

# On the Coverage Property of a Derivation Compression Algorithm

Robinson C. de M.B. Filho<sup>1</sup>, Jefferson de B. Santos<sup>2</sup>, Edward Hermann Haeusler<sup>1</sup>

<sup>1</sup> Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio)  
Rio de Janeiro, RJ – Brazil

<sup>2</sup>Fundação Getúlio Vargas (FGV)  
Rio de Janeiro, RJ – Brazil

{rfilho, hermann}@inf.puc-rio.br, jefferson.santos@fgv.br

**Abstract.** We further elaborate, with a short example, on the set of compression rules and the derivation compression algorithm presented in [Haeusler et al. 2023]. We also argue a proof, done with the Lean theorem prover, that this algorithm obtains a dag-like compressed derivation from any tree-like Natural Deduction derivation in Minimal Purely Implicational Logic.

## 1. Introduction

In [Haeusler et al. 2023], the authors provide an algorithm, named **HC**, for obtaining a compressed dag-like proof, meaning a proof represented by a **Directed Acyclic Graph**, of any purely implicational minimal tautology. This dag-like proof has more decoration elements and labels than regular proofs in purely implicational minimal logic ( $M_{\supset}$ ) and, using these elements, a verification that the dag-like proof is valid can be done in polynomial time [Haeusler et al. 2023]. The authors named this type of dag-like proof a **Dag-Like Derivability Structure (DLDS)**, which is defined in [Haeusler et al. 2023] and also in accordance to the following definitions below, which we write down here in order to have a more self-contained document.

In any  $M_{\supset}$  Natural Deduction (ND) derivation, any application of an  $\supset$ -Intro rule has a way to indicate which formula occurrences are discharged by the application of this  $\supset$ -Intro application. One way to formalize this indication is to add edges (discharging edges) linking the conclusion of the rule applies to each discharged formula occurrence in the derivation tree that represents the ND derivation<sup>1</sup>. This may be a convenient representation in many formulations of ND. However, to not crowd our dag-like derivations with unnecessary edges, we drop out the discharging edges by assigning to each deduction edge the string of bits that represents the set of assumptions from which the formula that labels the target of this deduction edge depends on. This is formalized in the sequel.

»**Definition 1.** Let  $\alpha$  be an implicational formula,  $Sub(\alpha)$  the set of subformulas of  $\alpha$ , and  $\mathcal{O}(\alpha) = \{\beta_0, \beta_1, \dots, \beta_k\}$  a linear ordering on  $Sub(\alpha)$ . A bit-string on  $\mathcal{O}(\alpha)$  is any string  $b_0b_1 \dots b_k$ , such that  $b_i \in \{0, 1\}$ , for each  $i = 0, 1, \dots, k$ .

There is a bijective correspondence between bit-strings on  $\mathcal{O}(\alpha)$  and sets of subformulas of  $\alpha$ , given by  $Set(b_0b_1 \dots b_k) = \{\beta_i / b_i = 1\}$ . The bit-string on  $\mathcal{O}(\alpha)$  will drop out the

---

<sup>1</sup>It can be noted that assigning unique marks (numbers, for example) to each formula occurrence in a derivation and attach to each  $\supset$ -Intro application the set of marks associated to the set of its discharged formula occurrences is equivalent to add edges indicating these discharges.

discharging edges and explicitly explain the formula dependencies in a derivation. The set of bit-strings on  $\mathcal{O}(\alpha)$  is denoted by  $Bits(\alpha, \mathcal{O}(\alpha))$ . The inverse function of  $Set$  is well-defined for a fixed ordering on  $Sub(\alpha)$  and is denoted by  $Set^{-1}$ . The set of all bit-strings on a set  $S$ , under ordering  $\mathcal{O}_S$  is denoted by  $\mathcal{B}(\mathcal{O}_S)$ .

The following definition argues that the set of theorems is unchanged even when considering a restricted form of  $\supset$ -Intro rules (greedy). This restricted form of an  $\supset$ -Intro provides a sound way to remove all the discharging edges from tree-like proofs.

»**Definition 2.** Consider a derivation  $\Pi$  of  $\beta$  having  $\Delta$  as assumptions. Let  $\alpha \in \Delta$  be an (open) formula assumption in  $\Pi$ . Applying an  $\supset$ -Intro in  $\Pi$  is greedy, iff it produces  $\alpha \supset \beta$  as conclusion and discharges in  $\Pi$  every open occurrence of  $\alpha$  from which its premise  $\beta$  depends on.

Applying an  $\supset$ -Intro to a tree-like derivation is greedy iff its corresponding application in a ND derivation is also greedy. We reaffirm the terminology used in [Prawitz 1965] that proofs are derivations with no open assumption. In a proof, every hypothesis is discharged by some  $\supset$ -intro rule application, and hence any branch is a complete sub-derivation (sub-tree). The function  $G$  below converts a given proof in  $M_{\supset}$  into a greedy proof in  $M_{\supset}$ , where  $\supset$ -greedy denotes a greedy application of  $\supset$ -intro rule, i.e., all open assumptions  $\beta_1$ , if any, in  $\Pi'$  are discharged by the  $\supset$ -greedy. The reader should observe that if there is no open assumption to discharge, then this discharge is vacuous.

$$G(\Pi) = \begin{cases} \alpha & \text{if } \Pi = \alpha \\ G(\Pi') & \text{if } \Pi = \frac{\Pi'}{\beta_1 \supset \beta_2} \supset\text{-intro} \\ \frac{\beta_1 \supset \beta_2}{G(\Pi_1)} \supset\text{-greedy} & \text{if } \Pi = \frac{\Pi_1}{\beta} \supset\text{-Elim} \\ \frac{G(\Pi_2)}{\beta} \supset\text{-Elim} & \text{if } \Pi = \frac{\Pi_2}{\beta} \supset\text{-Elim} \end{cases}$$

In [Haeusler et al. 2023], the authors demonstrate the following lemma about the preservation of the completeness of greedy proofs:

»**Lemma 1** (Greedy  $\supset$ -Intro is Complete). Let  $\Pi$  be a proof of a  $M_{\supset}$  formula  $\alpha$ . If  $\Pi' = G(\Pi)$ , then  $\Pi'$  is also a valid proof of  $\alpha$  in  $M_{\supset}$ .

In [Haeusler et al. 2023] (sections 3 and 5), in a series of hand-proven results, the authors show that any ND greedy proof of an  $M_{\supset}$  formula  $\alpha$  can be mapped to a **DLDS** having its root labelled with  $\alpha$ . This **DLDS** represents the underlying tree of the ND greedy proof, but instanced with all the decorations that a **DLDS** needs. The tree-like dependency of the ND greedy proof is also mapped into this **DLDS** proof.

In [Haeusler et al. 2023], the authors also argues that **HC**, when applied to any **DLDS** proof, preserves the logical information provided by the decorations used in the **DLDS**, resulting in the preservation of soundness of the proof. This demonstration is lengthy, however, and must go through multiple cases and the numerous rules of **HC**. A computer-assisted proof seems to be the most appropriate way of proving the soundness of these rules. Thus, this article has two focuses: to elaborate and exemplify on the working of **HC**; and to present the beginnings of a proof of the soundness preservation of **HC** done with the Lean interactive theorem prover [de Moura et al. 2023].

For our purposes, **HC** takes as input an arbitrary **DLDS**, defined as follows:

»**Definition 3** (Dag-Like Derivability Structures). Let  $\Gamma$  be a set of  $M_{\supset}$  formulas,  $\mathcal{O}_{\Gamma}$  an arbitrary linear ordering on  $\Gamma$ , such that  $n > 0$ , for  $n \in \mathcal{O}_{\Gamma}$ , and  $\mathcal{O}_{\Gamma}^0 = \mathcal{O}_{\Gamma} \cup \{0, \lambda\}$ . A **Dag-Like Derivability Structure**, is a tuple  $\langle V, (E_D^i)_{i \in \mathcal{O}_{\Gamma}^0}, E_A, r, l, L, P \rangle$ , where:

- $V$  is a non-empty set of nodes;
- For each  $i \in \mathcal{O}_{\Gamma}^0$ ,  $E_D^i \subseteq V \times V$  is the family of sets of edges of deduction;
- $E_A \subseteq V \times V$  is the set of edges of ancestry;
- $r \in V$  is the root of the **DLDS**;

- $l : V \rightarrow \Gamma$  is a function, such that, for  $v \in V$ ,  $l(v)$  is the (formula) label of  $v$ ;
- $L : \bigcup_{i \in \mathcal{O}_\Gamma^0} E_D^i \rightarrow \mathcal{B}(\mathcal{O}_S)$  is a function, such that, for every  $\langle u, v \rangle \in E_D^i$ ,  $L(\langle u, v \rangle)$  is the bitstring representing from which formulas the  $i$ -th colored deduction edge  $\langle u, v \rangle$  carries its dependency;
- $P : E_A \rightarrow \{1, \dots, \|\Gamma\|\}^*$ , such that, for every  $e \in E_A$ ,  $P(e)$  is a string of the form  $o_1; \dots; o_n$ , where each  $o_i$ ,  $i = 1, \dots, n$ , is an ordinal in  $\mathcal{O}_\Gamma$ .

For each  $i \in \mathcal{O}_\Gamma^0$  and  $\langle u, v \rangle \in E_D^i$ ,  $i$  is called the color of the edge  $\langle u, v \rangle$ . Each deduction edge is coloured with formulas from  $\Gamma$  or the colour 0. The colours are introduced every time a collapsing of nodes, as explained in [Haeusler et al. 2023], is performed. Tree-like greedy derivations have only 0 coloured deduction edges. **DLDS**s obtained from Tree-like greedy derivations by effectively collapsing vertexes, sometimes edges, have coloured deduction edges. Not all **DLDS** is in the image of the function that maps Tree-like derivations into **DLDS**, but any ND (usually tree-like) derivation of a formula  $\alpha$  can be seen as a **DLDS**, as shown in [Haeusler et al. 2023]. It is also worthy of note that every **DLDS** having  $E_A$  empty and only  $E_D^i$  for  $i = 0$ , i.e., a **DLDS** without ancestral edges and only with 0-coloured deductive edges, is a greedy tree-like derivation<sup>2</sup>.

## 2. An Example of Derivation Compression

A **ND** proof of a  $M_\supset$  tautology can end in a series of  $\supset$ -Intro rule applications, and it must if we consider normal proofs. The **HC** algorithm collapses equal formulas in each level from the bottom up and left to right. We cannot compress the final part with  $\supset$ -Intro only since it already has one formula by level. So, to show smaller ND derivations, we consider derivations without the final  $\supset$ -Intro only part. For example, the derivation in figure 1 comes from a proof of  $A_1 \supset A_2 \supset (A_1 \supset (A_2 \supset A_3)) \supset (A_2 \supset (A_3 \supset A_4)) \supset (A_3 \supset (A_4 \supset A_5)) \supset (A_1 \supset A_5)$  that had the final  $\supset$ -Intro part removed.

We use bitstrings to represent subsets of a linearly ordered finite set. According this, the subset  $\{A \supset B, B \supset C, C\}$  of the the ordered set  $\{A, B, C, A \supset B, B \supset C\}$ , with order  $A \prec B \prec C \prec A \supset B \prec B \supset C$ , is represented by 00111. The formal definition of this representation is in definition 4.

The derivation in figure 1 is greedy, i.e., all  $\supset$ -Intro rule applications concluding  $\alpha \supset \beta$  discharge every possible occurrence of  $\alpha$  that is a hypothesis of the derivation of its premiss,  $\beta$ . We can represent ND greedy derivations as labelled trees. The tree nodes are labelled by the formulas in the derivation, and the edges are labelled by bitstrings that represent sets of formulas. In Fig. 2a we consider the following linear ordering  $\prec$  on the set of formulas that are in figure 1:

$$A_1 \prec A_2 \prec A_3 \prec A_4 \prec A_5 \prec A_1 \supset A_2 \prec A_2 \supset A_3 \prec A_4 \supset A_5 \prec A_1 \supset A_5 \prec A_1 \supset (A_2 \supset A_3) \prec A_2 \supset (A_3 \supset A_4) \prec A_3 \supset (A_4 \supset A_5)$$

We can choose any linear order to represent ND derivation in the form of trees. The (fixed) linear order is used to represent sets of formulas as bitstrings. The linear order must be taken as part of the representation of the greedy ND tree representation. For example, the set  $\{A_1, A_1 \supset A_2, A_1 \supset (A_2 \supset A_3)\}$  is represented by the bitstring 100001000100.

»**Definition 4.** Let  $L$  be a set of formulas in  $M_\supset$  and  $\mathcal{O}(\alpha)$  be a linear order  $\mathcal{O}(L) = \{\beta_0, \beta_1, \dots, \beta_k\}$ . A bitstring on  $\mathcal{O}(L)$  is any string  $b_0 b_1 \dots b_k$ , such that  $b_i \in \{0, 1\}$ , for each  $i = 1 \dots k$ . There is a bijective correspondence between bitstrings on  $\mathcal{O}(\alpha)$  and subsets of  $L$ , given by  $Set(b_0 b_1 \dots b_k) = \{\beta_i / b_i = 1\}$ .

<sup>2</sup>Greedy tree-like derivations are defined in [Haeusler et al. 2023].

We can use HC to compress the derivation. The algorithm obtains smaller representations of proofs and derivations in minimal implicational logic by applying transformation rules to a given greedy derivation or proof that must be provided as a labelled tree, as in Fig.2a [↗](#). The rules are applied deterministically, bottom-up and left to right. The purpose of each rule is to collapse redundant parts of the derivation. They collapse nodes that have the same label in the same level. For example, the formula  $A_3$  appears twice in level<sup>3</sup> 3. These two occurrences of  $A_3$  are conclusions of identical derivations. Moreover, in level 4 three repeated occurrences of  $A_2$  can be collapsed too. It first collapses these two lower occurrences of the formula  $A_3$  using the rule depicted in Fig.2b [↗](#), whose official name is **ROEE**. It matches the tree in level 3 according to the markings in Fig.2c [↗](#). The nodes labelled with  $A_3$  match with  $u$  and  $v$ , in the rule, respectively, and their respective children  $p_1$  and  $p_4$ , and,  $p_2$ ,  $p_3$  match with the premisses in the rule accordingly.

All the compression rules of HC are deterministic and used to define a rewriting operation,  $HCom(u, v)$ , that collapses two nodes,  $u$  and  $v$  that are labeled with the same formula. Each rule has a context provided by pattern matching and applies to a specific graph, afterwards defined as a **DLDS**<sup>4</sup>  $\mathcal{D}$ , by the matching of its left-hand side.

The rules are named as  $Rim_l m_r$ , where  $i = 0 \dots 3$  and  $m_l, m_r \in \{I, E, H, X\}$ , such that  $i$  is the type of the rule itself and  $m_l/m_r$  is type of the left/right vertex to be collapsed. For example, in Fig.2b [↗](#), the rule named **ROEE**, collapses the conclusion of an application of the  $\supset$ -Elim ND rule with the conclusion of another application of the  $\supset$ -Elim ND rule. There are also small variations in this naming convention, with rules such as  $R_v 2m_l m_r$  and  $R_e 2m_l m_r$ . The indexes  $v$  and  $e$  indicate that the respective rule collapses only vertexes ( $v$ ) or vertexes and edges ( $e$ ). Rules **R<sub>v</sub>2EE**, Fig.3b [↗](#), and **R<sub>e</sub>2EE** depict these examples. In this introductory example it is not important to understand what a **DLDS** is in detail, but the fact that they represent derivations, possibly in the form of a *DAG*, Directed Acyclic Graph. in the naming rule scheme, the  $X$  represents that the formula is conclusion of more than one rule at the same time, what is necessarily a consequence of a previous collapse.

We use **ROEE** to show how to read the pictorial representation of **HC**-rules. The left and the right-hand sides are subgraphs of, respectively, two **DLDSs**,  $\mathcal{D}$  and  $\mathcal{D}'$ . This rule says that we must replace the subgraph represented by the left-hand side by the right-hand side graph in  $\mathcal{D}$ , resulting in  $\mathcal{D}'$ , defined below, where  $\bullet_a$  is the left  $\bullet$  in the figure, while  $\bullet_b$  is the right one. In **ROEE** left-hand side,  $p_i$ ,  $i = 1, 3$ ,  $u$  and  $v$  are different nodes in the subgraph, such that  $v$  and  $u$  have the same label(formula), the black arrows are deductive edges, which have as labels the bitstring representing the dependency set denoted by  $L$ . For example,  $L(\langle p_1, u \rangle) = \bar{c}_1$  shows that the deductive edge  $\langle p_1, u \rangle \in E_D^0$  is labeled by the dependency set  $Sets(\bar{c}_1)$ . The absence of a label on an edge indicates that it is irrelevant to the pattern. A label's node is  $\bullet$  whenever it is not relevant what is its label to read the rule. In this case, the  $\bullet$  is also used to denote the node.  $\bullet$ 's label different nodes unless explicitly stated by the rule. In Fig.2b [↗](#), the bullets label different nodes. In the set-theoretical semantics of the rules, see [Haeusler et al. 2023]; we use  $\bullet_a$  and  $\bullet_b$  as a reference to each of the two different nodes. Edges that belong to  $E_D^i$  have the colour  $i$ ;

---

<sup>3</sup>The root is in level zero

<sup>4</sup>Dag-Like Decorated Structure

$$\begin{array}{c}
\frac{[A_1] \quad A_1 \supset A_2}{A_2} \quad \frac{[A_1] \quad A_1 \supset (A_2 \supset A_3)}{A_2 \supset A_3} \quad \frac{[A_1] \quad A_1 \supset A_2}{A_2} \quad \frac{[A_1] \quad A_1 \supset (A_2 \supset A_3)}{A_2 \supset (A_3 \supset A_4)} \quad \frac{[A_1] \quad A_1 \supset A_2}{A_2} \quad \frac{[A_1] \quad A_1 \supset (A_2 \supset A_3)}{A_2 \supset A_3} \\
\frac{A_2 \quad A_2 \supset A_3}{A_3} \quad \frac{A_2 \quad A_2 \supset (A_3 \supset A_4)}{A_3 \supset A_4} \quad \frac{A_2 \quad A_2 \supset A_3}{A_3} \quad \frac{A_2 \quad A_2 \supset (A_3 \supset A_4)}{A_3 \supset (A_4 \supset A_5)} \\
\frac{A_3 \quad A_3 \supset A_4}{A_4} \quad \frac{A_3 \quad A_3 \supset (A_4 \supset A_5)}{A_4 \supset A_5} \\
\frac{A_4 \quad A_4 \supset A_5}{A_5} \\
\frac{A_5}{A_1 \supset A_5}
\end{array}$$

**Figure 1. Deriving  $A_1 \supset A_5$  from  $A_1 \supset A_2$ ,  $A_1 \supset (A_2 \supset A_3)$ ,  $A_2 \supset (A_3 \supset A_4)$ , and  $A_3 \supset (A_4 \supset A_5)$**

this is the red ordinal<sup>5</sup> number  $1, \dots, n$  on a black deduction edge. The members of  $E_A$ , the ancestor edges, are coloured blue, and their labels under  $P$  labelling function are red in the picture. For example,  $\langle \bullet, p_1 \rangle \in E_A$  and  $P(\langle \bullet, p_1 \rangle) = 1$ . Moreover, we have that  $\langle u, \bullet \rangle \in E_D^1$  in the graph on the right-hand side of **ROEE**. A **DLDS**  $\mathcal{D}$  is specified by a set of nodes  $V$ , and multiple sets of deductive edges,  $E_D^i \subseteq V \times V$ ,  $i = 0, n$ , called coloured edges, and a set of ancestor edges  $E_A \subseteq V \times V$ , plus some labelling functions. The definitive definition is in the following section. The definition of a valid **DLDS** is more involving, and is presented at [Haeusler et al. 2023]. We will find the label  $\lambda$  assigned to some edges, i.e., those belonging to  $E_D^\lambda$ . The members of  $E_D^\lambda$  are edges that must have the dependency set calculated by the validation verification algorithm that verifies if a **DLDS** is valid [Haeusler et al. 2023]. Moreover, a node on the left-hand side must show all the edges, outgoing or incoming. If no incoming edge is drawn, then the node should not appear in the rule. In a ND derivation, the correct and logical reading starts in the hypothesis and follows down the conclusion analysing the changing of dependency sets obtained by each rule application. The path downwards is the path that links a node to its parent, the latter to its respective parent and so on. In a **dag**, the correct reading is a bit more involving. The result of the application of **ROEE**, in Fig. 2b [↗](#), to the match in Fig. 2c [↗](#) is depicted in Fig. 2d [↗](#). The matching is represented by ellipses around the nodes that are bullets in the rule (**ROEE**) representation instead of the original rectangles. After the application, the matching is still present to us appreciate the effect of the application of **ROEE**. In Fig. 3a [↗](#), we remove this focus by defocusing and show the derivation resulting from rule **ROEE** application, ready to be used in a new matching by a new rule applies. We must observe that the graph in this figure, resulting from a node application collapse, is no longer a tree. It is represented by a **dag**. The collapsed node labelled by the formula  $A_3$  has out-degree 2, i.e., has two outgoing edges labelled with 1 and 2, in red, respectively, besides their respective dependency sets as bitstrings. Labels 1 and 2 inform which path a derivation validation algorithm should follow to have the correct logical dependency from the conclusion to the hypothesis. The path that must be followed from a given node  $v$  to obtain the correct reading is given by the label of the ancestor edge incident in  $v$ . If there is no ancestor edge incident in a node, then the correct reading is to follow the parenthood path. For example, in Fig. 2d [↗](#), the list  $[0, 2]$ , in red, labels the edges that go from the node labelled with  $A_4 \supset A_5$  to the labelled nodes  $A_2$  and  $A_2 \supset A_3$ , respectively.  $[0, 2]$  is the path to follow from  $A_2$  to  $A_4 \supset A_5$  passing by  $A_3$ . The same can be said about the path from  $A_2 \supset A_3$  to  $A_4 \supset A_5$ . These two paths are the only ones present in the tree-like derivation from these two nodes, labelled with  $A_2$  and  $A_2 \supset A_3$  to  $A_4 \supset A_5$ , respectively. Hence, the ancestor edges drive the correct reading of the dag-like derivation regarding the original tree-like derivation before applying the collapsing rules.

<sup>5</sup>Remember that we can use the formulas themselves as ordinal numbers in this case since they are linearly ordered



A reading algorithm reads and validates a dag-like derivation, [Haeusler et al. 2023]. We can observe that the correct paths are preserved after the **ROEE** application. This is the soundness of the **ROEE** application.

The following collapse from the bottom up involves three  $A_2$  in level 4. The HC algorithm always deals with levelled dag-like derivations. The rule that applies considers the first two occurrences of  $A_2$  from left to right in the dag. They are the leftmost  $A_2$  and who is the minor premiss of a  $\supset$ -Elim with  $A_2 \supset (A_3 \supset A_4)$  as major premiss. The rule  $\mathbf{R}_v2\mathbf{EE}$ , in Fig. 3b, is used, and it is matching to the current dag-like deduction, in Fig. 3a, shows up in Fig. 4a. Fig. 4b is a graphical rearrangement of the same dag-like derivation to have a more evident matching of  $\mathbf{R}_v2\mathbf{EE}$ , in Fig. 4a. In Fig. 4c, we show the subsequent result of the application of rule  $\mathbf{R}_v2\mathbf{EE}$ . From this point on the rules  $\mathbf{R}_e2\mathbf{XE}$ , in Fig. 3c and  $\mathbf{R}_e2\mathbf{EE}$ , see [Haeusler et al. 2023], provide the derivation depicted in Fig. 5b. Due to space limitations,  $\mathbf{R}_e2\mathbf{EE}$  is not shown in this article. Please, see [Haeusler et al. 2023] for all other rules.

The dag-like derivation in Fig. 5b has nodes labelled with the same formula only at the top formulas, i.e., hypothesis or top-formulas. In this case, the four occurrences of  $A_1$  and the three occurrences of  $A_1 \supset A_2$ . Moreover, it is a simple directed and coloured graph, i.e., for each colour  $c$  and pair  $u$  and  $v$  of nodes, there is at most one edge of colour  $c$  from  $u$  to  $v$ . Suppose we do not count the top formulas. In that case, the obtained dag-like derivation is of size  $h.m^3$ , where  $h$  is the height of the original tree-like ND derivation and  $m$  is the number of formulas in the original derivation. Please, refer to [Haeusler et al. 2023] for more detail.

The next, and final, step in the compression task is to collapse the top formulas. We use **Moving Downward Edges** rules (*MDE*-rules) in this task. The application of the *MDE*-rules moves the ancestor edges down, but preserves the number of ancestor edges, so that we have at most one top formula by level at the end of the compression process, and the same number of edges counted above. We can conclude that the dag-like proof is polynomial on the number of formulas and the height of the tree-like derivation.

### 3. Formalisation in Lean

This section describes the first steps towards a formalization in Lean (Lean 3, v0.16.53) for the proof of the following:

» **Main Theorem:** Let  $HCom(u, v)$  be the application of a type-0 or type-1 / type-2 or type-3 compression rule to nodes  $u$  and  $v$  of a **DLDS**  $\mathcal{D}$ . Let  $u'$  be the resulting collapsed node and  $\mathcal{D}'$  the resulting **DLDS** after the collapse. Let  $v'$  be a node in  $\mathcal{D}'$ , at the same level and with the same  $M_\supset$  formula as  $u'$ . Then there must be some valid application  $HCom(u', v')$  of some type-1 / type-3 compression rule to nodes  $u'$  and  $v'$ .

As a consequence of the above, if **HC** is applied to a valid **DLDS**, then we can be certain that it eventually halts exiting a **DLDS** that has no level with two nodes labelled with the same formula. From the Main Theorem 3, we take that every matching involving a collapsed node resulting from the application of a type-0 or type-1 rule will be mapped by a type-1 rule. Similarly, every matching involving a collapsed node resulting from the application of a type-2 or type-3 rule will be mapped by a type-3 rule. So long as there is a rule among the compression rules that maps the matching nodes, **HC** will collapse them. Therefore, **HC** will not halt before collapsing every node. This section prioritizes explaining more complex/important proofs, lemmas, and definitions

over simpler ones. The complete, up-to-date, formalization in Lean can be viewed at [Horizontal-Compression](#).

### 3.1. Type Definitions


As our result is about the compression rules that define  $HCom(u, v)$ , we need to create a type for the entries  $u$  and  $v$ . In this context,  $u$  and  $v$  indicate the nodes to be collapsed by the application of  $HCom(u, v)$ . However, the information of which nodes  $u$  and  $v$  represent is insufficient for our proof. Not only the nodes to be collapsed, we need to know all neighboring arrows and vertexes around the nodes  $u$  and  $v$ . The [neighborhood](#) type is directly defined by the compression rules, and composed of 4 distinct parts: a central node which is used for the collapse (CENTER); a list of deduction edges arriving at that central node (IN); a list of deduction edges exiting from that central node (OUT); and a list of edges of ancestry pertinent to the compression rule (ANCESTRAL). These parts are given as parameter to the `neighborhood` type constructor. The types of these parameters are [node](#), [deduction](#), and [ancestral](#).

In the `node` type's definition, a node's level and label (an identifier unique to that node) are each represented by a natural number, which must be given as parameters to the type constructor. This representation is justifiable because the number of possible levels/labels in a DLDS is always a natural number, so any arbitrary ordering over the set of possible levels/labels creates a bijection between the set of levels/labels and the natural numbers. The level parameter of a node is used to associate nodes of the DLDS for collapse, and check if they are at the same level of the DLDS. Using a parameter to represent the label of a node makes it possible to differentiate any two nodes of the tree even when looked at in isolation. These parameters allow for a better, more precise categorization of the nodes, something which our proof requires. The last part of a node is its formula, which in this context is a  $M_{\supset}$  formula, defined as [formula](#). The `deduction` type is used to instantiate a neighborhood's deduction edges while the `ancestral` type is used to instantiate a neighborhood's edges of ancestry. A deduction edge is composed of: a starting node (START); an end node (END); an identifying colour (COLOUR); and a dependency set (DEPENDENT). An edge of ancestry is composed of: a starting node (START); an end node (END); an identifying colour path (PATH). The formalisation of both these types is taken as a direct translation from their definition as stated in the previous sections.

### 3.2. Proving the Main Theorem

Central to understanding the [Lean-assisted proof](#) are the methods:

- [elimination\\_neighborhood](#), [introduction\\_neighborhood](#), [hypothesis\\_neighborhood](#), which validates if a given neighborhood of the DLDS represents a node that is the conclusion of an application of  $\supset$ -Elim, the conclusion of an application of  $\supset$ -Intro, or a hypothesis;
- [neighborhood\\_type\\_01](#), which validates if a given neighborhood of the DLDS represents a node that is the resulting collapsed node of an application of either a type-0 or a type-1 compression rule;
- [check\\_nodes](#), which checks if two given neighborhoods are valid and if they should be collapsed; and


- [horizontal\\_collapse](#) , which takes two applicable neighborhoods and exits a collapsed neighborhood, like shown in the figures at Subsection 2.

By analysing these four categories of neighborhoods, and considering the method for horizontal collapse, the proof of **Main Theorem 3** proceeds by comparing each type-0 and type-1 rule, again disregarding symmetrical cases, with each case formalised separately. To prove the same for rules of type-2 and type-3 the process is almost identical, save for the definitions invoked.

We wrote the formalization to more closely resemble a computer program, motivated by the reasoning that this would make it easier for us to write it and later for the reader to comprehend it. Usage of Lean’s tactics mode, marked by keywords `begin` and `end`, means that, when processing the input bracketed by the keywords, the theorem prover can execute each tactic in a compound sequence to produce an expression of its required type. It also helps the user keep track of the multiple goals and subgoals involved in a proof by forcing the theorem prover to show that kind of information. On the topic of goals/subgoals, tactics `have` and `from` are used to introduce new subgoals and solve existing goals/subgoals, respectively. The tactic `from` provides an exact proof term as the solution, meaning that if  $G$  is a goal and  $t$  is a term of type  $T$ , then `from t` succeeds, iff,  $G$  and  $T$  can be unified. Finally, `assume` is used here to introduce and name all our hypotheses and variables.

## 4. Conclusion

In this paper we intended to prove that **HC** halts for every  $M_{\supset}$  tautology, exiting a valid **DLDS** with no two equal nodes on the same level.

A **DLDS** is a type of graph, similar to a ND tree-like derivation, with levels over which recursion can be applied. Still, there is no need to ever instantiate an entire **DLDS** to prove our result, allowing us to use the [neighborhood](#)  type instead. The `neighborhood` type has a more direct definition, based solely on the compression rules. Using Lean’s `ℕ`, `list`, and `set` types required some additional definitions. Their number is minimal, though, and most come directly from the in-built recursion over those types.

Compression of naive and huge proofs in natural deduction for the non-hamiltonianicity of some graphs has been done in [[Barros Júnior and Haeusler 2019](#)]. A compression ratio of almost 90% was obtained, with bigger and more redundant proofs having the best compression ratio. We intend to write a Lean-assisted proof of the soundness of each compression rule, or that the set of compression rules preserves the validity of any **DLDS**, as shown informally in **Theorem 11** in [[Haeusler et al. 2023](#)]. This will finish the complete Lean-assisted proof of the **Main Theorem 3**.

## References

- Barros Júnior, J. and Haeusler, E. (2019). **A comparative study on compression techniques for Propositional Proofs**. In *Book of Abstracts, 19th Braz. Meeting on Logic*, pages 85–86, Brazil.
- de Moura, L., Kong, S., and Avigad, J. (2023). **Theorem Proving in Lean**.
- Haeusler, E., Moura Brasil Filho, R., and Barros Júnior, J. (2023). **On the horizontal compression of dag-derivations in minimal purely implicational logic**.
- Prawitz, D. (1965). *Natural Deduction: Proof-Theoretical Study*. Dover Publications.