R, and is of the form:*

$$(R)\ \frac{\begin{array}{ccc}\pi_1 & \ldots & \pi_n\\ \vdots & & \vdots \\ , _1 \triangleright \Delta_1 & \ldots & , _n \triangleright \Delta_n\end{array}}{, \triangleright \Delta}$$

  where $s\Phi > s_1\Phi_1$, ..., $s\Phi > s_n\Phi_n$, where $s\Phi$ is the principal formula in $, \triangleright \Delta$, $s_1\Phi_1$ is the principal formula in $, _1 \triangleright \Delta_1$, ..., $s_n\Phi_n$ is the principal formula in $, _n \triangleright \Delta_n$, and $\pi_1, \ldots, \pi_n$ are canonical proofs.

This definition in particular formalizes the fact that we don't wish to expand a left implication twice in a row, because $+(\Phi_1 \supset \Phi_2) \not\succ +(\Phi_1 \supset \Phi_2)$. This definition also forces a particular class of expansion strategies, through the use of the ordering $\succ$. Because of permutabilities, we wish that $\succ$ has negative implications $-\Phi_1 \supset \Phi_2$ less than all other signed formulas, i.e., we expand negative implications as late as possible.

Then, contraction on the right is eliminable, but we can actually say a bit more.

Lemma 6 *If $, \triangleright \Phi, \Phi, \Delta$ has a $>$-canonical proof, then $, \triangleright \Phi, \Delta$ has a $>$-canonical proof of no greater size, and with the same final principal formula.*

Proof: By induction on the depth of the proof π of $, \triangleright \Phi, \Phi, \Delta$. If π is an instance of (Ax) or of $(\supset R)$, then this is obvious.

Otherwise, if π ends in another rule R, whose principal formula is not the duplicated formula $-\Phi$, then contraction just permutes up in the sequent proof and we apply the induction hypothesis.

Finally, if π ends in some non-(Ax) non-$(\supset R)$ rule R whose principal formula is the duplicated formula $-\Phi$, then again contraction permutes over R, and we apply the induction hypothesis. □

Benign transformations are also benign in that they preserve canonicity:

Lemma 7 *Let π be a $>$-canonical proof, and π' be a benign transformation of π. Then π' is $>$-canonical.*

Proof: By induction on the number of elementary benign transformations from π to π', it remains to show that this holds when π' is the result of an elementary benign transformation of π.

In any case, we claim the stronger result that π' is a $>$-canonical proof whose final principal formula is no $>$-greater than the final principal formula of π.

If $\pi' = \pi, \Phi$ for some formula Φ, this is clear. In fact, in this case the final principal formula of π' is the same as that of π.

If π is a proof of $, , \Phi_1 \supset \Phi_2 \triangleright \Delta$ and $\pi' = [\Phi_1 \supset]\pi$, then the induction proceeds as follows. If π ends in an instance of $(\supset L)$ with principal formula $+\Phi_1 \supset \Phi_2$, with left subproof π_1 and right subproof π_2 (see Definition 5), then $\pi' = \pi_2$. Because π is $>$-canonical, π_2 is $>$-canonical and its final principal formula is $<$ than $+\Phi_1 \supset \Phi_2$, so the claim is proved. If π ends in some other rule or in $(\supset L)$ on some other principal formula, then the claim follows from the induction hypothesis applied to the premises.

When π' is $[\Phi_1 \vee]\pi$, or $[\vee \Phi_2]\pi$, or any other elementary benign transformation of π, the same kind of argument applies. $\Box$

It follows:

Theorem 8 *Every provable sequent has a $>$-canonical proof.*

Proof: Let π be a proof where all instances of (Ax) are atomic: we prove the result by induction on the number of subproofs π' of π that do not obey the conditions of Definition 13. If this number is 0, then we are done. Otherwise, consider some π' that is highest in π. We claim that we can transform π' into a $>$-canonical proof of the same sequent such that its end sequent has a principal formula that is not $>$-greater than the principal formula of the end sequent of π'. The lemma then follows from the claim.

Let's prove the claim. π' is a proof of the form, as above:

$$(R)\ \frac{\begin{array}{ccc}\pi_1 & \dots & \pi_n \\ \vdots & & \vdots \\ , _1 \triangleright \Delta_1 & \dots & , _n \triangleright \Delta_n\end{array}}{, \triangleright \Delta}$$

where R is neither (Ax) nor $(\supset R)$; $\pi_1, \dots, \pi_n$ are $>$-canonical proofs (since π' is highest); and letting $s\Phi$ be the principal formula in $, \triangleright \Delta$, $s_1\Phi_1$ be the principal formula in $, _1 \triangleright \Delta_1$, $\dots$, $s_n\Phi_n$ be the principal formula in $, _n \triangleright \Delta_n$, then $s\Phi \not> s_1\Phi_1$ or $\dots$ or $s\Phi \not> s_n\Phi_n$. Assume without loss of generality that $s\Phi \not> s_1\Phi_1$.

If R is not $(\supset L)$, then all the active formulas in R are strictly less than $s\Phi$, by definition of $\succ$, so $s_1\Phi_1$ is not the active formula. Therefore, if R_1 is the last rule applied in π_1 above R, R_1 and R are in permutation position. If R_1 is (Ax), then Φ_1 is in both $, _1$ and Δ_1, and therefore is also both in $,$ and Δ: we can replace π' by an instance of (Ax) with $+\Phi_1$ and $-\Phi_1$ as principal formulas. Otherwise, R_1 cannot be $(\supset R)$, because otherwise its principal formula $s_1\Phi_1$ (a negative implication) would be less than $s\Phi$ (which is not a negative implication, since R is not $(\supset R)$) by definition of $\succ$. It follows that R_1 permutes under R (see Figure 4). For instance, if R is $(\& R)$ and R_1 is $(\supset L)$ on the left premise of R, then π' has the form:

$$\begin{array}{c}(\supset L) \\ (\& R)\end{array}\ \frac{\dfrac{\begin{array}{ccc}\pi'_1 & & \pi'_2 \\ \vdots & & \vdots \\ , ', \Phi_1 \supset \Phi_2 \triangleright \Phi_1, \Phi_3, \Delta' & & , ', \Phi_2 \triangleright \Phi_3, \Delta'\end{array}}{, ', \Phi_1 \supset \Phi_2 \triangleright \Phi_3, \Delta'} \qquad \begin{array}{c}\pi'_3 \\ \vdots \\ , ', \Phi_1 \supset \Phi_2 \triangleright \Phi_4, \Delta'\end{array}}{, ', \Phi_1 \supset \Phi_2 \triangleright \Phi_3 \ \& \ \Phi_4, \Delta'}$$

where by assumption $-\Phi_3 \ \& \ \Phi_4 \not> +\Phi_1 \supset \Phi_2$ — hence $-\Phi_3 \ \& \ \Phi_4 \leq +\Phi_1 \supset \Phi_2$ by totality of $>$, so we transform it into:

$$\begin{array}{c}(\& R) \\ (\supset L)\end{array}\ \frac{\dfrac{\begin{array}{cc}\pi'_1 & \pi'_3, \Phi_1 \\ \vdots & \vdots \\ , ', \Phi_1 \supset \Phi_2 & , ', \Phi_1 \supset \Phi_2 \\ \triangleright \Phi_1, \Phi_3, \Delta' & \triangleright \Phi_1, \Phi_4, \Delta'\end{array}}{, ', \Phi_1 \supset \Phi_2 \triangleright \Phi_1, \Phi_3 \ \& \ \Phi_4, \Delta'} \ (\& R) \quad \dfrac{\begin{array}{cc}\pi'_2 & [\Phi_1 \supset]\pi'_3 \\ \vdots & \vdots \\ , ', \Phi_2 \triangleright \Phi_3, \Delta' & , ', \Phi_2 \triangleright \Phi_4, \Delta'\end{array}}{, ', \Phi_2 \triangleright \Phi_3 \ \& \ \Phi_4, \Delta'}}{, ', \Phi_1 \supset \Phi_2 \triangleright \Phi_3 \ \& \ \Phi_4, \Delta'}$$

Call π'' the latter proof, π''_1 the subproof of π'' that ends in the left-hand application of $(\& R)$, and π''_2 that which ends in the right-hand application of $(\& R)$. Observe that $|\pi'| = |\pi'_1| + |\pi'_2| + |\pi'_3| + 2$. Observe also that $|\pi''_1| = |\pi'_1| + |\pi'_3, \Phi_1| + 1 \leq |\pi'_1| + |\pi'_3| + 1$ (by Lemma 5) $< |\pi'|$, and $|\pi''_2| = |\pi'_2| + |[\Phi_1 \supset]\pi'_3| + 1 \leq |\pi'_2| + |\pi'_3| + 1$ (by Lemma 5) $< |\pi'|$. Observe also that π'_1 and π'_2 are $>$-canonical, and that π'_3, Φ_1 and $[\Phi_1 \supset]\pi'_3$ as well by Lemma 7. So we can apply the induction hypothesis on π''_1 and π''_2, getting new proofs π'''_1 and π'''_2, so that π'' transforms into:

$$(\supset L)\ \frac{\begin{array}{cc}\pi''_1 & \pi''_3 \\ \vdots & \vdots \\ , ', \Phi_1 \supset \Phi_2 \triangleright \Phi_1, \Phi_3 \ \& \ \Phi_4, \Delta' & , ', \Phi_2 \triangleright \Phi_3 \ \& \ \Phi_4, \Delta'\end{array}}{, ', \Phi_1 \supset \Phi_2 \triangleright \Phi_3 \ \& \ \Phi_4, \Delta'}$$

where the principal formula in the end-sequent of π_1'' (resp. π_2'') is no greater than $-\Phi_3 \ \& \ \Phi_4$ — hence is $\leq$ than $+\Phi_1 \supset \Phi_2$. That is, the latter subproof is $>$-canonical. All other cases work similarly to this one, or in simpler ways.

If R is $(\supset L)$, then the only difference with the previous case is that $s_1\Phi_1$ may still be active, although $s\Phi = +(\Phi_1 \supset \Phi_2) \not> s_1\Phi_1$: this happens exactly when $s_1\Phi_1 = +(\Phi_1 \supset \Phi_2)$, and π' is:

$$\dfrac{\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ , , \Phi_1 \supset \Phi_2 \triangleright \Phi_1, \Phi_1, \Delta\end{array} \quad \begin{array}{c}\pi_2\\ \vdots\\ , , \Phi_2 \triangleright \Phi_1, \Delta\end{array}}{, , \Phi_1 \supset \Phi_2 \triangleright \Phi_1, \Delta}(\supset L) \quad \begin{array}{c}\pi_3\\ \vdots\\ , , \Phi_2 \triangleright \Delta\end{array}}{, , \Phi_1 \supset \Phi_2 \triangleright \Delta}(\supset L)$$

But then, looking at π_1 and using Lemma 6, we get a $>$-canonical proof π_1' of $, , \Phi_1 \supset \Phi_2 \triangleright \Phi_1, \Delta$ of no greater size and ending in the same final principal formula as π_1. Because π_1' is $>$-canonical, the latter formula is then $<$ than $+\Phi_1 \supset \Phi_2$. Hence the proof:

$$\dfrac{\begin{array}{c}\pi_1'\\ \vdots\\ , , \Phi_1 \supset \Phi_2 \triangleright \Phi_1, \Phi_1, \Delta\end{array} \quad \begin{array}{c}\pi_3\\ \vdots\\ , , \Phi_2 \triangleright \Delta\end{array}}{, , \Phi_1 \supset \Phi_2 \triangleright \Delta}(\supset L)$$

is a $>$-canonical proof, as desired. □

Then, contraction is an admissible rule at the end of canonical proofs. This completes Lemma 6.

Lemma 9 *If $, , \Phi, \Phi \triangleright \Delta$ has a canonical proof, then $, , \Phi \triangleright \Delta$ has a canonical proof. And similarly for $, \triangleright \Phi, \Phi, \Delta$ and $, \triangleright \Phi, \Delta$.*

Proof: The "similar" case is just Lemma 6, so we just deal with contraction on the left.

We prove the more general statement that there is such a canonical proof of no greater size than the original proof and ending in the same principal formula. This is proved by induction on the size of the proof π of $, , \Phi, \Phi \triangleright \Delta$. If π is an instance of (Ax), this is obvious.

Otherwise, if π ends in another rule R, whose principal formula is not the duplicated formula $+\Phi$, then the result is obvious: contraction just permutes up in the sequent proof and we apply the induction hypothesis.

Finally, if π ends in some non-(Ax) rule R whose principal formula is the duplicated formula $+\Phi$, then we have the following cases. If R is any other rule other than $(\supset L)$ or (Ax), then this appeals directly to the induction hypothesis (contraction permutes over R).

The only non-trivial case is when π ends in an instance of $(\supset L)$, where $+\Phi$ is the duplicated formula, i.e. π is:

$$\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ , , \Phi_1 \supset \Phi_2, \Phi_1 \supset \Phi_2 \triangleright \Phi_1, \Delta\end{array} \quad \begin{array}{c}\pi_2\\ \vdots\\ , , \Phi_1 \supset \Phi_2, \Phi_2 \triangleright \Delta\end{array}}{, , \Phi_1 \supset \Phi_2, \Phi_1 \supset \Phi_2 \triangleright \Delta}(\supset L)$$

But by induction hypothesis, there is a $>$-canonical proof π_1' of $, , \Phi_1 \supset \Phi_2 \triangleright \Phi_1, \Delta$ of no greater size and having the same final principal formula as π_1. Similarly, $[\Phi_1 \supset]\pi_2$ can be transformed into a $>$-canonical proof of $, , \Phi_2 \triangleright \Delta$ of no greater size and having the same final principal formula. Therefore:

$$\dfrac{\begin{array}{c}\pi_1'\\ \vdots\\ , , \Phi_1 \supset \Phi_2 \triangleright \Phi_1, \Delta\end{array} \quad \begin{array}{c}\pi_2'\\ \vdots\\ , , \Phi_2 \triangleright \Delta\end{array}}{, , \Phi_1 \supset \Phi_2, \triangleright \Delta}(\supset L)$$

is the desired proof. □

Lemma 9 justifies the fact that, although the calculus is defined in terms of multisets, we may implement it in terms of sets.

We now wish to apply Theorem 8 so as to find proofs, starting from the goal to prove. This will be done according to a strategy (also called a *selection function* in model elimination, for example). But observe that the (mostly arbitrary) total ordering $>$ that we have used to define $>$-canonical proofs can be used in a more precise manner to define a strategy:

Definition 14 *Call a $<$-segment any downwards-closed set I of signed formulas, i.e. any set I such that whenever $s\Phi \in I$ and $s\Phi > s'\Phi'$, then $s'\Phi' \in I$.*

We say that a signed formula is ready *provided that it is not atomic and not a negative implication. We say that a sequent $, \triangleright \Delta$ is I-*ready *whenever there is a ready formula in $I \cap (+, \cup -\Delta)$.*

Intuitively, given a non-atomic sequent $, \triangleright \Delta$ that we wish to prove, we would like to apply some rule other than $(\supset R)$ first. This is possible exactly when there is a ready formula in the sequent. Now assume that I is the set of all signed formulas: then our strategy is to choose the $>$-maximal signed formula among the ready formulas of $+, , -\Delta$.

The index set I is used to constrain proof search so that only $>$-canonical proofs are looked for. Assume that we have obtained the sequent $, \triangleright \Delta$ by expanding some ready formula $s\Phi$: then we wish to find a ready formula to expand in the sequent, but it must also be $<$ than $s\Phi$. Therefore we let I be $\{s'\Phi' \mid s\Phi > s'\Phi'\}$, so that we can apply some rule other than (Ax) or $(\supset R)$ exactly when the sequent is I-ready; then we choose the $>$-maximal formula in $+, , -\Delta$ which is also in I, and expand it.

More formally, we define the following way of searching for $>$-canonical proofs. We define the following non-deterministic procedure $search_1(, \triangleright \Delta, I, S)$. To prove $, \triangleright \Delta$, we call $search_1(, \triangleright \Delta, I_0, \emptyset)$, where I_0 is the set of all signed formulas. We use the convention that all formulas are contracted first, or alternatively that $,$ and Δ are sets. The value of $search_1(, \triangleright \Delta, I, S)$ is defined as:

- If $, \triangleright \Delta$ is an instance of (Ax), then return true.
- Otherwise, if $, \triangleright \Delta$ is in S, then return false (loop check).
- Otherwise, if $, \triangleright \Delta$ is I-ready, then let $s\Phi$ be the $>$-maximal I-ready formula in $+, , -\Delta$, and $, {}_1 \triangleright \Delta_1, \ldots, , {}_n \triangleright \Delta_n$ be the premises of the unique rule that has $s\Phi$ as principal formula. If $search_1(, {}_i \triangleright \Delta_i, \{s'\Phi' \mid s\Phi > s'\Phi'\}, S \cup \{, \triangleright \Delta\})$ returns true for every i, $1 \leq i \leq n$, then return true; otherwise return false. (Apply some non-(Ax) non-$(\supset R)$ rule deterministically.)
- Otherwise, the only non-atomic formulas in $, \triangleright \Delta$ and in I are implications on the right-hand side, i.e. negative implications $-\Phi_i \supset \Phi'_i$, $1 \leq i \leq m$. If $search_1(, , \Phi_i \triangleright \Phi'_i, I_0, S \cup \{, \triangleright \Delta\})$ returns true for some i, $1 \leq i \leq m$, then return true; otherwise, return false. (Non-deterministically find some way of applying $(\supset R)$.)

Using S and I is redundant, however: as long as we only expand ready formulas, there cannot be any loops, because we expand formulas that are lower and lower with respect to the $>$ ordering. Therefore, it is enough to check for loops in the last point of the procedure. We therefore define the following refinement $search(, \triangleright \Delta, I, S)$ of $search_1$:

- If $, \triangleright \Delta$ is an instance of (Ax), then return true.
- Otherwise, if $, \triangleright \Delta$ is I-ready, then let $s\Phi$ be the $>$-maximal I-ready formula in $+, , -\Delta$, and $, {}_1 \triangleright \Delta_1, \ldots, , {}_n \triangleright \Delta_n$ be the premises of the unique rule that has $s\Phi$ as principal formula. If $search(, {}_i \triangleright \Delta_i, \{s'\Phi' \mid s\Phi > s'\Phi'\}, S)$ returns true for every i, $1 \leq i \leq n$, then return true; otherwise return false.
- Otherwise, if $, \triangleright \Delta$ is in S, then return false (loop check).
- Otherwise, the only non-atomic formulas in $, \triangleright \Delta$ and in I are implications on the right-hand side, i.e. negative implications $-\Phi_i \supset \Phi'_i$, $1 \leq i \leq m$. If $search(, , \Phi_i \triangleright \Phi'_i, I_0, S \cup \{, \triangleright \Delta\})$ returns true for some i, $1 \leq i \leq m$, then return true; otherwise, return false. (Non-deterministically find some way of applying $(\supset R)$.)

This observation on loop-checks has also allowed us to replace $S \cup \{, \triangleright \Delta\}$ in the I-ready case by just S. This is beneficial in that the procedure will have to check loops much less often per number of inferences.

Observe also that we have not been precise as to how everything here should be implemented. The set S, a set of sequents, can efficiently be implemented as a zero-suppressed BDD, and in fact as a TDD, since it only contains non-atomic sequents. The segment I can be implemented as either a tag `NONE` denoting I_0, or otherwise as the unique signed formula $s\Phi$ such that $I = \{s'\Phi' \mid s\Phi > s'\Phi'\}$. We shall see in Section 3.3 that we don't even have to represent it explicitly. In general, the TDD implementation in Section 3.3 will be much sleeker.

It remains to show that such an algorithm is complete. The algorithm looks for a special kind of $>$-canonical proof that we call fully canonical:

Definition 15 *Let I_0 be the set of all signed formulas.*

For any total ordering $>$ extending $\succ$, for any $>$-segment I, call a proof π of $, \triangleright \Delta$ fully I-canonical if and only if, writing π as:

$$(R)\ \frac{\begin{array}{ccc}\pi_1 & \ldots & \pi_n \\ \vdots & & \vdots \\ , _1 \triangleright \Delta_1 & \ldots & , _n \triangleright \Delta_n\end{array}}{, \triangleright \Delta}$$

then either:

- *R is (Ax),*
- *or $, \triangleright \Delta$ is not I-ready and R is $(\supset R)$, and $\pi_1, \ldots, \pi_n$ are all fully I_0-canonical,*
- *or $, \triangleright \Delta$ is I-ready, the principal formula $s\Phi$ in R is the $>$-maximal ready formula in $(+, , -\Delta) \cap I$, and letting I' be $\{s'\Phi' \mid s\Phi > s'\Phi'\}$, then $\pi_1, \ldots, \pi_n$ are all fully I'-canonical.*

Call a proof fully $>$-canonical *if and only if it is fully I_0-canonical.*

Then:

Lemma 10 *Every provable sequent has a fully $>$-canonical proof.*

Proof: We check that every $>$-canonical proof can be transformed into a fully $>$-canonical proof. More precisely, for any proof π, define the segment I_π as the set of all formulas $<$ than its final principal formula. We claim that for any $>$-canonical proof π, for any segment $I \supseteq I_\pi$, there is a fully I-canonical proof of the same sequent.

We prove the claim by induction on the size of the proof π. Every $>$-canonical proof of size 0 can be transformed in this way, because there are no proofs of size 0. Otherwise, assume that the induction hypothesis $(*)$:

For any $>$-canonical proof π of size less than k, for any segment $I \supseteq I_\pi$, there is a fully I-canonical proof of the same sequent

holds for all $k < n$, for some $n \geq 1$, and let π be a $>$-canonical proof of size n, I be a segment such that $I \supseteq I_\pi$.

If π ends in (Ax), the claim is clear. Otherwise, the idea is that we use the fact that all rules except $(\supset R)$ are invertible to add new instances of invertible rules below π. In general, this will transform π into a proof of the form:

$$\frac{\begin{array}{cccc}\vdots & \vdots & & \vdots \\ \pi_1 & \pi_2 & \ldots & \pi_n \\ \ddots & \ddots & & \iddots\end{array}}{, \triangleright \Delta}$$

of the same end sequent as π, where $\pi_1, \ldots, \pi_n$ are benign transformations of π, all rules used below $\pi_1, \ldots, \pi_n$ are among the invertible rules, and somehow every rule below $\pi_1, \ldots, \pi_n$ yields a principal formula that

is $>$-maximal among the ready formulas in its conclusion — the principal formula in the end-sequent being $>$-maximal in $(+, , -\Delta) \cap I$.

More formally, we define a new proof $f(\pi, I)$ of $, \triangleright \Delta$ by well-founded induction on $>$ (which is possible because we are here dealing with the restriction of $>$ to the finite set of subformulas of the given sequent) by defining the expression $f(\pi', I')$ for any benign transformation π' of π, and for any $I' \supseteq I_{\pi'}$, as follows. Note that $f(\pi', I')$ is undefined in any other case.

If the principal formula $s\Phi$ in the end sequent of π' is not the greatest of its signed formulas in I', then we do the following. Assume that $s\Phi$ is a positive implication $+\Phi_1 \supset \Phi_2$ (the other cases are similar or simpler). Then the end sequent of π' can be written $, ', \Phi_1 \supset \Phi_2 \triangleright \Delta'$, and letting I'' be $\{s'\Phi' \mid +\Phi_1 \supset \Phi_2 > s'\Phi'\}$, we define $f(\pi', I')$ as:

$$(\supset L)\ \dfrac{\begin{array}{cc} f((\pi', \Phi_1), I'') & f([\Phi_1 \supset]\pi', I'') \\ \vdots & \vdots \\ , ', \Phi_1 \supset \Phi_2 \triangleright \Phi_1, \Delta' & , ', \Phi_2 \triangleright \Delta' \end{array}}{, ', \Phi_1 \supset \Phi_2 \triangleright \Delta'}$$

If the principal formula $s\Phi$ in the end sequent of π' is already the greatest element of $(+, , -\Delta) \cap I'$, then we let $f(\pi', I')$ be a fully $>$-canonical proof that we get from the induction hypothesis $(*)$. By Lemma 5, $|\pi'| \leq |\pi| = n$, so we cannot apply $(*)$ immediately. Instead, we look at the last rule in π'. If π' ends in (Ax), then let $f(\pi', I')$ be π'. If π' ends in $(\supset R)$, i.e. π' is:

$$(\supset R)\ \dfrac{\begin{array}{c} \pi'_1 \\ \vdots \\ , ', \Phi_1 \triangleright \Phi_2 \end{array}}{, ' \triangleright \Phi_1 \supset \Phi_2, \Delta'}$$

then $|\pi'_1| < n$ and $I_0 \supseteq I_{\pi'_1}$, so we apply $(*)$ on π'_1, and we can therefore replace π'_1 in π' by a fully I_0-canonical proof of $, ', \Phi_1 \triangleright \Phi_2$. This way, π' is changed into a fully I_0-canonical proof of $, ' \triangleright \Phi_1 \supset \Phi_2, \Delta'$: we let $f(\pi', I')$ be the latter. Finally, if π' ends in some other rule, say $(\supset L)$, then π' has the form:

$$(\supset L)\ \dfrac{\begin{array}{cc} \pi_1 & \pi_2 \\ \vdots & \vdots \\ , ', \Phi_1 \supset \Phi_2 \triangleright \Phi_1, \Delta' & , ', \Phi_2 \triangleright \Delta' \end{array}}{, ', \Phi_1 \supset \Phi_2 \triangleright \Delta'}$$

and $|\pi_1| < n$ (resp. $|\pi_2| < n$), $I_{\pi'} \supseteq I_{\pi_1}$ (resp. $I_{\pi'} \supseteq I_{\pi_2}$) because π' is $>$-canonical, so that we can apply $(*)$ on π_1 and π_2, transforming them into fully $I_{\pi'}$-canonical proofs of $, ', \Phi_1 \supset \Phi_2 \triangleright \Phi_1, \Delta'$ (resp. $, ', \Phi_2 \triangleright \Delta'$). Replacing π_1 and π_2 by the latter in π' yields a proof that we define as being $f(\pi', I')$.

Now, by construction $f(\pi, I)$ is fully I-canonical, and we are done. □

Note that, whereas our main arguments for concentrating on canonical proofs was the set of permutabilities of the system of logic at hand, restricting to fully canonical proofs requires us to examine the invertibility status of rules as well.

Moreover, while finding canonical proofs was justified by induction on the size of proofs, full canonicity requires that we also use induction on the strategy, i.e. the ordering $>$. Should we wish to extend this procedure to, say, first-order intuitionistic logic, we would then be forced to extend $\succ$ so that $+\forall x \cdot \Phi \succ +\Phi[t/x]$ for instance (which is no problem) but also to $+\forall x \cdot \Phi \succ +\forall x \cdot \Phi$, to be able to represent the quantifier rule:

$$(\forall L)\ \dfrac{, , \forall x \cdot \Phi, \Phi[t/x] \triangleright}{, , \forall x \cdot \Phi \triangleright \Delta}$$

The fact that we must allow for $+\forall x \cdot \Phi \succ +\forall x \cdot \Phi$ is due to the fact that this duplication of $+\forall x \cdot \Phi$ is unavoidable in general, but unfortunately it prevents $\succ$ from being an ordering, which destroys our arguments. Fortunately, there is a simple way out, and it is to index quantifiers by indexes, and replace the $(\forall L)$ rule above by the following schema:

$$(\forall L_i)\ \dfrac{, , \forall_{i+1} x \cdot \Phi, \Phi[t/x] \triangleright}{, , \forall_i x \cdot \Phi \triangleright \Delta}$$

for every integer index i. We then assume that $\forall x \cdot \Phi$ just means $\forall_0 x \cdot \Phi$. We leave it to the reader to check that this does not change the notion of provability in first-order intuitionistic logic, and that we can define total orderings that terminate on the set of subformulas that appear in any given proof. This is slightly trickier than in the propositional case, and although getting $>$-canonical proofs from general proofs (the analogue of Theorem 8) won't be harder, getting fully $>$-canonical proofs will crucially depend on termination of an ordering where $\forall_0 x \cdot \Phi > \forall_1 x \cdot \Phi > \ldots > \forall_i x \cdot \Phi > \ldots$, i.e. we have to bound the number of quantifier expansions in advance. (This is easy, because these transformations will never create any indexes that were not in the original proof.) This justifies the usual tableau procedure that consists of fixing a so-called γ-limit on the number of quantifier expansions.

Having made this important remark on what it takes to get various levels of canonicity in proofs, we return to our main topic:

Theorem 11 *For any total ordering $>$ extending $\succ$, the search procedure terminates; moreover, $search(, \triangleright \Delta, I_0, \emptyset)$ returns true if and only if $, \triangleright \Delta$ is provable in the system of Figure 3.*

Proof: The tricky point of this proof is loop checking. Termination is obvious, because *search* may only generate finitely many non-I-ready sequents (which are here sets), and because $>$ is well-founded on finite sets of subformulas, *search* can only generate finitely many consecutive I-ready sequents. So the loop checks force termination.

If $search(, \triangleright \Delta, I_0, \emptyset)$ returns true, moreover, then it has implicitly built a proof of $, \triangleright \Delta$, which we might in fact as well return in place of the terser answer "true".

It remains to show that if $, \triangleright \Delta$ is provable, then $search(, \triangleright \Delta, I_0, \emptyset)$ will return true. So assume $, \triangleright \Delta$ provable. First, by Lemma 10, it has a fully $>$-canonical proof. Consider such a proof π of smallest size. In π, no instance of $(\supset R)$ is ever repeated along any branch of the proof. Indeed, otherwise π would look like:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\pi_1}{\vdots}}{, {}_1 \triangleright \Delta_1} \quad \dfrac{\pi_2}{\vdots}}{\vdots} \quad \dfrac{\pi_3}{\vdots}}{, {}_1 \triangleright \Delta_1} \quad \dfrac{\pi_4}{\vdots}}{\vdots}}{, \triangleright \Delta}$$

where both instances of $, {}_1 \triangleright \Delta_1$ are conclusions of an $(\supset R)$ instance. But then, we could transform this into:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\pi_1}{\vdots}}{, {}_1 \triangleright \Delta_1} \quad \dfrac{\pi_2}{\vdots}}{\vdots} \quad \dfrac{\pi_4}{\vdots}}{\vdots}}{, \triangleright \Delta}$$

which would be fully $>$-canonical (notice that this depends on the fact that both occurrences of $, {}_1 \triangleright \Delta_1$ are conclusions of instances of $(\supset R)$; for any other rules, the question of which I we should consider would muck things up), and moreover would be strictly smaller than π, a contradiction.

Given such a fully $>$-canonical proof π with no repetition of instances of $(\supset R)$, it is now easy to guide *search* by π, so that *search* returns true. $\Box$

Observe that we have proved *search* to be complete, but $search_1$ strangely seems harder to prove complete. In fact, $search_1$ might even be incomplete, because it may be the case that some sequents need to be repeated

in certain fully $>$-canonical proofs; these would be rejected by $search_1$. We estimate this to be unlikely, and conjecture that $search_1$ would also be complete. In fact, we don't care, since *search* is in any case more efficient than $search_1$.

3.3 A TDD Calculus for Intuitionistic Logic

Theorem 11, or the *search* procedure, suggests the following canonical tableau proof-search procedure. First, from a given initial tableau t, compute the tableau $N^*(t)$, where $N^*(t)$ is defined recursively on tableaux t as follows. Recall that a signed formula is called ready if and only if it is not atomic, and not a negative implication.

- If no signed formula in t is ready, then $N^*(t) = t$;
- Otherwise, let $s\Phi$ be a $\succ$-maximal ready signed formula among those that occur in t. Let R be the only expansion rule that can apply to $s\Phi$.

  Let t_1 be the subset of t consisting of all paths in which $s\Phi$ occurs, and t_0 be the subset of t consisting of all paths in which $s\Phi$ does not occur. Compute t_2, the set of all expansions of paths in t_1 by R on $s\Phi$. Then $N^*(t) = N^*(t_0 \cup t_2)$.

This definition of N^* is well-founded, as the sum of the sizes of the ready signed formulas occurring in $t_0 \cup t_2$ is strictly less that that occurring in t. (Beware that we sum over all distinct formulas, not over all their occurrences.)

While computing $N^*(t)$ by expanding kernels in tableaux (top-down, as above) is the most obvious way, there is another solution, namely to build it bottom-up, as is the tradition in the BDD world [1]. For example, assume that we wish to compute $N^*(\{\{+(\Phi_1 \ \& \ \Phi_2)\}\})$. We may compute it top-down by setting up the initial tableau $\{\{+(\Phi_1 \ \& \ \Phi_2)\}\}$, then by expanding $+(\Phi_1 \ \& \ \Phi_2)$ by rule $(+\&)$, giving $\{\{+\Phi_1\}, \{+\Phi_2\}\}$. Or we may first compute the tableaux t_1 for $\{\{+\Phi_1\}\}$ and t_2 for $\{\{+\Phi_2\}\}$, and produce the tableau whose paths are the unions of paths in t_1 and paths in t_2, i.e. the shuffle $t_1 \mathbin{@} t_2$.

We therefore define the following function:

```
fun TDDintN (+⊥) = 0 (* rule (+⊥) *)
  | TDDintN (−⊥) = 1 (* rule (−⊥) *)
  | TDDintN (+(Φ1 & Φ2)) = (* rule (+&) *)
      TDDshuffle (TDDintN (+Φ1), TDDintN (+Φ2))
  | TDDintN (−(Φ1 & Φ2)) = (* rule (−&) *)
      TDDunion (TDDintN (−Φ1), TDDintN (−Φ2))
  | TDDintN (+(Φ1 ∨ Φ2)) = (* rule (+∨) *)
      TDDunion (TDDintN (+Φ1), TDDintN (+Φ2))
  | TDDintN (−(Φ1 ∨ Φ2)) = (* rule (−∨) *)
      TDDshuffle (TDDintN (−Φ1), TDDintN (−Φ2))
  | TDDintN (+(Φ1 ⊃ Φ2)) = (* rule (+⊃) *)
      TDDunion (TDDshuffle (TDDmake (+(Φ1 ⊃ Φ2)),
                            TDDintN (−Φ1)),
                TDDintN (+Φ2))
  | TDDintN (sΦ) = TDDmake (sΦ)
```

The correctness of `TDDintN` (Lemma 13) relies on the following lemma:

Lemma 12 *N^* commutes with unions and shuffles, i.e. for any tableaux t and t', $N^*(t \cup t') = N^*(t) \cup N^*(t')$ and $N^*(t \mathbin{@} t') = N^*(t) \mathbin{@} N^*(t')$.*

Proof: By induction on the sum n of the sizes of ready formulas in $t \cup t'$. If $n = 0$, then this is clear: $N^*(t \cup t') = t \cup t' = N^*(t) \cup N^*(t')$, and $N^*(t \mathbin{@} t') = t \mathbin{@} t' = N^*(t) \mathbin{@} N^*(t')$.

Otherwise, assume that $n > 0$. Let $s\Phi$ be the $\succ$-maximal one and R be the associated rule. Let t_1 be the subset of t consisting of all paths in which $s\Phi$ occurs, and t_0 be that of all paths in which it does not occur. Similarly, let t'_1 be the subset of t' consisting of all paths in which $s\Phi$ occurs, and t'_0 be that of all

paths in which it does not occur. Then the set of all paths in $t \cup t'$ where $s\Phi$ occurs is $t_1 \cup t'_1$, and the set of all paths in $t \cup t'$ where $s\Phi$ does not occur is $t_0 \cup t'_0$. Then $N^*(t \cup t') = N^*((t_0 \cup t_1) \cup (t'_0 \cup t'_1)) = N^*((t_0 \cup t'_0) \cup (t_1 \cup t'_1))$ by associativity and commutativity of $\cup$. Let t_2 (resp. t'_2) be the set of all expansions of paths in t_1 (resp. t'_1) by rule R on $s\Phi$. Then $t_2 \cup t'_2$ is the set of all expansions of paths in $t_1 \cup t'_1$, so $N^*(t \cup t') = N^*((t_0 \cup t'_0) \cup (t_2 \cup t'_2)) = N^*((t_0 \cup t_2) \cup (t'_0 \cup t'_2))$ (by associativity and commutativity of $\cup$) $= N^*(t_0 \cup t_2) \cup N^*(t'_0 \cup t'_2)$ (by induction hypothesis) $= N^*(t) \cup N^*(t')$.

The argument is similar for $t @ t'$: $N^*(t @ t') = N^*((t_0 \cup t_1) @ (t'_0 \cup t'_1)) = N^*((t_0 @ t'_0) \cup (t_0 @ t'_1) \cup (t'_0 @ t_1) \cup (t_1 @ t'_1))$, where the paths that contain $s\Phi$ are in $(t_0 @ t'_1) \cup (t'_0 @ t_1) \cup (t_1 @ t'_1)$. Their expansions by R on $s\Phi$ are the paths in $(t_0 @ t'_2) \cup (t'_0 @ t_2) \cup (t_2 @ t'_2)$, hence $N^*(t @ t') = N^*((t_0 @ t'_0) \cup (t_0 @ t'_2) \cup (t'_0 @ t_2) \cup (t_2 @ t'_2)) = N^*((t_0 \cup t_2) @ (t'_0 \cup t'_2)) = N^*(t_0 \cup t_2) @ N^*(t'_0 \cup t'_2)$ (by induction hypothesis) $= N^*(t) @ N^*(t')$. □

Lemma 13 *For any tableau t, let $N(t)$ be the star-erasure of $N^*(t)$, i.e. $N^*(t)$ with every formula Φ^* replaced by Φ, from which every closed path has been removed.*

For any signed formula $s\Phi$, `TDDintN` *$(s\Phi)$ computes the TDD of $N(\{\{s\Phi\}\})$, for any total ordering $\succ$ such that $s\Phi \succ s'\Phi'$ whenever $s\Phi$ is not a negative implication and Φ' is an immediate subformula of Φ.*

Proof: First, notice the following. Let $TDDintN'$ be defined as `TDDintN`, except that `TDDunion`, `TDDshuffle` and `TDDmake` are replaced by their equivalent on zero-suppressed BDDs, i.e. the closing reduction rule is not applied any longer. Observe then that `TDDintN` $(s\Phi)$ is just the normal form of $TDDintN'(s\Phi)$ by the TDD reduction rules: this is because the TDD reduction rules form a convergent rewrite system.

It therefore only remains to show that $TDDintN'(s\Phi) = D(N(\{\{s\Phi\}\}))$, or rather that $N(\{\{s\Phi\}\}) = T(TDDintN'(s\Phi))$, which is equivalent by Theorem 3. We prove it by induction on the size of $s\Phi$. The only non-trivial case is when this argument is $+(\Phi_1 \supset \Phi_2)$. Then $N(\{\{-\Phi_1\}\})$ equals $T(TDDintN'(-\Phi_1))$ and $N(\{\{+\Phi_2\}\})$ equals $T(TDDintN'(+\Phi_2))$ by induction hypothesis. Then:

$$\begin{array}{l} N^*(\{\{+(\Phi_1 \supset \Phi_2)\}\}) \\ = N^*(\{\{+(\Phi_1 \supset \Phi_2)^*, -\Phi_1\}, \{+\Phi_2\}\}) \\ = N^*(\{\{+(\Phi_1 \supset \Phi_2)^*, -\Phi_1\}\}) \cup N^*(\{\{+\Phi_2\}\}) \\ = (N^*(\{\{+(\Phi_1 \supset \Phi_2)^*\}) @ N^*(\{\{-\Phi_1\}\})) \cup N^*(\{\{+\Phi_2\}\}) \end{array}$$

by Lemma 12. But $N^*(\{\{+(\Phi_1 \supset \Phi_2)^*\}) = \{\{+(\Phi_1 \supset \Phi_2)^*\}\}$, so:

$$N^*(\{\{+(\Phi_1 \supset \Phi_2)\}\}) = (\{\{+(\Phi_1 \supset \Phi_2)^*\}\} @ N^*(\{\{-\Phi_1\}\})) \cup N^*(\{\{+\Phi_2\}\})$$

Therefore,

$$\begin{array}{l} N(\{\{+(\Phi_1 \supset \Phi_2)\}\})) \\ = (\{\{+(\Phi_1 \supset \Phi_2)\}\} @ T(TDDintN'(-\Phi_1))) \cup T(TDDintN'(+\Phi_2)) \\ = T(TDDintN'(+(\Phi_1 \supset \Phi_2)) \end{array}$$

□

Note that `TDDintN` forces to expand the tableau with (Ax) or (Ax^*) applied as late as possible. Indeed, by Lemma 13, it does the same thing as expanding the tableau by N, which never applies (Ax) or (Ax^*), and then eliminates all closed paths. That is, although this choice of implementation keeps the code simple, it may have detrimental effects like expanding paths like $+\Phi_1 \supset \Phi_2, -\Phi_1 \supset \Phi_2$ (which could be closed) into tableaux like $+(\Phi_1 \supset \Phi_2)^*, -\Phi_1, -(\Phi_1 \supset \Phi_2)$ plus $+\Phi_2, -(\Phi_1 \supset \Phi_2)$, of which the second cannot be closed right away without using rule $(- \supset)$ (or $(limpR)$). This process may be done recursively, producing non-trivial expansions with a huge amount of unclosed paths, starting from only one closable path. We shall wait until Section 4 (experimental results) to see whether this is an important matter or not.

Using `TDDintN`, we define the following proof-search procedure, which alternates between appling eagerly all rules but $(\supset R)$ then (Ax), and applying $(\supset R)$ when we are forced to. To prove (or disprove) a formula Φ, compute $prove(D(\{\{-\Phi\}\}), \emptyset)$, where `prove`$(F, S)$ is defined on all TDDs F and finite sets S of TDDs (used to implement the loop-check) by:

- Let G be $TDDintN\ \ (F)$.
- If $G \in S$, then $prove\ \ (F, S)$ returns false (unprovable by loop-check).

- If $G = \mathbf{0}$, then *prove* (F, S) returns true (proved).
- If $G = \mathbf{1}$, then *prove* (F, S) returns false (unprovable).
- Otherwise, *prove* (F, S) returns true if and only if, for all paths p in G, there is a negative implication in p that we can expand by rule $(- \supset)$ (i.e., $(\supset R)$) into a path p', such that *prove* $(D(\{p'\}), S \cup \{F\})$.

The exploration of paths in the last step is easily done by using the function `TDDiter` of Section 2.3, as follows:

```
let val S' = S ∪ {F}
in
   TDDiter (fn p => (* take a path, and try to close it *)
            if there is sΦ of the form −(Φ1 ⊃ Φ2) in p
               such that prove (TDDshuffle (TDDshuffle (TDDmake (+Φ1),
                                                        TDDmake (−Φ2)),
                                            posPathToTdd p), S')
               then raise TDDITER (* success *)
            else false (* fail *)
            ) F handle TDDITER => true
end
```

where `posPathToTdd` builds a TDD from a the positive formulas in the (sorted) list of signed formulas given as arguments:

```
fun posPathToTDD p =
    let fun t ([], r) = r
          | t (+Φ :: l, r) = t (l, TDDnode (+Φ, 0, r))
          | t (−Φ :: l, r) = t (l, r)
    in
       t (p, 1)
    end
```

3.4 Optimizations

We can make the procedure of the last subsection more efficient by noticing a few facts.

3.4.1 Avoiding Redundancy.

First, it may be the case that, in the last step of the proof procedure, where we try to close all remaining paths by first applying $(- \supset)$, we have to close essentially the same tableaux over and over again. More precisely, assume that we have two paths p_1 and p_2 in G. We first try to prove p_1 by finding a negative implication in it which we can use to expand p_1 into a tableau that we can close. Doing this, it is often the case that we produce intermediate tableaux that contain paths which we shall meet again when trying to close p_2. We can therefore remember (in a global tableau `success`) a set of paths that we have already managed to close; before we try to close all paths in a TDD, we first eliminate from it all paths that are already in `success` (more generally, those that are *weakenings* of paths in `success`), which we have already closed earlier; and whenever we succeed in closing all paths of a TDD F, we add them all to `success` (by assigning `TDDunion`$(F, success)$ to `success`).

Computing the set of all paths in F that are not weakenings of some paths in another TDD F' (`success`) is done by computing `TDDwfilter` (F, F'), which is defined recursively as follows:

```
memofun TDDwfilter (0, F') = 0
      | TDDwfilter (F, 0) = F
      | TDDwfilter (F, 1) = 0
      | TDDwfilter (1, F') = if hasI F' then 0 else 1
```

```
  | TDDwfilter (F as sΦ ⟶ F1/F0, F' as s'Φ' ⟶ F1'/F0') =
    if sΦ = s'Φ'
       then TDDnode (sΦ,TDDwfilter (F0, F0'),
                         TDDwfilter (F1, TDDunion (F0', F1')))
    else if sΦ < s'Φ'
            then TDDnode (sΦ,TDDwfilter (F0, F'),
                                TDDwfilter (F1, F'))
    else TDDwfilter (F, F0')
```

where `hasI` tests whether its argument is a tableau containing the empty path:

```
fun hasI 1 = true
  | hasI 0 = false
  | hasI (sΦ ⟶ F1/F0) = hasI F0
```

3.4.2 Using Oracles.

A second optimization that we can apply comes from the fact that any provable intuitionistic formula and also provable in classical logic. We can therefore query an oracle for classical propositional logic before trying to expand paths. If the oracle returns "invalid", then we may stop the search right away.

Such an oracle can be implemented by any procedure we like, including a TDD method for classical logic. Since all rules in classical propositional logic permute, it suffices to define:

```
fun TDDclassN (+⊥) = 0 (* rule (+⊥) *)
  | TDDclassN (−⊥) = 1 (* rule (−⊥) *)
  | TDDclassN (+(Φ1 & Φ2)) = (* rule (+&) *)
      TDDshuffle (TDDclassN (+Φ1), TDDclassN (+Φ2))
  | TDDclassN (−(Φ1 & Φ2)) = (* rule (−&) *)
      TDDunion (TDDclassN (−Φ1), TDDclassN (−Φ2))
  | TDDclassN (+(Φ1 ∨ Φ2)) = (* rule (+∨) *)
      TDDunion (TDDclassN (+Φ1), TDDclassN (+Φ2))
  | TDDclassN (−(Φ1 ∨ Φ2)) = (* rule (−∨) *)
      TDDshuffle (TDDclassN (−Φ1), TDDclassN (−Φ2))
  | TDDclassN (+(Φ1 ⊃ Φ2)) = (* classical rule (+⊃) *)
      TDDunion (TDDclassN (−Φ1),
                TDDclassN (+Φ2))
  | TDDclassN (−(Φ1 ⊃ Φ2)) = (* classical rule (−⊃) *)
      TDDshuffle (TDDclassN (+Φ1), TDDclassN (−Φ2))
  | TDDclassN (sΦ) = TDDmake (sΦ)
```

It is easily seen that `TDDclassN` $(-\Phi)$ equals **0** if and only if Φ is classically provable.

Because classical provability is coNP-complete, `TDDclassN` may take exponential time to compute; we may therefore limit the time taken by the oracle by placing an upper bound on the size of the classical TDD and pruning it so as to stay below the size limit. This loses some information, but may be useful. This is exactly what Laurent Mauborgne did in [14], with standard BDDs. We may also use other oracles, say oracles that would test the formula on specified Kripke frames, or that would draw models at random and check that the formula holds on the random model. We have not implemented any of these ideas.

3.4.3 Special Cases for Negative Implications.

Finally, we can also refine our analysis by noticing that in the procedure of the last section, G can be split into the TDD G_0 of all paths that do not contain negative implications, the TDD G_1 of all paths that contain exactly one negative implication, and all other paths G_2.

Then, if G_0 is not **0**, then F is not provable (we cannot apply any rule on it). If G_0 is **0**, then G_1 is the disjoint union of TDDs $G_1(-\Phi_1 \supset \Phi_2))$, whose paths are all those whose sole negative implication is

$-(\Phi_1 \supset \Phi_2)$. The point is that instead of sweeping through the space of all paths in sequence, we may apply $(- \supset)$ in parallel on all paths on each $G_1(-\Phi_1 \supset \Phi_2))$. If some $G_1(-\Phi_1 \supset \Phi_2))$ is not provable, then return false. Otherwise, do the last step of the procedure on G_2 instead of G.

Finding the G_0 part is done by computing $G_0 = Gzero\ G$:

```
memofun Gzero 0 = 0
      | Gzero 1 = 1
      | Gzero (sΦ ⟶ F1/F0) =
        case sΦ of
             −(Φ1 ⊃ Φ2) => Gzero F0
           | _ => TDDnode (sΦ, Gzero F0, Gzero F1)
```

We compute the G_1 parts by computing `Gone` G, which returns a finite map from negative implications $-(\Phi_1 \to \Phi_2)$ to $G_1'(-(\Phi_1 \to \Phi_2))$, the set of all paths in $G_1(-(\Phi_1 \to \Phi_2))$ where $-(\Phi_1 \to \Phi_2)$ has been taken away. In the following, finite maps are taken to be finite sets of bindings $s\Phi \mapsto F$:

```
memofun Gone 0 = {}
      | Gone 1 = {}
      | Gone (sΦ ⟶ F1/F0) =
        case sΦ of
             −(Φ1 ⊃ Φ2) => Gone F0 ∪ {sΦ ↦ Gzero F1}
           | _ =>
             let val F0' = Gone F0 and F1' = Gone F1
             in
                 {s'Φ' ↦ TDDnode (sΦ, F0'(s'Φ'), F1'(s'Φ'))
                  | s'Φ' ∈ dom F0' ∩ dom F1'}∪
                 {s'Φ' ↦ F0'(s'Φ') | s'Φ' ∈ dom F0' \ dom F1'}∪
                 {s'Φ' ↦ TDDnode (sΦ, 0, F1'(s'Φ'))
                  | s'Φ' ∈ dom F1' \ dom F0'
                    such that F1'(s'Φ') ≠ 0}
             end
```

Computing G_1 is done by applying the following function `Gunion` to `Gone` (G):

```
fun Gunion {} = 0
  | Gunion {sΦ ↦ F} =
    TDDshuffle (TDDmake sΦ, F)
  | Gunion (s1 ∪ s2) =
    TDDunion (Gunion s1, Gunion s2)
```

where the pattern $s_1 \cup s_2$ is intended to split its arguments into two non-empty disjoint sets (with roughly as many elements each).

Given a TDD F, we may extract the TDD of all p^+ for p ranging over paths in F by computing `TDDPpart` F:

```
memofun TDDPpart 0 = 0
      | TDDPpart 1 = 1
      | TDDPpart (−Φ ⟶ F1/F0) = TDDunion (TDDPpart F0, TDDPpart F1)
      | TDDPpart (+Φ ⟶ F1/F0) = TDDnode (+Φ, TDDPpart F0, TDDPpart F1)
```

To sum up, we therefore reimplement the `prove`(F, S) procedure as follows (`success` is initialized to the empty tableau $\mathbf{0}$):

- If $F \in S$, then $prove\ (F, S)$ returns false (unprovable by loop-check).
- Let G be $TDDwfilter\ (TDDintN\ (F), success)$. (`TDDwfilter` is used to avoid reexpanding redundant paths.)

- Let G_0 be $Gzero\ \ G$. If $G_0 \neq \mathbf{0}$, then return false (unprovable).
- Otherwise, if $TDDclassN\ \ G \neq \mathbf{0}$, then return false (oracle says no).
- Otherwise, let s be the map $Gone\ \ G$. If for some $-(\Phi_1 \supset \Phi_2) \mapsto G'$ in s,

  ```
  prove (TDDshuffle (TDDshuffle (TDDmake (+Φ1),
                                 TDDmake (−Φ2)),
                     TDDPpart G'),
         S ∪ {F})
  ```

  returns false, then return false. (Expand paths in $G_1(-(\Phi_1 \supset \Phi_2))$ in parallel.)
- Otherwise, let G_2 be $TDDwfilter\ \ (G, Gunion\ \ s)$: return true if and only if, for all paths p in G_2, there is a negative implication in p that we can expand by rule $(-\ \supset)$ (i.e., $(\supset\ R)$) into a path p', such that $prove\ \ (D(\{p'\}), S \cup \{F\})$. If so, then do `success := TDDunion` $(success, G)$. (Record all successful paths for redundancy checking.)

4 Experimental Results

Experimental results are needed to validate our approach, not only because they are in general, but because we can run two contradicting arguments as to how efficient our procedure should be in practice.

The first argument is the thesis that we have developed here: TDDs provide an implementation of tableaux where (because of canonicity of representation, and mostly because of the nice interaction between sharing and canonical proofs) many paths can be closed or expanded in parallel, while keeping the non-determinism low.

The counter-argument is a now classical fact about uniform proofs, which extends to canonical proofs in general: an expansion strategy that clings to the motto of finding a proof in some canonical format may be highly detrimental in terms of proof length. Intuitively, there may be formulas that can be proved by expanding its tableaux in, say, 2 paths that can be closed by well-chosen instances of the $(-\ \supset)$ rule, but whose expansion by the N function (which expands all formulas but negative implications) would produce one million paths with several negative implications on each. This opens the possibility of having very simple formulas (for the right strategy) on which our prover would stall.

We have tested six versions of the prover:

- (1) is the basic prover of Section 3.3.
- (2) is (1) plus redundancy elimination through the use of an auxiliary `success` TDD.
- (3) is (2) plus the use of the classical oracle of Section 3.4.
- (4) is (1) plus the refinement on the use of $(-\ \supset)$ of the end of Section 3.4.
- (5) is (4) plus redundancy elimination.
- (6) is (5) plus the use of a classical oracle, i.e. the last incarnation of `prove` that we have described.
- (C) is just the oracle `TDDclassN`, which we put here so as to be able to make comparisons between the efficiency of the intuitionistic prover and that of a TDD prover for propositional classical logic.

We have used a simple lexicographic path ordering (lpo) on formulas to order the TDDs, so that $\Phi_1 < \Phi_2$ if and only if Φ_1 is greater than Φ_2 in the lpo. Atomic variables are compared by their names, lexicographically, and the lpo is based on a precedence such that $\& > \vee > \supset > \bot > A$ for any variable A; this is all rather arbitrary, but follows the heuristic that, if we don't have better knowledge of the nature of the formula, a BDD ordering based on the textual order of appearance of formulas usually works well.

Moreover, the refinement on the use of $(-\ \supset)$ of the end of Section 3.4 that we use in versions (4) through (6) needs to sweep through all bindings $s\Phi \mapsto F$ in `Gone` G, and this has to be done in some order.

By default, this order was unspecified in the first implementations, and test results showed some extreme variations in execution time as the order changed. Although the data of Figure 5 seemed mostly unchanged, those of Figure 6 were erratic: for instance, we got results ranging from 2.46 seconds to non termination in less than 20 minutes for the $Urq(6)$ line, column (5). We therefore decided on the following heuristic: sweep through all the bindings $s\Phi \mapsto F$ in increasing rough complexity order, where the rough complexity order is defined as follows (this is mostly arbitrary, except that we wish to express that what counts most during the search is how many negative implications will crop up). We define the following weight function w on signed formulas: $w(+A) = w(-A) = w(+\bot) = w(-\bot) = 1$, $w(+\Phi_1 \vee \Phi_2) = w(+\Phi_1 \ \& \ \Phi_2) = w(+\Phi_1) + w(+\Phi_2)$, $w(-\Phi_1 \vee \Phi_2) = w(-\Phi_1 \ \& \ \Phi_2) = w(-\Phi_1) + w(-\Phi_2)$, $w(+\Phi_1 \supset \Phi_2) = w(-\Phi_1) + w(+\Phi_2)$, and $w(-\Phi_1 \supset \Phi_2) = w(+\Phi_1) * w(-\Phi_2)$ (where the last clause uses multiplication instead of addition, therefore giving more weight in general to negative implications). Then we decide that $s\Phi$ is roughly less complex than $s'\Phi'$ if and only if $w(s\Phi) < w(s'\Phi')$, or $w(s\Phi) = w(s'\Phi')$ and $s\Phi$ is less than $s'\Phi'$ under some total ordering (the latter being used to break ties, so that our procedure runs deterministically; we have chosen the lpo above for simplicity).

As test formulas, first we have chosen the propositional problems in Pelletier's [18] test suite (entries 1 through 17). Some of them are equivalences $P \equiv Q$: if n is such a problem, then we have added entries n_1 (formula $P \supset Q$) and n_2 (formula $Q \supset P$). Formula I is $A \supset A$, S is $(A \supset B \supset C) \supset (A \supset B) \supset A \supset C$, K is $A \supset B \supset A$. $Nepr$ denotes Neprevoda's formula $((((P \supset Q) \supset P) \supset P) \supset Q) \supset Q$, a formula that is reputedly difficult to prove in natural deduction systems. For every positive integer n, $Urq(n)$ is the Urquhart formula:

$$(((\ldots(x_1 \equiv x_2) \equiv \ldots \equiv x_n) \equiv x_1) \equiv x_2) \equiv \ldots \equiv x_n$$

where $P \equiv Q$ denotes $(P \supset Q) \ \& \ (Q \supset P)$, so that the size of this formula is in fact an exponential in n. Finally, for every integers m and n, $pigm;n$ is a formula that says that if we have m holes and n pigeons, and every pigeon is in some hole, then there is a hole with more than one pigeon. (Which is both classically valid and intuitionistically provable if and only if $m < n$.) This formula is:

$$\begin{array}{l} in\_some\_hole(1) \ \& \ \ldots \ \& \ in\_some\_hole(n) \\ \supset more\_than\_one\_pigeon(1) \vee \ldots \vee more\_than\_one\_pigeon(m) \end{array}$$

where

$$\begin{array}{l} in\_some\_hole(j) = \\ in(j,1) \vee in(j,2) \vee \ldots \vee in(j,m) \end{array}$$

$$\begin{array}{l} more\_than\_one\_pigeon(i) = \\ (in(2,i) \ \& \ in(1,i)) \vee \\ (in(3,i) \ \& \ in(1,i)) \vee (in(3,i) \ \& \ in(2,i)) \vee \\ \ldots \\ (in(j,i) \ \& \ in(1,i)) \vee (in(j,i) \ \& \ in(2,i)) \vee \ldots \vee (in(j,i) \ \& \ in(j-1,i)) \vee \\ \ldots \\ (in(n,i) \ \& \ in(1,i)) \vee (in(n,i) \ \& \ in(2,i)) \vee \ldots \vee (in(n,i) \ \& \ in(n-1,i)) \end{array}$$

where each $in(j,i)$ is a propositional variable expressing that pigeon j is in hole i.

All these formulas are classically valid (hence invalidating the role of the classical oracle at the start of the search — but not necessarily afterwards), but not all are intuitionistically provable. The results are shown in Figures 5 and 6. Size comes into two flavors: dag size is the number of distinct subformulas in the formula, or alternatively the size of the formula represented as a directed acyclic graph where all common subformulas are shared; tree size denotes the number of distinct occurrences in the formula, or the size of the formula represented as a tree (as though we didn't share common subformulas). The latter is a more relevant measure of the size of formulas for tableau procedures: intuitively, paths in tableaux are split on an occurrence basis, and identical formulas cannot be expanded in the same way on all paths in a tableau. On the other hand, our implementation as TDDs allows us to recover some level of sharing, hence of parallelism in expansions of identical formulas, whatever the paths they are found in.

The tests were conducted on an implementation of the prover in HimML [9], a bytecoded implementation of the core Standard ML language, with fast equality and fast finite set and map operations. The figures we

Problem	dag size	tree size	provable?	(1)	(2)	(3)	(4)	(5)	(6)	(C)
I	2	3	$\surd$	0.00	0.00	0.02	0.00	0.00	0.02	0.00
S	9	13	$\surd$	0.10	0.12	0.17	0.10	0.10	0.13	0.03
K	4	5	$\surd$	0.02	0.02	0.02	0.02	0.02	0.02	0.00
1	10	23		0.07	0.10	0.13	0.07	0.07	0.10	0.00
1_1	8	11	$\surd$	0.03	0.05	0.05	0.05	0.05	0.07	0.02
1_2	8	11		0.05	0.05	0.08	0.07	0.08	0.08	0.00
2	7	15		0.03	0.03	0.05	0.03	0.03	0.03	0.02
2_1	5	7		0.02	0.02	0.02	0.03	0.02	0.03	0.00
2_2	5	7	$\surd$	0.00	0.02	0.03	0.02	0.03	0.02	0.00
3	7	9	$\surd$	0.03	0.02	0.03	0.03	0.03	0.05	0.02
4	10	23		0.07	0.05	0.08	0.08	0.08	0.10	0.02
4_1	8	11		0.05	0.05	0.08	0.05	0.05	0.07	0.00
4_2	8	11		0.05	0.05	0.07	0.05	0.05	0.07	0.00
5	9	13		0.03	0.03	0.08	0.05	0.05	0.07	0.02
6	4	5		0.00	0.02	0.02	0.00	0.02	0.02	0.02
7	6	9		0.03	0.02	0.02	0.02	0.02	0.02	0.00
8	5	7		0.02	0.02	0.02	0.03	0.02	0.03	0.00
9	13	25	$\surd$	0.03	0.07	0.07	0.05	0.07	0.05	0.00
10	14	23	$\surd$	0.13	0.17	0.23	0.22	0.15	0.53	0.02
11	3	7	$\surd$	0.00	0.00	0.00	0.02	0.00	0.00	0.02
12	18	79		0.37	0.68	0.80	0.77	0.70	0.57	0.43
12_1	16	39		0.61	0.43	0.58	0.43	0.47	0.57	0.05
12_2	16	39		0.35	0.37	0.45	0.40	0.42	0.53	0.05
13	11	27	$\surd$	0.05	0.03	0.05	0.05	0.08	0.08	0.02
14	14	39		0.05	0.07	0.10	0.12	0.12	0.17	0.02
15	9	19		0.05	0.03	0.05	0.05	0.07	0.08	0.02
16	5	7		0.00	0.02	0.02	0.03	0.02	0.02	0.02
17	18	51		0.07	0.07	0.15	0.08	0.08	0.15	0.02
$Nepr$	7	11	$\surd$	0.05	0.07	0.12	0.07	0.08	0.08	0.00
$Urq(1)$	3	7	$\surd$	0.00	0.00	0.00	0.00	0.02	0.02	0.00
$Urq(2)$	11	43		0.08	0.08	0.13	0.12	0.13	0.20	0.03
$Urq(3)$	18	187		1.88	0.58	0.93	0.38	0.40	0.62	0.05
$pig1;1$	3	3		0.00	0.00	0.00	0.00	0.00	0.00	0.00
$pig1;2$	4	7	$\surd$	0.00	0.00	0.02	0.02	0.00	0.02	0.00
$pig2;2$	11	15		0.05	0.05	0.05	0.03	0.05	0.03	0.02
$pig2;3$	23	35	$\surd$	0.12	0.10	0.13	0.12	0.12	0.12	0.07
$pig3;3$	35	53		0.22	0.28	0.17	0.25	0.47	0.13	0.15
$pig3;4$	59	95	$\surd$	0.70	0.72	0.63	0.72	0.50	0.52	0.38
$pig4;4$	79	127		0.87	0.88	0.63	0.92	0.93	0.53	0.60
$pig4;5$	119	199	$\surd$	2.03	2.03	2.25	1.98	1.95	2.05	1.53
$pig5;5$	149	249		2.98	2.92	2.22	3.13	3.07	2.05	2.17
$pig5;6$	209	359	$\surd$	6.20	6.53	7.07	6.25	6.20	6.35	4.43
Total:	–	–	–	17.53	16.83	17.82	16.80	16.80	16.38	10.25

Figure 5: Experimental Results

Problem	dag size	tree size	provable?	(1)	(2)	(3)	(4)	(5)	(6)	(C)
$Urq(1)$	3	7	$\surd$	0.00	0.02	0.00	0.00	0.00	0.00	0.00
$Urq(2)$	11	43		0.10	0.07	0.12	0.15	0.15	0.17	0.03
$Urq(3)$	18	187		1.90	0.67	0.83	0.45	0.43	0.42	0.10
$Urq(4)$	25	763		> 900	> 900	> 900	1.42	1.35	0.88	0.20
$Urq(5)$	32	3067		–	–	–	17.82	13.90	1.75	0.28
$Urq(6)$	39	12283		–	–	–	> 900	> 900	3.25	0.86
$Urq(7)$	46	49147		–	–	–	–	–	6.20	0.57
$Urq(8)$	53	196603		–	–	–	–	–	11.23	0.82
$Urq(9)$	60	786427		–	–	–	–	–	23.97	1.08

Figure 6: Urquhart's problems

give here were obtained on a SPARCstation ELC under SunOS 4.1.2, with 16 Mb main memory and 6.5 Mb swap space. The resolution of the clock is 1/60 second, and all figures were rounded to the nearest. Observe that times of 0.00 really mean less than 1/60 second. Moreover, there are some perturbations due to garbage collections in the timings: garbage collections take roughly 10% of the time, and may pause the program for as long as 0.3 to 0.5 second; vthese times are included in the timings, because it is difficult both to avoid counting them and to average out these errors (although we vcannot predict when garbage collection will occur, it nevertheless occurs roughly at the same times in two different sessions). Entries with a dash – denote absence of measurements. Entries like > 500 means that we have interrupted the prover after 500 seconds, but it didn't find any proof or any counterargument.

Observe that total timings in Figure 5 do not vary much when the set of optimizations changes. This is probably due to the fact that these formulas are not very hard to prove, for the most part. In fact, the classical prover (C), which should be more efficient than any of the intuitionistic provers, does not have a decisive edge on the latter. Therefore it seems that we have intuitionistic provers that are only little more inefficient than classical provers on the same technology.

But because these test problems are so small, this first table is likely to measure more the efficiency of searching for fully canonical proofs, and less that of a TDD implementation: indeed, sharing is not that important on small formulas. But consider formulas $Urq(n)$. These are really intricate formulas for even rather small values of n: although their dag size is linear in n, their tree size is exponential in n. Figure 6 shows that our optimized prover manages to prove $Urq(9)$ in about 24 seconds, and in fact that it proves $Urq(n)$ in time roughly proportional to the square root of the size of the formula. (Space consumption follows the same pattern, and $Urq(10)$ fails by memory overflow — more than 12 Mb needed — which confirms the fact that BDDs tend to consume lots of memory.) Observe that a *naive* tableau implementation, even taking the same permutabilities into account as we have done, would be lost for much smaller values of n. Indeed, the size of the formula is an exponential in n, and the size of a maximally expanded tableau is a double exponential in n, so $Urq(n)$ can only be shown to be intuitionistically unprovable by standard tableaux (with a canonical strategy) in double exponential time in n. We manage to show the same thing in single exponential time.

Of course, proof search in intuitionistic logic is harder than classical logic (PSPACE-complete vs. coNP-complete), and indeed our prover does not keep up with an implementation of standard BDDs in classical logic: previous experiments have shown that $Urq(n)$ was proved classically in time linear in n with BDDs [8]. The latter fact is however deceptive, in that BDDs are designed to handle equivalences well, and in fact Urquhart's formulas are amongst the easiest problems for BDDs. TDDs for classical logic are not quite as good, but are already competitive enough: continuing the table, we find that times are 1.37 for $Urq(10)$, 8.68 for $Urq(20)$, 15.2 for $Urq(25)$, 21.48 for $Urq(30)$ and 37.95 for $Urq(40)$, i.e. after a starting phase the asymptotical complexity is roughly $O(1.1^n)$. The asymptotic complexity for prover (6) is roughly $O(2^n)$, which is not bad, since the size of the formula is $O(4^n)$.

Another negative point is that the prover of the Logics Workbench [12], compared to our prover, in fact achieves better performance on our test problems. We may compare their approach with the one presented here for intuitionistic logic in this table:

	Logics Workbench	TDDs
Sequent system	Clever (Dyckhoff's system), many impermutabilities	Dumbed down, few impermutabilities
Parallelism	None	SIMD-like

A simple conclusion springs to mind: a clever logical system is better than a clever implementation strategy. It would be interesting to see how TDDs fare with a clever logical system, even if this system does not allow as much parallelism as the dumber system. We leave this as future investigation.

5 Related Works

Posegga and Schmitt [20] also used a form of tableaux encoded as BDDs. Again, paths leading to **1** in a BDD were meant to represent tableaux paths, but using standard BDDs forced them to use what is usually called tableaux with lemmata. (In tableaux with lemmata, $p, +\Phi_1 \vee \Phi_2$ is expanded into the two paths $p, +\Phi_1$ and $p, -\Phi_1, +\Phi_2$ instead of $p, +\Phi_1$ and $p, +\Phi_2$ in order to avoid redoing steps on the second paths that we've already done closing paths in the expansion of the first path.) Because their aim is first-order classical logic, they actually use shared Shannon graphs, i.e. BDDs that are neither ordered nor reduced. Consequently, they close paths one by one; speed is gained by compiling (sequential) proof search into an efficient form, such as a Prolog program. That is, they gain a rather large constant factor in speed; by parallelizing expansions and closings in TDDs, we gain a possibly much larger factor. But the main point is that their approach does not generalize to non-classical logics, at least not easily: the major roadblock is that lemmata cannot be used in non-classical logics, and this is why we have used a variant of Minato's zero-suppressed BDDs.

Looking at the intuitionistic proof procedure that we have proposed, we may be tempted to say that it implements a *uniform* proof search strategy, in the sense that it deals uniformly with all paths. This is what allows us to expand them in parallel. However, the term "uniform" is already taken in a more restricted sense in [15], where it denotes a proof strategy for fragments of intuitionistic logic related to the way that Prolog programs are executed, where right rules are systematically used first and a backchaining rule is used to encode expansions of left rules. As noticed by Galmiche and Perrier [7], uniformity in this restricted sense is only a particular case of looking for *canonical* proofs. There is no unique notion of canonical proof, and they depend on the chosen sequent system for the logic (and its combinatorics: permutabilities, invertibility), on the direction of the search (top-down as in tableaux, or bottom-up as in Maslov's inverse method [24], or even mixed) and on strategy choices. Funnily, in the system we have presented, we have estimated that the more natural canonical proofs for tableau search should apply *left* implication rules before any right implication rules, i.e. the converse of the uniform strategy.

This leads us to discuss our choice of Dummett's system for intuitionistic logic. Dyckhoff [4] presents two systems without contraction — one based on LJ, the other on Dragalin's multi-conclusioned GHPC —, which may save us the trouble of loop checking. But they have more impermutabilities, so it is not clear whether we would gain something by using his systems.

Some other systems encode the impermutabilities of the deduction system in various ways, so as to actually force new permutabilities. Shankar [22] uses skolemization and herbrandization so as to encode impermutabilities as variable dependencies: this delays the check for impermutabilities until we try to close paths, and makes the provability problem quite similar to proof-search in first-order classical logic with bounded Herbrand multiplicity. Although there is no reason to think that such a technique would be superior to the one that we have presented here, this opens up another avenue for using BDDs with non-classical logics, using for instance techniques from [10] or [19], and this is probably the only sensible way of doing it for non-classical *first-order* logics. Techniques based on encoding semantical Kripke accessibility relationships as path-expressions [17, 23] seem to pose similar problems and to offer similar solutions. A recent technique [21] is to look for intuitionistic proofs as (hidden) subproofs of classical proofs, by a clever deciphering on proof-terms for classical logic expressed in Parigot's $\lambda\mu$-calculus. Unfortunately, in the latter technique we need to decorate formulas on paths by terms; if done naively, this will require us to decorate TDD paths by terms, thus possibly destroying the amount of sharing, hence of parallelism that we had managed to obtain.

Adapting the latter technique is thus conditioned by our ability to efficiently extract proof-terms from TDD proofs, a problem that we don't know how to solve at the moment.

6 Conclusion

The results of this paper are of two kinds.

First, we have shown that BDD technology was not limited to classical logic. Although we have not used usual BDDs, we have used very similar data-structures, called TDDs, which are basically Minato's zero-suppressed BDDs with an additional closing rule. We have successfully implemented an automated theorem prover for propositional intuitionistic logic using this technology.

Second, we have shown that using TDDs was in fact an efficient enough way of implementing tableaux for classical or non-classical logic, provided that the logic of interest has a Gentzen system with many nice combinatorial properties, including permutabilities (for canonicity) and invertibility of rules (for full canonicity). The purported main source of efficiency here is the interplay between sharing in TDDs and proof combinatorics, which allows us to close and expand many paths in parallel. To put it shortly and a bit incorrectly, TDDs allow us to implement mostly parallel search strategies on sequential machines. Experiments show that using canonical proof search strategies is not detrimental to the efficiency, and that the high level of parallelism that TDDs offers allows us to prove (or show to be unprovable) formulas that are out of reach of tableaux theorem provers that look for uniform proofs, like Urquhart's formulas.

Finally, note that TDDs are not limited to classical or intuitionistic logic, but can be applied to any logic where sequents can be coded as finite sets, like most modal logics. TDDs may however be less interesting in logics that only have Gentzen systems with many impermutabilities. We believe that TDDs can also be extended to logics where sequents must be multisets, like fragments of linear logic, by having each node having a variable number of successors, each corresponding to a different number of occurrences of the literal. (For example, if $s\Phi$ does not occur in p_0, occurs once in p_1 and three times in p_2, we might represent the set of all paths by a ternary node $s\Phi(\stackrel{0}{\longrightarrow}F_0)(\stackrel{1}{\longrightarrow}F_1)(\stackrel{3}{\longrightarrow}F_3)$, where F_0 represents p_0, F_1 represents p_1, and F_3 represents p_3.) We leave it as future studies.

And even if a logic has a Gentzen system with many permutabilities, clever tableaux systems, implemented sequentially, may still be better than dumber but "parallelizable" tableaux systems. As Roy Dyckhoff once told me, logic wins. We leave the question of the performance of TDD technology on clever tableau systems to be answered in future works.

References

[1] R. E. Bryant. Graph-based algorithms for boolean functions manipulation. *IEEE Trans. Comp.*, C35(8):677–692, 1986.

[2] C.-L. Chang and R. C.-T. Lee. *Symbolic Logic and Mechanical Theorem Proving.* Computer Science Classics. Academic Press, 1973.

[3] M. Dummett. *Elements of Intuitionism.* Clarendon Press, Oxford, 1977.

[4] R. Dyckhoff. Contraction-free sequent calculi for intuitionistic logic. *Journal of Symbolic Logic*, 57(3):795–807, 1992.

[5] M. Fitting. *Proof Methods for Modal and Intuitionistic Logics*, volume 169 of *Synthese Library.* D. Reidel, Dordrecht, Holland, 1983.

[6] M. C. Fitting. *First-Order Logic and Automated Theorem Proving.* Springer Verlag, 1990.

[7] D. Galmiche and G. Perrier. Foundations of proof search strategies design in linear logic. In *International Symposium on Logical Foundations of Computer Science, Logic at St. Petersburg'94*, pages 101–113, 1994. Lecture Notes in Computer Science 813.

[8] J. Goubault. *Démonstration automatique en logique classique : complexité et méthodes.* PhD thesis, École Polytechnique, Palaiseau, France, 1993.

[9] J. Goubault. HimML: Standard ML with fast sets and maps. In *5th ACM SIGPLAN Workshop on ML and its Applications*, 1994.

[10] J. Goubault. Proving with BDDs and control of information. In A. Bundy, editor, *12th International Conference on Automated Deduction (CADE-12)*, volume 814 of *Lecture Notes in Artificial Intelligence*, Nancy, France, june–july 1994. Springer Verlag.

[11] J. Goubault. A BDD-based simplification and skolemization procedure. *Bulletin of the Special Interest Group in Pure and Applied Logics*, 3(1), June 1995. available by ftp on `theory.doc.ic.ac.uk:/home/leonardo/papers/GoubaultJ/`.

[12] A. Heuerding, G. Jger, S. Schwendimann, and M. Seyfried. Propositional logics on the computer. In P. Baumgartner, Hähnle, and J. Posegga, editors, *Tableaux'95*, pages 310–323. LNAI 918, Springer Verlag, 1995. Home page at `http://lwbwww.unibe.ch:8080/LWBinfo.html`.

[13] S. C. Kleene. Permutability of inferences in Gentzen's calculi LK and LJ. *Memoirs of the American Mathematical Society*, 1952.

[14] L. Mauborgne. Abstract interpretation using TDGs. In *First Static Analysis Symposium (SAS'94)*, pages 363–379, 1994.

[15] D. Miller, G. Nadathur, F. Pfenning, and A. Scedrov. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic*, 51:125–157, 1991.

[16] S.-I. Minato. Zero-suppressed BDDs for set manipulation in combinatorial problems. In *Proceedings of the 30th ACM/IEEE Design Automation Conference*, pages 272–277, Dallas, TX, June 1993. ACM Press.

[17] H. J. Ohlbach. A resolution calculus for modal logics. In E. Lusk and R. Overbeek, editors, *9th International Conference on Automated Deduction*, volume 310 of *Lecture Notes in Computer Science*, pages 500–516, Argonne, Illinois, USA, May 1988. Springer Verlag.

[18] F. J. Pelletier. Seventy-five problems for testing automatic theorem provers. *Journal of Automated Reasoning*, 2:191–216, 1986. errata in JAR 4:235–236, 1988.

[19] J. Posegga. *Deduktion mit Shannongraphen für Prädikatenlogik erster Stufe.* Infix Verlag, Sankt Augustin, 1993.

[20] J. Posegga and P. H. Schmitt. Implementing semantic tableaux. To be pusblished as part of the Handbook on Tableaux, 1995.

[21] E. Ritter, D. Pym, and L. Wallen. On the intuitionistic force of classical search. In *Tableaux'96*, 1996. Lecture Notes in Artificial Intelligence, Springer Verlag.

[22] N. Shankar. Proof search in the intuitionistic sequent calculus. In D. Kapur, editor, *11th International Conference on Automated Deduction*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 522–536, Saratoga Springs, New York, USA, June 1992. Springer Verlag.

[23] L. Wallen. *Automated Proof Search in Non Classical Logics: Efficient Matrix Proof Methods for Modal and Intuitionistic Logics.* PhD thesis, Edinburgh, 1987. published by MIT Press, 1989.

[24] N. Zamov. Maslov's inverse method and decidable classes. *Annals of Pure and Applied Logic*, 42:165–194, 1989.

A Proof of Figure 4 (Permutabilities)

First, the $\bot L$ column is obvious, since this rule has no premise, so no rule is in permutation position above $\bot L$.

Similarly, the $\bot L$ row is also obvious. For instance, the $(\bot L)/(\supset R)$ case is as follows:

$$\dfrac{\dfrac{}{,\ ,\ \bot, \Phi_1 \triangleright \Phi_2}\ (\bot L)}{,\ ,\ \bot \triangleright \Phi_1 \supset \Phi_2, \Delta}\ (\supset R)$$

which is transformed into:

$$\dfrac{}{,\ ,\ \bot \triangleright \Phi_1 \supset \Phi_2, \Delta}\ (\bot L)$$

Now, look at the $\bot R$ column (except for the entry in the $\bot L$ row). If R is in permutation position above $\bot R$, then either R is not $(\supset R)$ and $(\bot R)$ may then permute up, or R is $(\supset R)$, and we transform:

$$\dfrac{\dfrac{\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ ,\ ,\ \Phi_1 \triangleright \Phi_2\end{array}}{,\ \triangleright \Phi_1 \supset \Phi_2, \Delta}\ (\supset R)}{,\ \triangleright \Phi_1 \supset \Phi_2, \bot, \Delta}\ (\bot R)$$

into:

$$\dfrac{\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ ,\ ,\ \Phi_1 \triangleright \Phi_2\end{array}}{,\ \triangleright \Phi_1 \supset \Phi_2, \bot, \Delta}\ (\supset R)$$

A.1 The $\& R$ Column

The $(\supset L)/(\& R)$ case is as follows:

$$\dfrac{\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ ,\ ,\ \Phi_1 \supset \Phi_2 \triangleright \Phi_1, \Phi_3, \Delta\end{array} \quad \begin{array}{c}\pi_2\\ \vdots\\ ,\ ,\ \Phi_2 \triangleright \Phi_3, \Delta\end{array}}{,\ ,\ \Phi_1 \supset \Phi_2 \triangleright \Phi_3, \Delta}\ (\supset L) \quad \begin{array}{c}\pi_3\\ \vdots\\ ,\ ,\ \Phi_1 \supset \Phi_2 \triangleright \Phi_4, \Delta\end{array}}{,\ ,\ \Phi_1 \supset \Phi_2 \triangleright \Phi_3\ \&\ \Phi_4, \Delta}\ (\& R)$$

(or symmetrically), and we transform it into:

$$\dfrac{\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ ,\ ,\ \Phi_1 \supset \Phi_2\\ \triangleright \Phi_1, \Phi_3, \Delta\end{array} \quad \begin{array}{c}\pi_3, \Phi_1\\ \vdots\\ ,\ ,\ \Phi_1 \supset \Phi_2\\ \triangleright \Phi_1, \Phi_4, \Delta\end{array}}{,\ ,\ \Phi_1 \supset \Phi_2 \triangleright \Phi_1, \Phi_3\ \&\ \Phi_4, \Delta}\ (\& R) \quad \dfrac{\begin{array}{c}\pi_2\\ \vdots\\ ,\ ,\ \Phi_2 \triangleright \Phi_3, \Delta\end{array} \quad \begin{array}{c}[\Phi_1 \supset]\pi_3\\ \vdots\\ ,\ ,\ \Phi_2 \triangleright \Phi_4, \Delta\end{array}}{,\ ,\ \Phi_2 \triangleright \Phi_3\ \&\ \Phi_4, \Delta}\ (\& R)}{,\ ,\ \Phi_1 \supset \Phi_2 \triangleright \Phi_3\ \&\ \Phi_4, \Delta}\ (\supset L)$$

The $(\supset R)/(\& R)$ case is as follows (up to symmetries):

$$\dfrac{\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ ,\ ,\ \Phi_1 \triangleright \Phi_2\end{array}}{,\ \triangleright \Phi_1 \supset \Phi_2, \Phi_3, \Delta}\ (\supset R) \quad \begin{array}{c}\pi_2\\ \vdots\\ ,\ \triangleright \Phi_1 \supset \Phi_2, \Phi_4, \Delta\end{array}}{,\ \triangleright \Phi_1 \supset \Phi_2, \Phi_3\ \&\ \Phi_4, \Delta}\ (\& R)$$

and we transform it into:

$$(\supset R)\ \dfrac{\begin{array}{c}\pi_1\\ \vdots\\ ,\ ,\ \Phi_1 \triangleright \Phi_2\end{array}}{,\ \triangleright \Phi_1 \supset \Phi_2, \Phi_3\ \&\ \Phi_4, \Delta}$$

The $(\vee L)/(\,\&\, R)$ case is as follows (up to symmetries):

$$\begin{array}{l}(\vee L)\\ (\,\&\, R)\end{array}\ \dfrac{\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ ,\ ,\ \Phi_1 \triangleright \Phi_3, \Delta\end{array}\quad\begin{array}{c}\pi_2\\ \vdots\\ ,\ ,\ \Phi_2 \triangleright \Phi_3, \Delta\end{array}}{,\ ,\ \Phi_1 \vee \Phi_2 \triangleright \Phi_3, \Delta}\quad\begin{array}{c}\pi_3\\ \vdots\\ ,\ ,\ \Phi_1 \vee \Phi_2 \triangleright \Phi_4, \Delta\end{array}}{,\ ,\ \Phi_1 \vee \Phi_2 \triangleright \Phi_3\ \&\ \Phi_4, \Delta}$$

and we transform it into:

$$\begin{array}{l}(\,\&\, R)\\ (\vee L)\end{array}\ \dfrac{\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ ,\ ,\ \Phi_1 \triangleright \Phi_3, \Delta\end{array}\quad\begin{array}{c}[\vee\Phi_2]\pi_3\\ \vdots\\ ,\ ,\ \Phi_1 \triangleright \Phi_4, \Delta\end{array}}{,\ ,\ \Phi_1 \triangleright \Phi_3\ \&\ \Phi_4, \Delta}\quad(\,\&\, R)\ \dfrac{\begin{array}{c}\pi_2\\ \vdots\\ ,\ ,\ \Phi_2 \triangleright \Phi_3, \Delta\end{array}\quad\begin{array}{c}[\Phi_1\vee]\pi_3\\ \vdots\\ ,\ ,\ \Phi_2 \triangleright \Phi_4, \Delta\end{array}}{,\ ,\ \Phi_2 \triangleright \Phi_3\ \&\ \Phi_4, \Delta}}{,\ ,\ \Phi_1 \vee \Phi_2 \triangleright \Phi_3\ \&\ \Phi_4, \Delta}$$

The $(\vee R)/(\,\&\, R)$ case is as follows, up to symmetries:

$$\begin{array}{l}(\vee R)\\ (\,\&\, R)\end{array}\ \dfrac{\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ ,\ \triangleright \Phi_1, \Phi_2, \Phi_3, \Delta\end{array}}{,\ \triangleright \Phi_1 \vee \Phi_2, \Phi_3, \Delta}\quad\begin{array}{c}\pi_2\\ \vdots\\ ,\ \triangleright \Phi_1 \vee \Phi_2, \Phi_4, \Delta\end{array}}{,\ \triangleright \Phi_1 \vee \Phi_2, \Phi_3\ \&\ \Phi_4, \Delta}$$

and we transform it into:

$$\begin{array}{l}(\,\&\, R)\\ (\vee R)\end{array}\ \dfrac{\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ ,\ \triangleright \Phi_1, \Phi_2, \Phi_3, \Delta\end{array}\quad\begin{array}{c}\pi_2[\Phi_1 \vee \Phi_2]\\ \vdots\\ ,\ \triangleright \Phi_1, \Phi_2, \Phi_4, \Delta\end{array}}{,\ \triangleright \Phi_1, \Phi_2, \Phi_3\ \&\ \Phi_4, \Delta}}{,\ \triangleright \Phi_1 \vee \Phi_2, \Phi_3\ \&\ \Phi_4, \Delta}$$

The $(\,\&\, L)/(\,\&\, R)$ case is as follows, up to symmetries:

$$\begin{array}{l}(\,\&\, L)\\ (\,\&\, R)\end{array}\ \dfrac{\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ ,\ ,\ \Phi_1, \Phi_2 \triangleright \Phi_3, \Delta\end{array}}{,\ ,\ \Phi_1\ \&\ \Phi_2 \triangleright \Phi_3, \Delta}\quad\begin{array}{c}\pi_2\\ \vdots\\ ,\ ,\ \Phi_1\ \&\ \Phi_2 \triangleright \Phi_4, \Delta\end{array}}{,\ ,\ \Phi_1\ \&\ \Phi_2 \triangleright \Phi_3\ \&\ \Phi_4, \Delta}$$

and we transform it into:

$$\begin{array}{l}(\,\&\, R)\\ (\,\&\, L)\end{array}\ \dfrac{\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ ,\ ,\ \Phi_1, \Phi_2 \triangleright \Phi_3, \Delta\end{array}\quad\begin{array}{c}[\Phi_1\ \&\ \Phi_2]\pi_2\\ \vdots\\ ,\ ,\ \Phi_1, \Phi_2 \triangleright \Phi_4, \Delta\end{array}}{,\ ,\ \Phi_1, \Phi_2 \triangleright \Phi_3\ \&\ \Phi_4, \Delta}}{,\ ,\ \Phi_1\ \&\ \Phi_2 \triangleright \Phi_3\ \&\ \Phi_4, \Delta}$$

The $(\,\&\,R)/(\,\&\,R)$ case is as follows, up to symmetries:

$$\dfrac{\dfrac{\begin{array}{cc}\pi_1 & \pi_2\\ \vdots & \vdots\\ ,\ \triangleright\,\Phi_1,\Phi_3,\Delta & ,\ \triangleright\,\Phi_2,\Phi_3,\Delta\end{array}}{,\ \triangleright\,\Phi_1\ \&\ \Phi_2,\Phi_3,\Delta}(\,\&\,R) \quad \begin{array}{c}\pi_3\\ \vdots\\ ,\ \triangleright\,\Phi_1\ \&\ \Phi_2,\Phi_4,\Delta\end{array}}{,\ \triangleright\,\Phi_1\ \&\ \Phi_2,\Phi_3\ \&\ \Phi_4,\Delta}(\,\&\,R)$$

which we transform into:

$$\dfrac{\dfrac{\begin{array}{cc}\pi_1 & \pi_3[\,\&\,\Phi_2]\\ \vdots & \vdots\\ ,\ \triangleright\,\Phi_1,\Phi_3,\Delta & ,\ \triangleright\,\Phi_1,\Phi_4,\Delta\end{array}}{,\ \triangleright\,\Phi_1,\Phi_3\ \&\ \Phi_4,\Delta}(\,\&\,R) \quad \dfrac{\begin{array}{cc}\pi_2 & \pi_3[\Phi_1\,\&\,]\\ \vdots & \vdots\\ ,\ \triangleright\,\Phi_2,\Phi_3,\Delta & ,\ \triangleright\,\Phi_2,\Phi_4,\Delta\end{array}}{,\ \triangleright\,\Phi_2,\Phi_3\ \&\ \Phi_4,\Delta}(\,\&\,R)}{,\ \triangleright\,\Phi_1\ \&\ \Phi_2,\Phi_3\ \&\ \Phi_4,\Delta}(\,\&\,R)$$

The $(\bot R)/(\,\&\,R)$ case looks like:

$$\dfrac{\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ ,\ \triangleright\,\Phi_1,\Delta\end{array}}{,\ \triangleright\,\bot,\Phi_1,\Delta}(\bot R) \quad \begin{array}{c}\pi_2\\ \vdots\\ ,\ \triangleright\,\bot,\Phi_2,\Delta\end{array}}{,\ \triangleright\,\bot,\Phi_1\ \&\ \Phi_2,\Delta}(\,\&\,R)$$

which we transform into:

$$\dfrac{\dfrac{\begin{array}{cc}\pi_1 & \pi_2[\bot]\\ \vdots & \vdots\\ ,\ \triangleright\,\Phi_1,\Delta & ,\ \triangleright\,\Phi_2,\Delta\end{array}}{,\ \triangleright\,\Phi_1\ \&\ \Phi_2,\Delta}(\,\&\,R)}{,\ \triangleright\,\bot,\Phi_1\ \&\ \Phi_2,\Delta}(\bot R)$$

A.2 The $\&\,L$ Column

The $(\supset L)/(\,\&\,L)$ case is as follows:

$$\dfrac{\dfrac{\begin{array}{cc}\pi_1 & \pi_2\\ \vdots & \vdots\\ ,\,,\Phi_3,\Phi_4,\Phi_1\supset\Phi_2\triangleright\Phi_1,\Delta & ,\,,\Phi_3,\Phi_4,\Phi_2\triangleright\Delta\end{array}}{,\,,\Phi_3,\Phi_4,\Phi_1\supset\Phi_2\triangleright\Delta}(\supset L)}{,\,,\Phi_3\ \&\ \Phi_4,\Phi_1\supset\Phi_2\triangleright\Delta}(\,\&\,L)$$

which we transform into:

$$\dfrac{\dfrac{\begin{array}{cc}\pi_1 & \\ \vdots & \\ ,\,,\Phi_3,\Phi_4,\Phi_1\supset\Phi_2\triangleright\Phi_1,\Delta & \end{array}}{,\,,\Phi_3\ \&\ \Phi_4,\Phi_1\supset\Phi_2\triangleright\Phi_1,\Delta}(\,\&\,L) \quad \dfrac{\begin{array}{c}\pi_2\\ \vdots\\ ,\,,\Phi_3,\Phi_4,\Phi_2\triangleright\Delta\end{array}}{,\,,\Phi_3\ \&\ \Phi_4,\Phi_2\triangleright\Delta}(\,\&\,L)}{,\,,\Phi_3\ \&\ \Phi_4,\Phi_1\supset\Phi_2\triangleright\Delta}(\supset L)$$

The $(\supset R)/(\,\&\,L)$ case is as follows:

$$\dfrac{\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ ,\,,\Phi_3,\Phi_4,\Phi_1\triangleright\Phi_2\end{array}}{,\,,\Phi_3,\Phi_4\triangleright\Phi_1\supset\Phi_2,\Delta}(\supset R)}{,\,,\Phi_3\ \&\ \Phi_4\triangleright\Phi_1\supset\Phi_2,\Delta}(\,\&\,L)$$

which we simply permute into:

$$\dfrac{\dfrac{\dfrac{\pi_1}{\vdots}}{, , \Phi_3, \Phi_4, \Phi_1 \rhd \Phi_2}\,(\,\&\,L)}{, , \Phi_3\ \&\ \Phi_4, \Phi_1 \rhd \Phi_2}\,(\supset R)}{, , \Phi_3\ \&\ \Phi_4 \rhd \Phi_1 \supset \Phi_2, \Delta}$$

The $(\vee L)/(\,\&\,L)$ case is as follows:

$$\dfrac{\dfrac{\dfrac{\pi_1}{\vdots} \quad \dfrac{\pi_2}{\vdots}}{, , \Phi_1, \Phi_3, \Phi_4 \rhd \Delta \quad , , \Phi_2, \Phi_3, \Phi_4 \rhd \Delta}\,(\vee L)}{, , \Phi_1 \vee \Phi_2, \Phi_3, \Phi_4 \rhd \Delta}\,(\,\&\,L)}{, , \Phi_1 \vee \Phi_2, \Phi_3\ \&\ \Phi_4 \rhd \Delta}$$

which permutes into:

$$\dfrac{\dfrac{\dfrac{\pi_1}{\vdots}}{, , \Phi_1, \Phi_3, \Phi_4 \rhd \Delta}\,(\,\&\,L) \quad \dfrac{\dfrac{\pi_2}{\vdots}}{, , \Phi_2, \Phi_3, \Phi_4 \rhd \Delta}\,(\,\&\,L)}{, , \Phi_1, \Phi_3\ \&\ \Phi_4 \rhd \Delta}\,(\vee L)}{, , \Phi_1 \vee \Phi_2, \Phi_3\ \&\ \Phi_4 \rhd \Delta}$$

The $(\vee R)/(\,\&\,L)$ case is as follows:

$$\dfrac{\dfrac{\dfrac{\pi_1}{\vdots}}{, , \Phi_3, \Phi_4 \rhd \Phi_1, \Phi_2, \Delta}\,(\vee R)}{, , \Phi_3, \Phi_4 \rhd \Phi_1 \vee \Phi_2, \Delta}\,(\,\&\,L)}{, , \Phi_3\ \&\ \Phi_4 \rhd \Phi_1 \vee \Phi_2, \Delta}$$

which permutes into:

$$\dfrac{\dfrac{\dfrac{\pi_1}{\vdots}}{, , \Phi_3, \Phi_4 \rhd \Phi_1, \Phi_2, \Delta}\,(\,\&\,L)}{, , \Phi_3\ \&\ \Phi_4 \rhd \Phi_1, \Phi_2, \Delta}\,(\vee R)}{, , \Phi_3\ \&\ \Phi_4 \rhd \Phi_1 \vee \Phi_2, \Delta}$$

The $(\,\&\,L)/(\,\&\,L)$ is as follows:

$$\dfrac{\dfrac{\dfrac{\pi_1}{\vdots}}{, , \Phi_3, \Phi_4, \Phi_1, \Phi_2 \rhd \Delta}\,(\,\&\,L)}{, , \Phi_3, \Phi_4, \Phi_1\ \&\ \Phi_2 \rhd \Delta}\,(\,\&\,L)}{, , \Phi_3\ \&\ \Phi_4, \Phi_1\ \&\ \Phi_2 \rhd \Delta}$$

which permutes into:

$$\dfrac{\dfrac{\dfrac{\pi_1}{\vdots}}{, , \Phi_3, \Phi_4, \Phi_1, \Phi_2 \rhd \Delta}\,(\,\&\,L)}{, , \Phi_3\ \&\ \Phi_4, \Phi_1, \Phi_2 \rhd \Delta}\,(\,\&\,L)}{, , \Phi_3\ \&\ \Phi_4, \Phi_1\ \&\ \Phi_2 \rhd \Delta}$$

The $(\,\&\,R)/(\,\&\,L)$ case is as follows:

$$\dfrac{\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ ,\,,\,\Phi_3,\Phi_4 \rhd \Phi_1,\Delta\end{array}\qquad\begin{array}{c}\pi_2\\ \vdots\\ ,\,,\,\Phi_3,\Phi_4 \rhd \Phi_2,\Delta\end{array}}{,\,,\,\Phi_3,\Phi_4 \rhd \Phi_1\ \&\ \Phi_2,\Delta}\,(\,\&\,R)}{,\,,\,\Phi_3\ \&\ \Phi_4 \rhd \Phi_1\ \&\ \Phi_2,\Delta}\,(\,\&\,L)$$

which permutes into:

$$\dfrac{\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ ,\,,\,\Phi_3,\Phi_4 \rhd \Phi_1,\Delta\end{array}}{,\,,\,\Phi_3\ \&\ \Phi_4 \rhd \Phi_1,\Delta}\,(\,\&\,L)\qquad\dfrac{\begin{array}{c}\pi_2\\ \vdots\\ ,\,,\,\Phi_3,\Phi_4 \rhd \Phi_2,\Delta\end{array}}{,\,,\,\Phi_3\ \&\ \Phi_4 \rhd \Phi_2,\Delta}\,(\,\&\,L)}{,\,,\,\Phi_3\ \&\ \Phi_4 \rhd \Phi_1\ \&\ \Phi_2,\Delta}\,(\,\&\,R)$$

The $(\bot R)/(\,\&\,L)$ case is as follows:

$$\dfrac{\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ ,\,,\,\Phi_1,\Phi_2 \rhd \Delta\end{array}}{,\,,\,\Phi_1,\Phi_2 \rhd \bot,\Delta}\,(\bot R)}{,\,,\,\Phi_1\ \&\ \Phi_2 \rhd \bot,\Delta}\,(\,\&\,L)$$

which permutes into:

$$\dfrac{\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ ,\,,\,\Phi_1,\Phi_2 \rhd \Delta\end{array}}{,\,,\,\Phi_1\ \&\ \Phi_2 \rhd \Delta}\,(\,\&\,L)}{,\,,\,\Phi_1\ \&\ \Phi_2 \rhd \bot,\Delta}\,(\bot R)$$

A.3 The $\vee R$ Column

The $(\supset L)/(\vee R)$ case is as follows:

$$\dfrac{\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ ,\,,\,\Phi_1 \supset \Phi_2 \rhd \Phi_1,\Phi_3,\Phi_4,\Delta\end{array}\qquad\begin{array}{c}\pi_2\\ \vdots\\ ,\,,\,\Phi_2 \rhd \Phi_3,\Phi_4,\Delta\end{array}}{,\,,\,\Phi_1 \supset \Phi_2 \rhd \Phi_3,\Phi_4,\Delta}\,(\supset L)}{,\,,\,\Phi_1 \supset \Phi_2 \rhd \Phi_3 \vee \Phi_4,\Delta}\,(\vee R)$$

which we transform into:

$$\dfrac{\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ ,\,,\,\Phi_1 \supset \Phi_2 \rhd \Phi_1,\Phi_3,\Phi_4,\Delta\end{array}}{,\,,\,\Phi_1 \supset \Phi_2 \rhd \Phi_1,\Phi_3 \vee \Phi_4,\Delta}\,(\vee R)\qquad\dfrac{\begin{array}{c}\pi_2\\ \vdots\\ ,\,,\,\Phi_2 \rhd \Phi_3,\Phi_4,\Delta\end{array}}{,\,,\,\Phi_2 \rhd \Phi_3 \vee \Phi_4,\Delta}\,(\vee R)}{,\,,\,\Phi_1 \supset \Phi_2 \rhd \Phi_3 \vee \Phi_4,\Delta}\,(\supset L)$$

The $(\supset R)/(\vee R)$ case is as follows:

$$\dfrac{\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ ,\,\Phi_1 \rhd \Phi_2\end{array}}{,\,\rhd \Phi_1 \supset \Phi_2,\Phi_3,\Phi_4,\Delta}\,(\supset R)}{,\,\rhd \Phi_1 \supset \Phi_2,\Phi_3 \vee \Phi_4,\Delta}\,(\vee R)$$

and simplifies to:

$$(\supset R)\ \dfrac{\begin{array}{c}\pi_1\\ \vdots\\ ,\ ,\ \Phi_1 \triangleright \Phi_2\end{array}}{,\ \triangleright \Phi_1 \supset \Phi_2, \Phi_3 \vee \Phi_4, \Delta}$$

The $(\vee L)/(\vee R)$ case is as follows:

$$\begin{array}{ll}(\vee L) & \dfrac{\begin{array}{cc}\pi_1 & \pi_2\\ \vdots & \vdots\\ ,\ ,\ \Phi_1 \triangleright \Phi_3, \Phi_4, \Delta & ,\ ,\ \Phi_2 \triangleright \Phi_3, \Phi_4, \Delta\end{array}}{,\ ,\ \Phi_1 \vee \Phi_2 \triangleright \Phi_3, \Phi_4, \Delta}\\ (\vee R) & \dfrac{}{,\ ,\ \Phi_1 \vee \Phi_2 \triangleright \Phi_3 \vee \Phi_4, \Delta}\end{array}$$

which permutes into:

$$\begin{array}{ll}(\vee R) & \dfrac{\begin{array}{cc}\pi_1 & \pi_2\\ \vdots & \vdots\\ ,\ ,\ \Phi_1 \triangleright \Phi_3, \Phi_4, \Delta & ,\ ,\ \Phi_2 \triangleright \Phi_3, \Phi_4, \Delta\end{array}}{,\ ,\ \Phi_1 \triangleright \Phi_3 \vee \Phi_4, \Delta}\\ (\vee L) & \dfrac{}{,\ ,\ \Phi_1 \vee \Phi_2 \triangleright \Phi_3 \vee \Phi_4, \Delta}\end{array}$$

The $(\vee R)/(\vee R)$ case is as follows:

$$\begin{array}{ll}(\vee R) & \dfrac{\begin{array}{c}\pi_1\\ \vdots\\ ,\ \triangleright \Phi_1, \Phi_2, \Phi_3, \Phi_4, \Delta\end{array}}{,\ \triangleright \Phi_1 \vee \Phi_2, \Phi_3, \Phi_4, \Delta}\\ (\vee R) & \dfrac{}{,\ \triangleright \Phi_1 \vee \Phi_2, \Phi_3 \vee \Phi_4, \Delta}\end{array}$$

which permutes into:

$$\begin{array}{ll}(\vee R) & \dfrac{\begin{array}{c}\pi_1\\ \vdots\\ ,\ \triangleright \Phi_1, \Phi_2, \Phi_3, \Phi_4, \Delta\end{array}}{,\ \triangleright \Phi_1, \Phi_2, \Phi_3 \vee \Phi_4, \Delta}\\ (\vee R) & \dfrac{}{,\ \triangleright \Phi_1 \vee \Phi_2, \Phi_3 \vee \Phi_4, \Delta}\end{array}$$

The $(\,\&\,L)/(\vee R)$ is as follows:

$$\begin{array}{ll}(\,\&\,L) & \dfrac{\begin{array}{c}\pi_1\\ \vdots\\ ,\ ,\ \Phi_1, \Phi_2 \triangleright \Phi_3, \Phi_4, \Delta\end{array}}{,\ ,\ \Phi_1\ \&\ \Phi_2 \triangleright \Phi_3, \Phi_4, \Delta}\\ (\vee R) & \dfrac{}{,\ ,\ \Phi_1\ \&\ \Phi_2 \triangleright \Phi_3 \vee \Phi_4, \Delta}\end{array}$$

which permutes into:

$$\begin{array}{ll}(\vee R) & \dfrac{\begin{array}{c}\pi_1\\ \vdots\\ ,\ ,\ \Phi_1, \Phi_2 \triangleright \Phi_3, \Phi_4, \Delta\end{array}}{,\ ,\ \Phi_1, \Phi_2 \triangleright \Phi_3 \vee \Phi_4, \Delta}\\ (\,\&\,L) & \dfrac{}{,\ ,\ \Phi_1\ \&\ \Phi_2 \triangleright \Phi_3 \vee \Phi_4, \Delta}\end{array}$$

The $(\,\&\,R)/(\vee R)$ case is as follows:

$$\dfrac{\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ ,\ \triangleright \Phi_1, \Phi_3, \Phi_4, \Delta\end{array} \quad \begin{array}{c}\pi_2\\ \vdots\\ ,\ \triangleright \Phi_2, \Phi_3, \Phi_4, \Delta\end{array}}{,\ \triangleright \Phi_1\ \&\ \Phi_2, \Phi_3, \Phi_4, \Delta}\,(\,\&\,R)}{,\ \triangleright \Phi_1\ \&\ \Phi_2, \Phi_3 \vee \Phi_4, \Delta}\,(\vee R)$$

which permutes into:

$$\dfrac{\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ ,\ \triangleright \Phi_1, \Phi_3, \Phi_4, \Delta\end{array}}{,\ \triangleright \Phi_1, \Phi_3 \vee \Phi_4, \Delta}\,(\vee R) \quad \dfrac{\begin{array}{c}\pi_2\\ \vdots\\ ,\ \triangleright \Phi_2, \Phi_3, \Phi_4, \Delta\end{array}}{,\ \triangleright \Phi_2, \Phi_3 \vee \Phi_4, \Delta}\,(\vee R)}{,\ \triangleright \Phi_1\ \&\ \Phi_2, \Phi_3 \vee \Phi_4, \Delta}\,(\,\&\,R)$$

The $(\bot R)/(\vee R)$ case is as follows:

$$\dfrac{\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ ,\ \triangleright \Phi_1, \Phi_2, \Delta\end{array}}{,\ \triangleright \bot, \Phi_1, \Phi_2, \Delta}\,(\bot R)}{,\ \triangleright \bot, \Phi_1\ \&\ \Phi_2, \Delta}\,(\vee R)$$

which permutes into:

$$\dfrac{\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ ,\ ,\ \Phi_1, \Phi_2 \triangleright \Delta\end{array}}{,\ \triangleright \Phi_1 \vee \Phi_2, \Delta}\,(\vee R)}{,\ \triangleright \bot, \Phi_1 \vee \Phi_2, \Delta}\,(\bot R)$$

A.4 The $\vee L$ Column

The $(\supset L)/(\vee L)$ case is as follows:

$$\dfrac{\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ ,\ ,\ \Phi_3, \Phi_1 \supset \Phi_2 \triangleright \Phi_1, \Delta\end{array} \quad \begin{array}{c}\pi_2\\ \vdots\\ ,\ ,\ \Phi_3, \Phi_2 \triangleright \Delta\end{array}}{,\ ,\ \Phi_3, \Phi_1 \supset \Phi_2 \triangleright \Delta}\,(\supset L) \quad \begin{array}{c}\pi_3\\ \vdots\\ ,\ ,\ \Phi_4, \Phi_1 \supset \Phi_2 \triangleright \Delta\end{array}}{,\ ,\ \Phi_3 \vee \Phi_4, \Phi_1 \supset \Phi_2 \triangleright \Delta}\,(\vee L)$$

(or symmetrically), and we transform it into:

$$\dfrac{\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ ,\ ,\ \Phi_3, \Phi_1 \supset \Phi_2\\ \triangleright \Phi_1, \Delta\end{array} \quad \begin{array}{c}\pi_3, \Phi_1\\ \vdots\\ ,\ ,\ \Phi_4, \Phi_1 \supset \Phi_2\\ \triangleright \Phi_1, \Delta\end{array}}{,\ ,\ \Phi_3 \vee \Phi_4, \Phi_1 \supset \Phi_2 \triangleright \Phi_1, \Delta}\,(\vee L) \quad \dfrac{\begin{array}{c}\pi_2\\ \vdots\\ ,\ ,\ \Phi_3, \Phi_2 \triangleright \Delta\end{array} \quad \begin{array}{c}[\Phi_1 \supset]\pi_3\\ \vdots\\ ,\ ,\ \Phi_4, \Phi_2 \triangleright \Delta\end{array}}{,\ ,\ \Phi_3 \vee \Phi_4, \Phi_2 \triangleright \Delta}\,(\vee L)}{,\ ,\ \Phi_3 \vee \Phi_4, \Phi_1 \supset \Phi_2 \triangleright \Delta}\,(\supset L)$$

The $(\supset R)/(\vee L)$ case is as follows (up to symmetries):

$$\dfrac{\dfrac{\dfrac{\pi_1}{\vdots}}{\dfrac{,\,,\Phi_3,\Phi_1 \triangleright \Phi_2}{,\,,\Phi_3 \triangleright \Phi_1 \supset \Phi_2,\Delta}\,(\supset R)} \quad \dfrac{\dfrac{\pi_2}{\vdots}}{,\,,\Phi_4 \triangleright \Phi_1 \supset \Phi_2,\Delta}}{,\,,\Phi_3 \vee \Phi_4 \triangleright \Phi_1 \supset \Phi_2,\Delta}\,(\vee L)$$

and we would like to transform it into:

$$\dfrac{\dfrac{\dfrac{\pi_1}{\vdots}}{,\,,\Phi_3,\Phi_1 \triangleright \Phi_2} \quad \dfrac{?}{,\,,\Phi_4,\Phi_1 \triangleright \Phi_2}}{,\,,\Phi_3 \vee \Phi_4,\Phi_1 \triangleright \Phi_2}\,(\vee L)}{,\,,\Phi_3 \vee \Phi_4 \triangleright \Phi_1 \supset \Phi_2,\Delta}\,(\supset R)$$

but no proof can stand for the question mark in general, so that this pair is not permutable.

The $(\vee L)/(\vee L)$ case is as follows (up to symmetries):

$$\dfrac{\dfrac{\dfrac{\pi_1}{\vdots} \quad \dfrac{\pi_2}{\vdots}}{,\,,\Phi_3,\Phi_1 \triangleright \Delta \quad ,\,,\Phi_3,\Phi_2 \triangleright \Delta}\,(\vee L) \quad \dfrac{\dfrac{\pi_3}{\vdots}}{,\,,\Phi_4,\Phi_1 \vee \Phi_2 \triangleright \Delta}}{,\,,\Phi_3 \vee \Phi_4,\Phi_1 \vee \Phi_2 \triangleright \Delta}\,(\vee L)$$

and we transform it into:

$$\dfrac{\dfrac{\dfrac{\pi_1}{\vdots} \quad \dfrac{[\vee\Phi_2]\pi_3}{\vdots}}{,\,,\Phi_3,\Phi_1 \triangleright \Delta \quad ,\,,\Phi_4,\Phi_1 \triangleright \Delta}\,(\vee L) \quad \dfrac{\dfrac{\pi_2}{\vdots} \quad \dfrac{[\Phi_1\vee]\pi_3}{\vdots}}{,\,,\Phi_3,\Phi_2 \triangleright \Delta \quad ,\,,\Phi_4,\Phi_2 \triangleright \Delta}\,(\vee L)}{,\,,\Phi_3 \vee \Phi_4,\Phi_1 \vee \Phi_2 \triangleright \Delta}\,(\vee L)$$

The $(\vee R)/(\vee L)$ case is as follows, up to symmetries:

$$\dfrac{\dfrac{\dfrac{\pi_1}{\vdots}}{\dfrac{,\,,\Phi_3 \triangleright \Phi_1,\Phi_2,\Delta}{,\,,\Phi_3 \triangleright \Phi_1 \vee \Phi_2,\Delta}\,(\vee R)} \quad \dfrac{\dfrac{\pi_2}{\vdots}}{,\,,\Phi_4 \triangleright \Phi_1 \vee \Phi_2,\Delta}}{,\,,\Phi_3 \vee \Phi_4 \triangleright \Phi_1 \vee \Phi_2,\Delta}\,(\vee L)$$

and we transform it into:

$$\dfrac{\dfrac{\dfrac{\pi_1}{\vdots} \quad \dfrac{\pi_2[\Phi_1 \vee \Phi_2]}{\vdots}}{,\,,\Phi_3 \triangleright \Phi_1,\Phi_2,\Delta \quad ,\,,\Phi_4 \triangleright \Phi_1,\Phi_2,\Delta}\,(\vee L)}{,\,,\Phi_3 \vee \Phi_4 \triangleright \Phi_1,\Phi_2,\Delta}\,(\vee R)$$

The $(\,\&\,L)/(\vee L)$ case is as follows, up to symmetries:

$$\dfrac{\dfrac{\dfrac{\pi_1}{\vdots}}{\dfrac{,\,,\Phi_3,\Phi_1,\Phi_2 \triangleright \Delta}{,\,,\Phi_3,\Phi_1 \,\&\, \Phi_2 \triangleright \Delta}\,(\,\&\,L)} \quad \dfrac{\dfrac{\pi_2}{\vdots}}{,\,,\Phi_4,\Phi_1 \,\&\, \Phi_2 \triangleright \Delta}}{,\,,\Phi_3 \vee \Phi_4,\Phi_1 \,\&\, \Phi_2 \triangleright \Delta}\,(\vee L)$$

and we transform it into:

$$\dfrac{\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ ,\,,\,\Phi_3,\Phi_1,\Phi_2 \rhd \Delta\end{array} \quad \begin{array}{c}[\Phi_1\ \&\ \Phi_2]\pi_2\\ \vdots\\ ,\,,\,\Phi_4,\Phi_1,\Phi_2 \rhd \Delta\end{array}}{,\,,\,\Phi_3 \vee \Phi_4,\Phi_1,\Phi_2 \rhd \Delta}\,(\vee L)}{,\,,\,\Phi_3 \vee \Phi_4,\Phi_1\ \&\ \Phi_2 \rhd \Delta}\,(\&\ L)$$

The $(\&\ R)/(\vee L)$ case is as follows, up to symmetries:

$$\dfrac{\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ ,\,,\,\Phi_3 \rhd \Phi_1,\Delta\end{array} \quad \begin{array}{c}\pi_2\\ \vdots\\ ,\,,\,\Phi_3 \rhd \Phi_2,\Delta\end{array}}{,\,,\,\Phi_3 \rhd \Phi_1\ \&\ \Phi_2,\Delta}\,(\&\ R) \quad \begin{array}{c}\pi_3\\ \vdots\\ ,\,,\,\Phi_4 \rhd \Phi_1\ \&\ \Phi_2,\Delta\end{array}}{,\,,\,\Phi_3 \vee \Phi_4 \rhd \Phi_1\ \&\ \Phi_2,\Delta}\,(\vee L)$$

which we transform into:

$$\dfrac{\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ ,\,,\,\Phi_3 \rhd \Phi_1,\Delta\end{array} \quad \begin{array}{c}\pi_3[\&\ \Phi_2]\\ \vdots\\ ,\,,\,\Phi_4 \rhd \Phi_1,\Delta\end{array}}{,\,,\,\Phi_3 \vee \Phi_4 \rhd \Phi_1,\Delta}\,(\vee L) \quad \dfrac{\begin{array}{c}\pi_2\\ \vdots\\ ,\,,\,\Phi_3 \rhd \Phi_2,\Delta\end{array} \quad \begin{array}{c}\pi_3[\Phi_1\ \&\ ]\\ \vdots\\ ,\,,\,\Phi_4 \rhd \Phi_2,\Delta\end{array}}{,\,,\,\Phi_3 \vee \Phi_4 \rhd \Phi_2,\Delta}\,(\vee L)}{,\,,\,\Phi_3 \vee \Phi_4 \rhd \Phi_1\ \&\ \Phi_2,\Delta}\,(\&\ R)$$

The $(\bot R)/(\vee L)$ case looks like:

$$\dfrac{\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ ,\,,\,\Phi_1 \rhd \Delta\end{array}}{,\,,\,\Phi_1 \rhd \bot,\Delta}\,(\bot R) \quad \begin{array}{c}\pi_2\\ \vdots\\ ,\,,\,\Phi_2 \rhd \bot,\Delta\end{array}}{,\,,\,\Phi_1 \vee \Phi_2 \rhd \bot,\Delta}\,(\vee L)$$

which we transform into:

$$\dfrac{\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ ,\,,\,\Phi_1 \rhd \Delta\end{array} \quad \begin{array}{c}\pi_2[\bot]\\ \vdots\\ ,\,,\,\Phi_2 \rhd \Delta\end{array}}{,\,,\,\Phi_1 \vee \Phi_2 \rhd \Delta}\,(\vee L)}{,\,,\,\Phi_1 \vee \Phi_2 \rhd \bot,\Delta}\,(\bot R)$$

A.5 The $\supset R$ Column

In the $(\supset L)/(\supset R)$ case, we have:

$$\dfrac{\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ ,\,,\,\Phi_1 \supset \Phi_2,\Phi_3 \rhd \Phi_1,\Phi_4\end{array} \quad \begin{array}{c}\pi_2\\ \vdots\\ ,\,,\,\Phi_2,\Phi_3 \rhd \Phi_4\end{array}}{,\,,\,\Phi_1 \supset \Phi_2,\Phi_3 \rhd \Phi_4}\,(\supset L)}{,\,,\,\Phi_1 \supset \Phi_2 \rhd \Phi_3 \supset \Phi_4,\Delta}\,(\supset R)$$

which we would like to transform into:

$$\dfrac{\dfrac{\dfrac{\begin{array}{c}?\\ \vdots\\ ,\,,\,\Phi_1 \supset \Phi_2,\Phi_3 \rhd \Phi_4\end{array}}{,\,,\,\Phi_1 \supset \Phi_2 \rhd \Phi_1,\Phi_3 \supset \Phi_4,\Delta}\,(\supset R) \quad \dfrac{\begin{array}{c}\pi_2\\ \vdots\\ ,\,,\,\Phi_2,\Phi_3 \rhd \Phi_4\end{array}}{,\,,\,\Phi_2 \rhd \Phi_3 \supset \Phi_4,\Delta}\,(\supset R)}{,\,,\,\Phi_1 \supset \Phi_2 \rhd \Phi_3 \supset \Phi_4,\Delta}\,(\supset L)$$

but there is no way to find a proof in place of the question mark above.

The $(\supset R)/(\supset R)$ case cannot occur, since the only possibility for having an instance of $(\supset R)$ atop another one is:

$$\dfrac{\dfrac{\dfrac{\pi_1}{\vdots}}{, , \Phi_3, \Phi_1 \triangleright \Phi_2}\,(\supset R)}{, , \Phi_3 \triangleright \Phi_1 \supset \Phi_2}\,(\supset R)$$
$$, \triangleright \Phi_3 \supset \Phi_1 \supset \Phi_2$$

where the principal formula in the upper instance is $\Phi_1 \supset \Phi_2$, and is active in the lower one.

The $(\vee L)/(\supset R)$ case is as follows:

$$\dfrac{\dfrac{\dfrac{\pi_1}{\vdots}}{, , \Phi_1, \Phi_3 \triangleright \Phi_4} \quad \dfrac{\dfrac{\pi_2}{\vdots}}{, , \Phi_2, \Phi_3 \triangleright \Phi_4}}{, , \Phi_1 \vee \Phi_2, \Phi_3 \triangleright \Phi_4}\,(\vee L)}{, , \Phi_1 \vee \Phi_2 \triangleright \Phi_3 \supset \Phi_4, \Delta}\,(\supset R)$$

which we permute into:

$$\dfrac{\dfrac{\dfrac{\pi_1}{\vdots}}{, , \Phi_1, \Phi_3 \triangleright \Phi_4}}{, , \Phi_1 \triangleright \Phi_3 \supset \Phi_4, \Delta}\,(\supset R) \quad \dfrac{\dfrac{\dfrac{\pi_2}{\vdots}}{, , \Phi_2, \Phi_3 \triangleright \Phi_4}}{, , \Phi_2 \triangleright \Phi_3 \supset \Phi_4, \Delta}\,(\supset R)}{, , \Phi_1 \vee \Phi_2 \triangleright \Phi_3 \supset \Phi_4, \Delta}\,(\vee L)$$

The $(\vee R)/(\supset R)$ case cannot happen, for reasons similar to the $(\supset R)/(\supset R)$ case.

The $(\,\&\,L)/(\supset R)$ case is as follows:

$$\dfrac{\dfrac{\dfrac{\pi_1}{\vdots}}{, , \Phi_1, \Phi_2, \Phi_3 \triangleright \Phi_4}\,(\,\&\,L)}{, , \Phi_1 \;\&\; \Phi_2, \Phi_3 \triangleright \Phi_4}\,(\supset R)$$
$$, , \Phi_1 \;\&\; \Phi_2 \triangleright \Phi_3 \supset \Phi_4, \Delta$$

which transforms into:

$$\dfrac{\dfrac{\dfrac{\pi_1}{\vdots}}{, , \Phi_1, \Phi_2, \Phi_3 \triangleright \Phi_4}\,(\supset R)}{, , \Phi_1, \Phi_2 \triangleright \Phi_3 \supset \Phi_4, \Delta}\,(\,\&\,L)$$
$$, , \Phi_1 \;\&\; \Phi_2 \triangleright \Phi_3 \supset \Phi_4, \Delta$$

And the $(\,\&\,R)/(\supset R)$ and $(\bot R)/(\supset R)$ cases cannot happen, for reasons similar to the $(\supset R)/(\supset R)$ case.

A.6 The $\supset L$ Column

The $(\supset L)/(\supset L)$ case is either as follows:

$$\dfrac{\dfrac{\dfrac{\pi_1}{\vdots} \quad \dfrac{\dfrac{\pi_2}{\vdots}}{, , \Phi_2, \Phi_3 \supset \Phi_4 \triangleright \Phi_3, \Delta} \quad \dfrac{\dfrac{\pi_3}{\vdots}}{, , \Phi_1 \supset \Phi_2, \Phi_4 \triangleright \Delta}}{, , \Phi_1 \supset \Phi_2, \Phi_3 \supset \Phi_4 \triangleright \Phi_3, \Delta}\,(\supset L)}{, , \Phi_1 \supset \Phi_2, \Phi_3 \supset \Phi_4 \triangleright \Delta}\,(\supset L)$$

in which case we transform it into:

$$\dfrac{\dfrac{\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ , , \Phi_1 \supset \Phi_2, \Phi_3 \supset \Phi_4\\ \triangleright \Phi_1, \Phi_3, \Delta\end{array} \quad \begin{array}{c}\pi_3, \Phi_1\\ \vdots\\ , , \Phi_1 \supset \Phi_2, \Phi_4\\ \triangleright \Phi_1, \Delta\end{array}}{, , \Phi_1 \supset \Phi_2, \Phi_3 \supset \Phi_4 \triangleright \Phi_1, \Delta}(\supset L) \quad \dfrac{\begin{array}{c}\pi_2\\ \vdots\\ , , \Phi_2, \Phi_3 \supset \Phi_4\\ \triangleright \Phi_3, \Delta\end{array} \quad \begin{array}{c}[\Phi_1 \supset]\pi_3\\ \vdots\\ , , \Phi_2, \Phi_4\\ \triangleright \Delta\end{array}}{, , \Phi_2, \Phi_3 \supset \Phi_4 \triangleright \Delta}(\supset L)}{, , \Phi_1 \supset \Phi_2, \Phi_3 \supset \Phi_4 \triangleright \Delta}(\supset L)$$

or it is as follows:

$$\dfrac{\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ , , \Phi_1 \supset \Phi_2, \Phi_3 \supset \Phi_4 \triangleright \Phi_3, \Delta\end{array} \quad (\supset L) \quad \dfrac{\begin{array}{c}\pi_2\\ \vdots\\ , , \Phi_1 \supset \Phi_2, \Phi_4 \triangleright \Phi_1, \Delta\end{array} \quad \begin{array}{c}\pi_3\\ \vdots\\ , , \Phi_2, \Phi_4 \triangleright \Delta\end{array}}{, , \Phi_1 \supset \Phi_2, \Phi_4 \triangleright \Delta}}{, , \Phi_1 \supset \Phi_2, \Phi_3 \supset \Phi_4 \triangleright \Delta}(\supset L)$$

in which case we transform it into:

$$\dfrac{\dfrac{\dfrac{\begin{array}{c}\pi_1, \Phi_1\\ \vdots\\ , , \Phi_1 \supset \Phi_2, \Phi_3 \supset \Phi_4\\ \triangleright \Phi_1, \Phi_3, \Delta\end{array} \quad \begin{array}{c}\pi_2\\ \vdots\\ , , \Phi_1 \supset \Phi_2, \Phi_4\\ \triangleright \Phi_1, \Delta\end{array}}{, , \Phi_1 \supset \Phi_2, \Phi_3 \supset \Phi_4 \triangleright \Phi_1, \Delta}(\supset L) \quad \dfrac{\begin{array}{c}[\Phi_1 \supset]\pi_1\\ \vdots\\ , , \Phi_2, \Phi_3 \supset \Phi_4\\ \triangleright \Phi_3, \Delta\end{array} \quad \begin{array}{c}\pi_3\\ \vdots\\ , , \Phi_2, \Phi_4\\ \triangleright \Delta\end{array}}{, , \Phi_2, \Phi_3 \supset \Phi_4 \triangleright \Delta}(\supset L)}{, , \Phi_1 \supset \Phi_2, \Phi_3 \supset \Phi_4 \triangleright \Delta}(\supset L)$$

In the $(\supset R)/(\supset L)$ case, we have a proof of the form:

$$\dfrac{\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ , , \Phi_3 \supset \Phi_4, \Phi_1 \triangleright \Phi_2\end{array}}{, , \Phi_3 \supset \Phi_4 \triangleright \Phi_1 \supset \Phi_2, \Phi_3, \Delta}(\supset R) \quad \begin{array}{c}\pi_2\\ \vdots\\ , , \Phi_4 \triangleright \Phi_1 \supset \Phi_2, \Delta\end{array}}{, , \Phi_3 \supset \Phi_4 \triangleright \Phi_1 \supset \Phi_2, \Delta}(\supset L)$$

which we would like to transform into:

$$\dfrac{\dfrac{\begin{array}{c}\pi_1, \Phi_3\\ \vdots\\ , , \Phi_3 \supset \Phi_4, \Phi_1 \triangleright \Phi_2, \Phi_3\end{array} \quad \begin{array}{c}?\\ \vdots\\ , , \Phi_4, \Phi_1 \triangleright \Phi_2\end{array}}{, , \Phi_3 \supset \Phi_4, \Phi_1 \triangleright \Phi_2}(\supset L)}{, , \Phi_3 \supset \Phi_4 \triangleright \Phi_1 \supset \Phi_2, \Delta}(\supset R)$$

but there is no way we can instantiate the question mark above, or we have:

$$\dfrac{\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ , , \Phi_3 \supset \Phi_4 \triangleright \Phi_1 \supset \Phi_2, \Phi_3, \Delta\end{array} \quad (\supset R) \quad \dfrac{\begin{array}{c}\pi_2\\ \vdots\\ , , \Phi_4, \Phi_1 \triangleright \Phi_2\end{array}}{, , \Phi_4 \triangleright \Phi_1 \supset \Phi_2, \Delta}}{, , \Phi_3 \supset \Phi_4 \triangleright \Phi_1 \supset \Phi_2, \Delta}(\supset L)$$

which we would like to transform into:

$$\dfrac{\dfrac{\begin{array}{c}?\\ \vdots\\ , , \Phi_3 \supset \Phi_4, \Phi_1 \triangleright \Phi_2, \Phi_3\end{array} \quad \begin{array}{c}\pi_2\\ \vdots\\ , , \Phi_4, \Phi_1 \triangleright \Phi_2\end{array}}{, , \Phi_3 \supset \Phi_4, \Phi_1 \triangleright \Phi_2}(\supset L)}{, , \Phi_3 \supset \Phi_4 \triangleright \Phi_1 \supset \Phi_2, \Delta}(\supset R)$$

but there is no way we can instantiate the question mark above in general.

In the $(\vee L)/(\supset L)$ case, we have:

$$\dfrac{\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ ,\,,\Phi_1,\Phi_3\supset\Phi_4\triangleright\Phi_3,\Delta\end{array}\quad\begin{array}{c}\pi_2\\ \vdots\\ ,\,,\Phi_2,\Phi_3\supset\Phi_4\triangleright\Phi_3,\Delta\end{array}}{(\vee L)\quad ,\,,\Phi_1\vee\Phi_2,\Phi_3\supset\Phi_4\triangleright\Phi_3,\Delta}\quad\begin{array}{c}\pi_3\\ \vdots\\ ,\,,\Phi_1\vee\Phi_2,\Phi_4\triangleright\Delta\end{array}}{(\supset L)\quad ,\,,\Phi_1\vee\Phi_2,\Phi_3\supset\Phi_4\triangleright\Delta}$$

which we transform into:

$$\dfrac{\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ ,\,,\Phi_1,\Phi_3\supset\Phi_4\\ \triangleright\Phi_3,\Delta\end{array}\quad\begin{array}{c}[\vee\Phi_2]\pi_3\\ \vdots\\ ,\,,\Phi_1,\Phi_4\\ \triangleright\Delta\end{array}}{(\supset L)\quad ,\,,\Phi_1,\Phi_3\supset\Phi_4\triangleright\Delta}\quad\dfrac{\begin{array}{c}\pi_2\\ \vdots\\ ,\,,\Phi_2,\Phi_3\supset\Phi_4\\ \triangleright\Phi_3,\Delta\end{array}\quad\begin{array}{c}[\Phi_1\vee]\pi_3\\ \vdots\\ ,\,,\Phi_2,\Phi_4\\ \triangleright\Delta\end{array}}{(\supset L)\quad ,\,,\Phi_2,\Phi_3\supset\Phi_4\triangleright\Delta}}{(\vee L)\quad ,\,,\Phi_1\vee\Phi_2,\Phi_3\supset\Phi_4\triangleright\Delta}$$

or it is as follows:

$$\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ ,\,,\Phi_1\vee\Phi_2,\Phi_3\supset\Phi_4\triangleright\Phi_3,\Delta\end{array}\quad(\vee L)\quad\dfrac{\begin{array}{c}\pi_2\\ \vdots\\ ,\,,\Phi_1,\Phi_4\triangleright\Delta\end{array}\quad\begin{array}{c}\pi_3\\ \vdots\\ ,\,,\Phi_2,\Phi_4\triangleright\Delta\end{array}}{,\,,\Phi_1\vee\Phi_2,\Phi_4\triangleright\Delta}}{(\supset L)\quad ,\,,\Phi_1\vee\Phi_2,\Phi_3\supset\Phi_4\triangleright\Delta}$$

in which case we transform it into:

$$\dfrac{\dfrac{\begin{array}{c}[\vee\Phi_2]\pi_1\\ \vdots\\ ,\,,\Phi_1,\Phi_3\supset\Phi_4\\ \triangleright\Phi_3,\Delta\end{array}\quad\begin{array}{c}\pi_2\\ \vdots\\ ,\,,\Phi_1,\Phi_4\\ \triangleright\Delta\end{array}}{(\supset L)\quad ,\,,\Phi_1,\Phi_3\supset\Phi_4\triangleright\Delta}\quad\dfrac{\begin{array}{c}[\Phi_1\vee]\pi_1\\ \vdots\\ ,\,,\Phi_2,\Phi_3\supset\Phi_4\\ \triangleright\Phi_3,\Delta\end{array}\quad\begin{array}{c}\pi_3\\ \vdots\\ ,\,,\Phi_2,\Phi_4\\ \triangleright\Delta\end{array}}{(\supset L)\quad ,\,,\Phi_2,\Phi_3\supset\Phi_4\triangleright\Delta}}{(\vee L)\quad ,\,,\Phi_1\vee\Phi_2,\Phi_3\supset\Phi_4\triangleright\Delta}$$

In the $(\vee R)/(\supset L)$ case, we have:

$$\dfrac{\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ ,\,,\Phi_3\supset\Phi_4\triangleright\Phi_1,\Phi_2,\Phi_3,\Delta\end{array}}{(\vee R)\quad ,\,,\Phi_3\supset\Phi_4\triangleright\Phi_1\vee\Phi_2,\Phi_3,\Delta}\quad\begin{array}{c}\pi_2\\ \vdots\\ ,\,,\Phi_4\triangleright\Phi_1\vee\Phi_2,\Delta\end{array}}{(\supset L)\quad ,\,,\Phi_3\supset\Phi_4\triangleright\Phi_1\vee\Phi_2,\Delta}$$

which we transform into:

$$\dfrac{\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ ,\,,\Phi_3\supset\Phi_4\triangleright\Phi_1,\Phi_2,\Phi_3,\Delta\end{array}\quad\begin{array}{c}\pi_2[\Phi_1\vee\Phi_2]\\ \vdots\\ ,\,,\Phi_4\triangleright\Phi_1,\Phi_2,\Delta\end{array}}{(\supset L)\quad ,\,,\Phi_3\supset\Phi_4\triangleright\Phi_1,\Phi_2,\Delta}}{(\vee R)\quad ,\,,\Phi_3\supset\Phi_4\triangleright\Phi_1\vee\Phi_2,\Delta}$$

or we have:

$$(\supset L)\ \dfrac{\begin{array}{c}\pi_1\\ \vdots\\ , , \Phi_3 \supset \Phi_4 \triangleright \Phi_1 \vee \Phi_2, \Phi_3, \Delta\end{array}\ (\vee R)\ \dfrac{\begin{array}{c}\pi_2\\ \vdots\\ , , \Phi_4 \triangleright \Phi_1, \Phi_2, \Delta\end{array}}{, , \Phi_4 \triangleright \Phi_1 \vee \Phi_2, \Delta}}{, , \Phi_3 \supset \Phi_4 \triangleright \Phi_1 \vee \Phi_2, \Delta}$$

which we transform into:

$$\begin{array}{l}(\supset L)\\ (\vee R)\end{array}\ \dfrac{\dfrac{\begin{array}{c}\pi_1[\Phi_1 \vee \Phi_2]\\ \vdots\\ , , \Phi_3 \supset \Phi_4 \triangleright \Phi_1, \Phi_2, \Phi_3, \Delta\end{array}\quad \begin{array}{c}\pi_2\\ \vdots\\ , , \Phi_4 \triangleright \Phi_1, \Phi_2, \Delta\end{array}}{, , \Phi_3 \supset \Phi_4 \triangleright \Phi_1, \Phi_2, \Delta}}{, , \Phi_3 \supset \Phi_4 \triangleright \Phi_1 \vee \Phi_2, \Delta}$$

In the $(\,\&\,L)/(\supset L)$ case, we have:

$$\begin{array}{l}(\,\&\,L)\\ (\supset L)\end{array}\ \dfrac{\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ , , \Phi_1, \Phi_2, \Phi_3 \supset \Phi_4 \triangleright \Phi_3, \Delta\end{array}}{, , \Phi_1\ \&\ \Phi_2, \Phi_3 \supset \Phi_4 \triangleright \Phi_3, \Delta}\quad \begin{array}{c}\pi_2\\ \vdots\\ , , \Phi_1\ \&\ \Phi_2, \Phi_4 \triangleright \Delta\end{array}}{, , \Phi_1\ \&\ \Phi_2, \Phi_3 \supset \Phi_4 \triangleright \Delta}$$

which we transform into:

$$\begin{array}{l}(\supset L)\\ (\,\&\,L)\end{array}\ \dfrac{\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ , , \Phi_1, \Phi_2, \Phi_3 \supset \Phi_4 \triangleright \Phi_3, \Delta\end{array}\quad \begin{array}{c}[\Phi_1\ \&\ \Phi_2]\pi_2\\ \vdots\\ , , \Phi_1, \Phi_2, \Phi_4 \triangleright \Delta\end{array}}{, , \Phi_1, \Phi_2, \Phi_3 \supset \Phi_4 \triangleright \Delta}}{, , \Phi_1\ \&\ \Phi_2, \Phi_3 \supset \Phi_4 \triangleright \Delta}$$

or we have:

$$(\supset L)\ \dfrac{\begin{array}{c}\pi_1\\ \vdots\\ , , \Phi_1\ \&\ \Phi_2, \Phi_3 \supset \Phi_4 \triangleright \Phi_3, \Delta\end{array}\ (\,\&\,L)\ \dfrac{\begin{array}{c}\pi_2\\ \vdots\\ , , \Phi_1, \Phi_2, \Phi_4 \triangleright \Delta\end{array}}{, , \Phi_1\ \&\ \Phi_2, \Phi_4 \triangleright \Delta}}{, , \Phi_1\ \&\ \Phi_2, \Phi_3 \supset \Phi_4 \triangleright \Delta}$$

which we transform into:

$$\begin{array}{l}(\supset L)\\ (\,\&\,L)\end{array}\ \dfrac{\dfrac{\begin{array}{c}[\Phi_1\ \&\ \Phi_2]\pi_1\\ \vdots\\ , , \Phi_1, \Phi_2, \Phi_3 \supset \Phi_4 \triangleright \Phi_3, \Delta\end{array}\quad \begin{array}{c}\pi_2\\ \vdots\\ , , \Phi_1, \Phi_2, \Phi_4 \triangleright \Delta\end{array}}{, , \Phi_1, \Phi_2, \Phi_3 \supset \Phi_4 \triangleright \Delta}}{, , \Phi_1\ \&\ \Phi_2, \Phi_3 \supset \Phi_4 \triangleright \Delta}$$

In the $(\,\&\,R)/(\supset L)$ case, we have:

$$\begin{array}{l}(\,\&\,R)\\ (\supset L)\end{array}\ \dfrac{\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ , , \Phi_3 \supset \Phi_4 \triangleright \Phi_3, \Phi_1, \Delta\end{array}\quad \begin{array}{c}\pi_2\\ \vdots\\ , , \Phi_3 \supset \Phi_4 \triangleright \Phi_3, \Phi_2, \Delta\end{array}}{, , \Phi_3 \supset \Phi_4 \triangleright \Phi_3, \Phi_1\ \&\ \Phi_2, \Delta}\quad \begin{array}{c}\pi_3\\ \vdots\\ , , \Phi_4 \triangleright \Phi_1\ \&\ \Phi_2, \Delta\end{array}}{, , \Phi_3 \supset \Phi_4 \triangleright \Phi_1\ \&\ \Phi_2, \Delta}$$

which we transform into:

$$\dfrac{\dfrac{\dfrac{\pi_1 \quad \pi_3[\,\&\,\Phi_2]}{\dfrac{,\,,\Phi_3 \supset \Phi_4 \quad ,\,,\Phi_4}{\triangleright\Phi_3,\Phi_1,\Delta \quad \triangleright\Phi_1,\Delta}}{,\,,\Phi_3 \supset \Phi_4 \triangleright \Phi_1,\Delta}(\supset L) \quad \dfrac{\dfrac{\pi_2 \quad \pi_3[\Phi_1\,\&\,]}{\dfrac{,\,,\Phi_3 \supset \Phi_4 \quad ,\,,\Phi_4}{\triangleright\Phi_3,\Phi_2,\Delta \quad \triangleright\Phi_2,\Delta}}{,\,,\Phi_3 \supset \Phi_4 \triangleright \Phi_2,\Delta}(\supset L)}{,\,,\Phi_3 \supset \Phi_4 \triangleright \Phi_1 \;\&\; \Phi_2,\Delta}(\&\,R)$$

or it is as follows:

$$\dfrac{\dfrac{\pi_1}{,\,,\Phi_3 \supset \Phi_4 \triangleright \Phi_3,\Phi_1 \;\&\; \Phi_2,\Delta}(\supset L) \quad \dfrac{\dfrac{\pi_2 \quad \pi_3}{,\,,\Phi_4 \triangleright \Phi_1,\Delta \quad ,\,,\Phi_4 \triangleright \Phi_2,\Delta}}{,\,,\Phi_4 \triangleright \Phi_1 \;\&\; \Phi_2,\Delta}(\&\,R)}{,\,,\Phi_3 \supset \Phi_4 \triangleright \Phi_1 \;\&\; \Phi_2,\Delta}$$

in which case we transform it into:

$$\dfrac{\dfrac{\dfrac{\pi_1[\,\&\,\Phi_2] \quad \pi_2}{\dfrac{,\,,\Phi_3 \supset \Phi_4 \quad ,\,,\Phi_4}{\triangleright\Phi_3,\Phi_1,\Delta \quad \triangleright\Phi_1,\Delta}}{,\,,\Phi_3 \supset \Phi_4 \triangleright \Phi_1,\Delta}(\supset L) \quad \dfrac{\dfrac{\pi_1[\Phi_1\,\&\,] \quad \pi_3}{\dfrac{,\,,\Phi_3 \supset \Phi_4 \quad ,\,,\Phi_4}{\triangleright\Phi_3,\Phi_2,\Delta \quad \triangleright\Phi_2,\Delta}}{,\,,\Phi_3 \supset \Phi_4 \triangleright \Phi_2,\Delta}(\supset L)}{,\,,\Phi_3 \supset \Phi_4 \triangleright \Phi_1 \;\&\; \Phi_2,\Delta}(\&\,R)$$

Finally, the $(\bot R)/(\supset L)$ case is as follows:

$$\dfrac{\dfrac{\pi_1}{\dfrac{,\,,\Phi_1 \supset \Phi_2 \triangleright \Phi_1,\Delta}{,\,,\Phi_1 \supset \Phi_2 \triangleright \Phi_1,\bot,\Delta}(\bot R) \quad \dfrac{\pi_2}{,\,,\Phi_2 \triangleright \bot,\Delta}}{,\,,\Phi_1 \supset \Phi_2 \triangleright \bot,\Delta}(\supset L)$$

which transforms into:

$$\dfrac{\dfrac{\pi_1 \quad \pi_2[\bot]}{\dfrac{,\,,\Phi_1 \supset \Phi_2 \triangleright \Phi_1,\Delta \quad ,\,,\Phi_2 \triangleright \Delta}{,\,,\Phi_1 \supset \Phi_2 \triangleright \Delta}(\supset L)}{,\,,\Phi_1 \supset \Phi_2 \triangleright \bot,\Delta}(\bot R)$$

or:

$$\dfrac{\dfrac{\pi_1}{,\,,\Phi_1 \supset \Phi_2 \triangleright \Phi_1,\bot,\Delta}(\bot R) \quad \dfrac{\dfrac{\pi_2}{,\,,\Phi_2 \triangleright \Delta}}{,\,,\Phi_2 \triangleright \bot,\Delta}}{,\,,\Phi_1 \supset \Phi_2 \triangleright \bot,\Delta}(\supset L)$$

which transforms into:

$$\dfrac{\dfrac{\pi_1[\bot] \quad \pi_2}{\dfrac{,\,,\Phi_1 \supset \Phi_2 \triangleright \Phi_1,\Delta \quad ,\,,\Phi_2 \triangleright \Delta}{,\,,\Phi_1 \supset \Phi_2 \triangleright \Delta}(\supset L)}{,\,,\Phi_1 \supset \Phi_2 \triangleright \bot,\Delta}(\bot R)$$