On the horizontal compression of dag-derivations in minimal purely implicational logic

Edward Hermann Haeusler
PUC-Rio, Rio de Janeiro, Brasil
Departamento de Informática
Pontifical University Catholic of Rio de Janeiro
Rio de Janeiro, RJ, Brasil
hermann@inf.puc-rio.br

José Flávio Cavalcante Barros Jr.
Quixadá - UFC
Quixadá, Brasil
flavio@gmail.com

Robinson C. de M.B. Filho
PUC-Rio, Rio de Janeiro, Brasil
Departamento de Informática
Pontifical University Catholic of Rio de Janeiro
Rio de Janeiro, RJ, Brasil
rfilho@inf.puc-rio.br

In this report, we define (plain) Dag-like derivations in the purely implicational fragment of minimal logic $M$. Introduce the horizontal collapsing set of rules and the algorithm $HC$. Explain why $HC$ can transform any polynomial height-bounded tree-like proof of a $M$ tautology into a smaller dag-like proof. Sketch a proof that $HC$ preserves the soundness of any tree-like ND in $M$ in its dag-like version after the horizontal collapsing application. We show some experimental results about applying the compression method to a class of (huge) propositional proofs and an example, with non-hamiltonian graphs, for qualitative analysis. The contributions include the comprehensive presentation of the set of horizontal compression (HC), the (sketch) of a proof that HC rules preserve soundness and the demonstration that the compressed dag-like proofs are polynomially upper-bounded when the submitted tree-like proof is height and foundation polynomially upper-bounded. Finally, in the appendix, we show an algorithm that verifies in polynomial time on the size of the dag-like proofs whether they are valid proofs of their conclusions.

1 Introduction

In a series of articles, [GH19] [GH20] [GH22c] L.Gordeev and the first author provided a proof that $NP = PSPACE$ based on the transformation of tree-like Natural Deduction derivations for purely intuitionistic minimal logic ($M$) into Direct Acyclic Graphs (Dags). This transformation acts by identifying equal occurrences of formulas at the same level of the tree. This identification, or collapse, changes the tree-like derivation to preserve information about the logical consequence after the collapse. It turns the tree into a Dag. Since the publication of [GH19], many readers of it demanded a computer-assisted proof of the main result due to its underlying combinatorial structure being hard to follow. The size of the entire collapsed (Dag-like) proof is upper-bounded by a polynomial on the conclusion of the proof whenever the set of formula occurrences in the original tree-like proof and the height of the tree-like proof also is. Using [Hud93], the article published in Studia Logica, [GH19],

*CAPES
†CAPES
‡CAPES
On the horizontal compression

closes the requirements on a polynomial bound on both the height and the number of formula occurrences in any ND tree-like proof of a tautology in \( M \). Demands of some readers about the need to read another proof-theoretical paper to understand a proof of a result on computational complexity moved the research to a further improvement. In [GH22c], we drop out the need of [Hud93]. We show that the horizontal compression can be applied directly to a relevant and specific class of height and formula occurrences poly-bounded Natural Deduction proofs in \( M \), namely the class of proofs of non-hamiltonicity for (non-Hamiltonian) graphs. Thus, [GH22c] shows a proof of \( \text{CoNP} = \text{NP} \) that does not need Hudelmaier linear bound. Finally, in [GH22b] we can find an overview of these proofs that proposes a non-deterministic approach that takes into account the Dag-like proofs without the need for a deterministic way of reading it.

In this report, we describe some experimental results on the application of an implementation of the compression method to a class of (huge) propositional proofs, as well as we show an illustrative example, with non-hamiltonian graphs, to provide a qualitative analysis of HC, see section 8. The contributions of this report are along with the respective sections. It conveys a comprehensive presentation of the set of horizontal compression (HC) in section 5. It shows the (sketch) of a proof that HC rules preserve soundness in section 6. We show, in section 7, a demonstration that the compressed dag-like proofs are polynomially upper-bounded when submitted to height and foundation poly-bounded tree-like ND proofs. In section 8 we discuss the experimental results we have on the implementation of the HC compression algorithm and how it performs when compared with the Huffman compression of strings benchmark. Finally, in the appendix A we show an algorithm that verifies in polynomial time on the size of the dag-like proofs whether they are valid proofs of their conclusions. We want to inform the reader that we are, hopefully, developing a proof assisted version of some sections of this report. The reader should have a basic knowledge of Natural Deduction and proof-theory to appreciate better what we convey here. It is interesting mentioning that in [Hae22] there is a proof-theoretical argument that provides a good explanation of reasons for the excellent compression rate of HC when compared to traditional benchmarks in terms of string compression.

We start this report by showing an example of the HC compression on a very simple and, we hope, illustrative example. This is the content of the following section 2. Our logical language is restricted to the implication. The logic is the purely implicational propositional logic. We use the symbol \( \supset \) for the implication to avoid confusion with the \( \rightarrow \) symbol, largely used in graphs pictures and representations.

## 2 Horizontally compressing Natural Deduction derivations into a Dag-like Derivations

In this article we use bitstrings for representing subsets of a linearly ordered finite set. According this mapping, the subset \( \{ A \supset B, B \supset C, \} \) 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 00111. The formal definition of this mapping, that is a bijection, can be found in section 4, definition 5.

The above derivation is a greedy derivation, i.e., all \( \supset \)-Intro rule applications of a formula \( \alpha \supset \beta \) discharge every possible occurrence of \( \alpha \) that is a hypothesis of the derivation of the its premiss, \( \alpha \). Below we have the representation of the above derivation as a labeled tree. The nodes of the tree are labeled by the formulas in the derivation and the edges are labeled by bitstrings that represent sets of formulas. In the tree depicted in figure 2 we consider the following linear ordering \( < \) on the set of formulas in the derivation in figure 1. We note that the we can chosse any linear order to represent
Figure 1: Deriving $A_1 \triangleright A_5$ from $A_1 \triangleright A_2$, $A_1 \triangleright (A_2 \triangleright A_3)$, $A_2 \triangleright (A_3 \triangleright A_4)$ and $A_3 \triangleright (A_4 \triangleright A_5)$.

Natural Deduction derivation in the form of trees. In the sequel, this linear order is used to represent sets as bitstrings in the context of directed acyclic graphs (DAGs).

With the sake of a better understanding, we remember that to any subset of formulas in the derivation there is a unique bitstring, see again definition 5. For example, the set \{\(A_1, A_1 \triangleright A_2, A_1 \triangleright (A_2 \triangleright A_3)\}\) is represented by the bitstring 100001000000.

Figure 2: Labelled graph representation of derivation 1.

The goal of the HC-compression algorithm is to obtain a smaller representation of proofs in minimal implicational logic by applying a set of rules to a given valid derivation in the implicational minimal logic. The rules are applied in a deterministic way, bottom-up and left to right, and collapse redundant parts of the derivation. HC-compression firstly collapses two lower occurrences of the formula \(A_3\) and follows the direction from bottom-up and left to right in the tree, creating a directed acyclic graph. The rule that is used appears in figure 3, and its official name is \(R0EE\). The rule in question matches the tree in level 3 according the markings in figure 4. The nodes labelled with \(A_3\) match with \(u\) and \(v\), appearing in the rule, respectively, and their respective children \(p_1\) and \(p_4\), and, \(p_2, p_3\) match with the premises in the rule accordingly.

We use rule \(R0EE\) to show how to read the pictorial representation of each horizontal compression rule. Both the left and the right-hand sides are subgraphs of, respectively, derivations, \(DLDS\), \(D\) and
The meaning of this rule is to replace the subgraph represented by the left-hand side by the right-hand side graph in \( \mathcal{D} \), resulting in \( \mathcal{D}' \). The definitive definition is in a following section. Moreover, we have that those belonging to \( E \) and also it is shown in section 23. In the sequel, we will find the label under \( \mathcal{P} \). The new derivation is represented by a formula) with the conclusion of an application of \(-\text{Elim}\) too. Other rules represent the collapse of a hypothesis (assumption or top-formula) with the conclusion of a \( \supset \)-Elim rule with an application of \( \supset \)-Elim too. Other rules represent the collapse of a hypothesis (assumption or top-formula) with the conclusion of a \( \supset \)-Elim, the collapse of two hypotheses. We will see that we have four classes of rules. The naming of the rules follow a schema. They are named as \( R_{\text{im}} \), where \( i = 0 \ldots 3 \) and \( m_l, m_r \in \{ I, E, H, X \} \), such that, \( i \) is the type of the rule, \( m_l \) and \( m_r \) means the kind of the labels in the left and right nodes that will collapse. There is a small variation in this naming rule. There are names such as \( R_{\text{c}}2m_l m_r \) and \( R_{\text{c}}2m_l m_r \). The indexes \( v \) and \( e \) indicates that the respective rule collapses only vertexes (\( v \)) or edges (\( e \)) too. See figures 7 and 51 for a better understanding. For example, the rule \( \text{R0EE} \) is in figure 5. In this introductory example it is not important to understand

\footnote{Dag-Like Decorated Structure}
what a **DLDS** is in detail, but the fact that they represent derivations, possibly in the form of a **DAG**, Directed Acyclic Graph. The \( X \) represents that the formula is conclusion of more than one rule at the same time.

\[
\begin{align*}
p_4 & \quad p_1 \\
\bar{a}_1 & \quad \downarrow \bar{c}_1 \\
\bar{c} = \bar{a}_1 \lor \bar{c}_1 & \quad \downarrow \bar{b}_2 = \bar{b}_1 \lor \bar{a}_1
\end{align*}
\]

\[ \quad \text{HCom}(u,v) \xrightarrow{R0EE} \]

**Figure 3**: (a) \( u \) and \( v \) collapse (b) After collapse \( \text{HCom}(u,v) \)

\[
\begin{align*}
\text{u:} \quad A_3 & \quad \text{b1:} \quad 010001000010 \\
\text{d1:} \quad 010001000010 & \quad \text{A3:} \quad A_2 \lor \text{A5} \\
\text{A4:} \quad A_3 \lor & \quad \text{A5:} \quad A_4 \lor \text{A5}
\end{align*}
\]

**Figure 4**: Matching of the HC-compression rule **R0EE** with the derivation-tree in figure 2

How does someone read this dag-like derivation? The blue edges, the so called ancestor egdes drive the reading. The reader should note that each of these edges are labelled with bracketed lists of numbers. For example, the list \([0,2]\), in red, labels the edges that goes from the node labelled with \( A_4 \supset A_5 \) to the labelled nodes \( A_2 \) and \( A_2 \supset A_3 \), respectively. Consider the deductive edges labeled by the number 0, i.e., all the edges in a tree-like original derivation are labelled with 0. The dag-like derivation, on the other hand, have two edges labelled with numbers different of 0. Well, note that the list \([0,2]\) 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 to follow from \( A_2 \supset A_3 \) to \( A_4 \supset A_5 \). These two paths are only 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. In this way
On the horizontal compression

the ancestor edges drive the correct reading of the dag-like derivation when regarding to the original tree-like derivation, before the application of the collapsing rules.

Continuing the application of the HC-compression algorithm to the resulted dag-like derivation, the following collapse from bottom up is related to the occurrence of three \( A_2 \) in level 4 of the dag-like derivation. The dag-like derivations that the HC-algorithm deals with is always leveled dags. Here we have to note that which rule will be applied depends on what are the first two occurrences from left to right in the dag. They are the leftmost one and the other leftmost node with label \( A_2 \) and that it is minor premise of a \( \supset \)-Elim with \( A_2 \supset (A_3 \supset A_4) \) as major premise. The rule \( R_v2EE \) that appears in figure [7] is used in this case and its matching to the current dag-like deduction, in figure [8] shows up in figure [8]. In figure [9] we show a graphical rearrangement of the same dag-like derivation and match information with rule \( R_v2EE \) that appears in figure [8]. In figure [10] we show the result of the application of rule \( R_v2EE \) according the matching in figure [9]. In figure [11] we show the match of the dag-like derivation in figure [11] with rule \( R_e2EE \) that appears in figure [51]. In figure [12] we show the result of the application of rule \( R_e2EE \) to the resulted dag-like derivation in figure [14] to collapse the two occurrences of \( A_2 \supset A_3 \) in level 5. The associated match is depicted in figure [16]. The resulted application appears in figure [17]. The defocused version appears in figure [18]. Finally, by repeated applications of rules \( R_v2HH \), \( R_e3XH \) and \( R_e2HH \), see figures [20], [45] and [19] we obtain the fully compressed DLDS. We detail the next steps in the sequel.

\[ A_1 \supset A_2 \]
\[ A_2 \supset A_3 \]
\[ A_3 \supset (A_4 \supset A_5) \]
\[ A_1 \supset A_2 \]
\[ A_2 \supset (A_3 \supset A_4) \]
\[ A_3 \supset A_4 \]
\[ A_4 \supset A_5 \]

Figure 5: Result of the HC-compression rule \( R0EE \) application according the matching shown in figure [4]
Figure 6: Defocused result of the HC-compression rule $R_0EE$ application appearing focused in figure 5.

Figure 7: (a) $u$ and $v$ collapse. (b) After collapse $HCom(u, v)$.
Figure 8: Matching of the HC-compression rule $R_{2EE}$ with the dag-like derivation in figure 6

Figure 9: Rearrangement of figure 8 for graphical reasons
Figure 10: Result of the HC-compression rule $R_{v2}EE$ application according the matching shown in figure 9.

Figure 11: Defocused dag-like derivation after application of rule $R_{v2}EE$ depicted in figure 10.
On the horizontal compression

Figure 12: (a) $u$ and $v$ collapse by rule $R_c3XE$

(b) After collapse $HCom(u,v)$
Figure 13: Match of rule $R_{e3XE}$ in figure 12 with dag-like derivation in figure 11

Figure 14: Result of the application of rule $R_{e3XE}$ to the match in figure 13
On the horizontal compression

Figure 15: (a) $u$ and $v$ collapse (b) After collapse $HCom(u,v)$

Figure 16: Match of rule $R_2EE$ with dag-like derivation in figure 14
Figure 17: Dag-like derivation resulted from rule $R_{e2EE}$ application after match in figure 16

Figure 18: Defocused dag-like derivation after application of rule $R_{e2EE}$ as in figure 17
The DLDS in figure 18 shows that in all levels, except the last one, there are no repeated formulas labelling its nodes. There repeated formulas/labels only in the last levels. This is the form obtained by the application of MUE-rules, i.e., Moving Up Edges rules of the HC algorithm. In this form of DLDS two nodes labeled with the same formula have no incoming deduction, also called top-nodes.

The MDE-rules, for Moving Down Edges, are used to collapse at least one top-nodes that have the same formula labeling them. The first two rules are $R_v 2HH$ and $R_e 3XH$ in figures 20 and 45. The result of the application of these two rules is in figure 22. The graph is not coloured simple anymore, since there are two ancestor edges from $A_4$ to $A_2$. The next step is to collapse the two occurrences of the formula $A_1 \supset (A_2 \supset A_3)$ by an application of $R_e 2HH$, yielding the DLDS in figure 23. Finally, the two leftmost nodes are collapsed by the rule $R_e 2HH$. The last steps are one application of $R_e 2HH$, figure 19, and three subsequent applications of $R_e 3XH$ to collapse the five occurrences of $A_1$. In figure 24 the two leftmost $A_1$ are collapsed by the rule $R_e 2HH$. Note that the rule adds an additional ancestor edge from $A_4$ to $A_2$ labeled with the path $[1;1]$. This labeled ancestor edge is already there. This rule collapses edges, the new one and the already existing one, by the set-theoretical definition of DLDS. However, in the figures of this example we will show the repeated ancestor edges to provide a better understanding on the compression mechanism to the reader.

In figures 28 and 29 we show the summary of MUE and MDE rule applications, respectively, along this example on the derivation in figure 1.
Figure 20: (a) $u$ and $v$ collapse  
(b) After collapse $HCom(u, v)$

Figure 21: (a) $u$ and $v$ collapse  
(b) After collapse $HCom(u, v)$
On the horizontal compression

Figure 22: Defocused dag-like derivation after application of rules $R_e2HH$ and $R_e2XH$ to the dag-like in figure 18, collapsing the three occurrences of $A_1 \supset A_2$,
Figure 23: Defocused dag-like derivation after application of the rule $R_{v2HH}$ to the dag-like in figure 22, collapsing the two occurrences of $A_1 \supset (A_2 \supset A_3)$
Figure 24: Defocused dag-like derivation after application of $R_e2HH$ to collapse the two leftmost $A_1$’s in the dag-like in figure 23.
Figure 25: Defocused dag-like derivation after application of $R_2HH$ to collapse the two leftmost $A_1$’s and a $R_2XH$ to collapse the third occurrence of $A_1$ in the dag-like in figure 24.
Figure 26: Defocused dag-like derivation after application of $R_{e3XH}$ to collapse the third leftmost $A_1$’s and another $R_{e2XH}$ to collapse the fourth leftmost $A_1$ in the dag-like in figure 25.
Figure 27: Defocused dag-like derivation after application of $R_e2HH$ and threetimes $R_e2XH$ to collapse all occurrences of $A_1$ in the dag-like in figure 26.
On the horizontal compression

Figure 28: Summary of MUE applications to the initial derivation in [1]
Figure 29: Summary of MDE applications to the initial derivation in [1]
There are no repeated occurrences of the same formula in any level of the dag-like derivation in figure 27 and the last DLDS in figure 28. The HC-compression has compressed the tree-like derivation in figure 2 that represents the Natural Deduction derivation in figure 1. Figure 27 is not a tree-like derivation with the standard top-down reading from the hypothesis downwards to the conclusion used to validate the derivation. We have to define a formal device to take care of the dependency set related to each deductive edge for dag-like derivations. In the sequel, we show how would be this formal device using the dag-like derivation in figure 27 or better, figure 31 that removes the emphatic ancestor edges repetitions and with names to each node. We would like to show the that the collapsed dag-like derivation corresponds also to a kind of Natural Deduction that is made by linking derivations so that. For example, the tree-like derivation in figure 1 can be viewed as in figure 30 in this linked form.

Figure 30: Labelled linked-graph representation of derivation 1

In figure 31 we show a version of figure 27 with names for each node to facilitate the reading of the following explanation. We also removed the emphasis on the repetition of E-edges caused by the MDE rules, for they have the same labels. The set-theoretical definition of DLDS takes at most one labeled edge by colour by pair of edges. Nodes p01, p02 and p03 have top-formulas of the dag-like derivation as labels. Reflexivity is the logical principle that states that any proposition is the logical consequence of itself. In minimal logic, reflexivity holds. Thus, the deductive edges that link any node labelled with a proposition α to another node in the dag should have the bitstring ¯α as its label. The edges ⟨p01, p12⟩, ⟨p02, p13⟩ and ⟨p03, p13⟩ should be ¯b(1 ⊃ A2), ¯b(A1), and ¯b(1 ⊃ (A2 ⊃ A3)), respectively. These sets of dependence label the original edges in the dag-like derivation in the figure that came from the original tree-like derivation. This agreement shows that the dag-like’s soundness agrees with the semantics of Minimal Implicacional Logic, or MImP. Consider the nodes p02, p03 and p13. They form an ⊃-elim rule, with p02 and p03 premisses and p13 conclusion. In the figure, we can see that p02 is labelled with the minor premiss, the major
premiss labels \( p_{03} \), and finally, the conclusion labels the node \( p_{13} \). Thus, according to the logical reading of this inference rule, the conclusion depends on whatever both premises depend on. In this case, this means that the edge \( \langle p_{13}, p_{22} \rangle \) that goes out of the node \( p_{13} \) should be a label with the union of the dependency sets that labels the edges that go into \( p_{13} \). This is the set represented by the bistring \( 1000000000100 = b(\{ A_1 \supset (A_2 \supset A_3), A_1 \}) \). As we can note, \( \bar{\lambda} \) labels the deductive edge \( \langle p_{13}, p_{22} \rangle \). The symbol \( \bar{\lambda} \) represents a bistring that is dynamically computed and assigned to the edge. Theoretically, the label \( \bar{\lambda} \) agrees with any bistring. It is a kind of stakeholder assigned dynamically a computed dependency set that takes part in the validity verification process of the dag-like derivation. Moreover, the above analysis can correspond to the verification that the node \( p_{13} \) represents a correct application of \( \supset \text{-Elim} \), where the nodes \( p_{02} \) and \( p_{03} \) are labelled with the minor and major premises and \( p_{13} \) the conclusion. The dependency sets are assigned correctly hence. However, even considering that the \( \lambda \) agrees with any dependency set, it remains to convey the dependency set information relative to the edge \( \langle p_{13}, p_{22} \rangle \) to have conditions to propagate downwards the verification to the next inference rule. That is, we have to keep the information that the edge \( \langle p_{13}, p_{22} \rangle \) carries the dependency set \( 1000000000100,0 \), in bistring form, that is nothing more than \( \bar{b}(\{ A_1 \supset (A_2 \supset A_3), A_1 \}) \).

This can be performed by coding a verification algorithm, that we will show in the sequel. However, we have chosen to formalize the dependency sets associated to each edge in the graph, by the recursive function \( \text{Flow} \). The formal definition of this function is provided by definition \( 22 \) and it is used to state the validity of dag-like derivations. However, we advise the reader to consult the formal definition after finishing the reading of the explanations below. We believe this facilitates the understanding of the example and why, and how, the compression rules preserve validity of derivations, which is proved in a forthcoming section. For each dag-like derivation \( \pi \), for each node \( w \) in \( \pi \), we define \( Pre_\pi(w) = \{ v : \text{there is a deductive path from } v \text{ to } w \text{ in } \pi \} \). A deductive path from \( v \) to \( w \) is any sequence of deductive edges, of any color, \( e_0, \ldots, e_n \equiv w \), such that, for any \( i = 0, n - 1, e_i = \langle v_i, v_{i + 1} \rangle \), \( v = v_0 \) and \( v_n = w \). The definition of \( \text{Flow} \) takes a graph \( \pi \), a node \( w \in \pi \) and for each \( v \in Pre(w) \), provides the dependency set of the edges that go out of \( w \), for each possible path arriving in \( w \) from \( v \). Explaining better, if \( \pi \) is the dag-like derivation in figure \( 31 \) then, below we have all the cases that \( \text{Flow}(\pi, w)(v) \) are defined with \( v \in Pre_\pi(w) \).

\[
\begin{align*}
\text{Flow}(\pi, p_{12}|p_{01}) &= \{000001000000, 0\} \\
\text{Flow}(\pi, p_{12}|p_{02}) &= \{100000000000, 0\} \\
\text{Flow}(\pi, p_{13}|p_{01}) &= \{000000000010, 0\} \\
\text{Flow}(\pi, p_{13}|p_{02}) &= \{000000010000, 0\} \\
\text{Flow}(\pi, p_{21}|p_{01}) &= \{000000000000, 0\} \\
\text{Flow}(\pi, p_{21}|p_{02}) &= \{000000000000, 0\} \\
\text{Flow}(\pi, p_{21}|p_{11}) &= \{000000000010, 0\} \\
\text{Flow}(\pi, p_{22}|p_{01}) &= \{000000000000, 0\} \\
\text{Flow}(\pi, p_{22}|p_{02}) &= \{000000000000, 0\} \\
\text{Flow}(\pi, p_{22}|p_{03}) &= \{000000000010, 0\} \\
\text{Flow}(\pi, p_{22}|p_{12}) &= \{000000000000, 0\} \\
\text{Flow}(\pi, p_{22}|p_{13}) &= \{000000000000, 0\} \\
\text{Flow}(\pi, p_{22}|p_{11}) &= \{000000000010, 0\} \\
\text{Flow}(\pi, p_{23}|p_{01}) &= \{000000000000, 0\} \\
\text{Flow}(\pi, p_{23}|p_{02}) &= \{000000000000, 0\} \\
\text{Flow}(\pi, p_{23}|p_{03}) &= \{000000000010, 0\} \\
\text{Flow}(\pi, p_{23}|p_{11}) &= \{000000000010, 0\} \\
\text{Flow}(\pi, p_{23}|p_{12}) &= \{000000000000, 0\} \\
\text{Flow}(\pi, p_{23}|p_{13}) &= \{000000000000, 0\} \\
\text{Flow}(\pi, p_{23}|p_{11}) &= \{000000000010, 0\} \\
\text{Flow}(\pi, p_{23}|p_{12}) &= \{000000000000, 0\} \\
\text{Flow}(\pi, p_{23}|p_{13}) &= \{000000000000, 0\} \\
\text{Flow}(\pi, p_{23}|p_{11}) &= \{000000000010, 0\} \\
\text{Flow}(\pi, p_{23}|p_{12}) &= \{000000000000, 0\} \\
\text{Flow}(\pi, p_{23}|p_{13}) &= \{000000000000, 0\} \\
\text{Flow}(\pi, p_{23}|p_{11}) &= \{000000000010, 0\} \\
\text{Flow}(\pi, p_{23}|p_{12}) &= \{000000000000, 0\} \\
\text{Flow}(\pi, p_{23}|p_{13}) &= \{000000000000, 0\}
\end{align*}
\]
By analysing the dag-like derivation top-down, and, considering that the leaf-nodes determine the initial dependency sets, we can calculated the dependecy set induced by the dag-like derivation. This must be done anyway to verify that the dag-like derivation is correct.

To verify that the dag-like derivation is sound we perform an algorithm that is linear on the size of the dag-like derivation. The algorithm uses a set of registers, each register stores a pair formed by a dependency set and a path downwards. There is one register for each leaf-node and path that labels the ancestor edge that arrives in this leaf-node. The algorithm goes top-down following the paths in each register and updating the respective dependency set and verifying the correct application of the corresponding rules. If all rules are correct the algorithm ends with the dependency set that is associated to the derivation. This algorithmm its correctness and complexity analysis is shown in appendix A.

3 Dag-like proofs basic definitions

In this section, we define dag-like proofs and derivations in $M_{\supset}$, the minimal purely implicational logic. Dag-like proofs and derivations are data structures resulting from the compression of tree-like Natural Deduction proofs and derivations by the method that we initially defined in [GH19] based on the collapsing of nodes of the underlying tree of given derivations or proofs.
This section contains all the basic definitions and concepts used by the horizontal compression method and most of the results, namely theorems and other related results. They are: 1- The method preserves soundness of the original derivation, and consequently proofs also; 2- The method is terminating, and. 3- The size of the compressed proof belongs to $O(H \times F^3)$, where $H$ is the height of the original proof tree and $F$ is the size of the foundation of the proof tree, i.e., the set of all formulas that occurs in the derivation or proof. The minimal implicational logic has only implicational propositional formulas. The Natural Deduction for $\mathbb{M}_\square$ has only the $\top$-Elimination and $\top$-Introduction rules of N.D. Prawitz for minimal logic, see [Pra65].

**Definition 1.** Let $\Pi$ be a Natural Deduction derivation in $\mathbb{M}_\square$. Each formula $\alpha$ occurring in $\Pi$ is called a formula occurrence. Each formula occurrence is unique in $\Pi$.

It is important to emphasize that two formula occurrences in a derivation $\Pi$ may be an instance of the same formula. However, in a derivation, they are unique. We can formalize the feature we mentioned by enumerating the set of all formulas in the derivation so that each formula occurrence in $\Pi$ is a pair $\langle i, \beta \rangle$, where $i$ corresponds to the enumeration of $\beta$. Thus, $\langle i, \beta \rangle$ is unique in the derivation. We omit this pair notation whenever we explicitly state that $\beta$ is a formula occurrence.

**Definition 2.** A rooted Dag $\langle V, D, r \rangle$ is a directed acyclic connected graph $\langle V, E \rangle$ together with a distinguished node $r \in V$ that has out-degree 0. $r$ is the only node in $V$, such that out($r$) = 0.

In what follows, if $\langle V, D, r \rangle$ is a rooted dag, for each $v \in V$, In($v$) is the number of incoming edges to $v$ and out($v$) is the number of edges outgoing from $v$. In the case of colored edges, In$_c(v)$ and Out$_c(v)$ is the number of incoming and outgoing edge to and from the vertex $v$ with color $c$. A colored graph is a structure $\langle V, E_1, \ldots, E_n, r \rangle$, such that, for each $i = 1, n$, $E_i \subseteq V \times V$ is the sets of edges colored with color $i$. Moreover, for each $i \neq j$, we have $E_i \cap E_j = \emptyset$. We say that a colored graph has no cycles, iff, each subgraph of color $i$, for each $i = 1, n$, has no cycles. Finally, a path from $v$ to $w$ in a colored graph is a non-empty sequences of edges of the same color, linking $v$ to $w$. When such a path exists for a color $c$ we say that the predicate $\text{path}_c(v, w)$ holds on the respective colored graph.

**Definition 3.** A tree-like derivation of a formula $\alpha$ in $\mathbb{M}_\square$ is a rooted Dag $\langle V, E_D, E_d, r, l \rangle$, bi-colored, with colors $D$ and $d$, where $\langle V, E_D, r \rangle$ is a rooted Dag with no cycles, and $l : V \rightarrow L$, such that:

1. $\forall v \in V (v \neq r \Rightarrow (\text{in}_D(v) = 1 \lor \text{in}_D(v) = 2 \lor \text{in}_D(v) = 0));$
2. $\forall v \in V (\text{in}_D(v) = 1 \Rightarrow \exists w (E_D(w, v) \land \forall v_1 \in V (E_d(v_1, v) \Rightarrow (l(v) = "l(w) \cup l(v)"))));$
3. $\forall v \in V \forall v_1 \in V (E_d(v_1, v) \Rightarrow \text{path}_D(v, v_1) \land \text{in}_D(v_1) = 0);$
4. $\forall v \in V (\text{in}(v) = 2 \Rightarrow \exists w_1 \exists w_2 (E_D(w_1, v) \land E_D(w_2, v) \land (l(w_1) = "l(w) \cup l(v)")));$
5. $L \subset \text{Formulas}(\mathbb{M}_\square)$ is a superset of the set of subformulas of $\alpha$.

We call $L$ the foundation of the derivation. Moreover, $l(r)$ is said to be the conclusion of the derivation and the set $\{l(v) / (\text{in}(v) = 0) \land \neg \exists w (E_d(v, w))\}$ is the set of assumptions of the derivation.

The computational verification that a structure of the form $\langle V, E_D, E_d, r, l \rangle$ is a tree-like derivation can be performed in time $O((|E_D| + |E_d|))$. Of course, with data structures improved with pre-compiled features, such as its list of leaves, we have that the computational complexity can reach the linear time on the number of all edges. However, this is not essential for us because we want to show that the upper bound is polynomial, which is already the case.

**Definition 4.** We say that a tree-like derivation $T = \langle V, E_D, E_d, r, l \rangle$, is a derivation of $\alpha$, a $\mathbb{M}_\square$ formula, from a set of assumptions $\Gamma$, a set of $\mathbb{M}_\square$ formulas, if and only if, $l(r) = \alpha$ and the set of assumption of $T$, namely $\{l(v) / (\text{in}(v) = 0) \land \neg \exists w (E_d(v, w))\}$, is $\Gamma$. 
In the sequel, we define a mapping from Natural Deduction derivations of $\alpha$ from $\Gamma$ into tree-like derivations of $\alpha$ from $\Gamma$. It is straightforward to show that all conditions stated in definition 3 holds. Finally, the proposition 2 concerns the mapping from tree-like derivations to $M_\Delta$ N.D. derivations, i.e., in the other directions.

**Proposition 1.** Let $\Pi$ be a Natural Deduction derivation of $\alpha$ from $\Gamma$ in $M_\Delta$. There is a tree-like derivation $T = \langle V, E_D, E_d, r, l \rangle$, such that:

1. $\langle V, E_D, \alpha \rangle$ is the underlying tree of $\Pi$;
2. $r \in V$ is the root of $\langle V, E_D, \alpha \rangle$;
3. $l$ labels each node $v \in V$ to is formula occurrence in $\Pi$, in particular $l(r) = \alpha$;
4. The edges $E_D$ link the nodes regarded to formula occurrences that are premisses of a natural deduction rule application in $\Pi$ to its respective node that is its conclusion;
5. The edges $E_d$ represent the discharging function associated to the application of an $\exists$-Introduction, such that $E_d(v, w)$ if and only if, $l(w)$ is an implication that is proved via an application of a $\exists$-rule that discharges the occurrence of $l(v)$ located at $v$.

**Proposition 2.** There is a bijective mapping $F$ between finite tree-like derivations and derivations in $M_\Delta$, such that:

- If $T = \langle V, E_D, E_d, r, l \rangle$ is a tree-like derivation of $\alpha$ from $\Gamma$, then $F(T)$ is a natural deduction derivation $\Pi$ of $\alpha$ from $\Gamma$.
- $l(r)$ is the conclusion of $\Pi$;
- $\{l(v) / (in(v) = 0) \land \neg \exists w(E_d(v, w))\}$ is the set of open assumptions of $\Pi$.
- For each node $v \in V$, there is a unique formula occurrence $\beta$ in $\Pi$, such that $l(v) = \beta$, and:
  1. If $\text{in}(v) = 0$ and $\neg \exists w \in V(E_d(v, w))$ then $\beta \in \Gamma$;
  2. If $\text{in}(v) = 0$ and $\exists w \in V(E_d(v, w))$ then the $\beta$ occurrence is discharged by a $\exists$-introduction rule that has $l(w)$ as premiss;
  3. If $\text{in}(v) = 1$ then there is $w \in V$, such that $E_D(w, v)$. So, $\beta$ is conclusion of an $\exists$-introduction in $\Pi$ that has an occurrence $l(w)$ as premiss;
  4. If $\text{in}(v) = 2$ then there are $w_1, w_2 \in V$, such that $E_D(w_1, v)$ and $E_D(w_2, v)$ holds and $l(w_2) = \left(\right. l(w_1) \supset l(v)$. So, $\beta$ is the conclusion of an $\exists$-elimination rule in $\Pi$ that has an occurrence of $l(w_1)$ as minor-premiss and an occurrence of $l(w_2)$ and major-premiss.
- For each formula occurrence $\beta$ in $\Pi$ there is a unique node $v \in V$, such that $l(v) = \beta$, and:
  1. If $\beta$ is the conclusion of $\Pi$ then $l(v) = \beta$ and $\text{out}(v) = 0$;
  2. If $\beta$ is a discharged assumption then $\text{in}(v) = 0$ and $\exists w \in V(E_d(v, w))$ and $l(w) = \left(\right. \beta \supset \beta' \right)$ for some $\beta'$;
  3. If $\beta$ is an open assumption then $\text{in}(v) = 0$ and $\neg \exists w \in V(E_d(v, w))$;
  4. If $\beta$ is $\beta_1 \supset \beta_2$ that is the conclusion of an $\exists$-introduction rule in $\Pi$ then there is $w \in V$, such that, $l(w) = \beta_2$ and $E_D(w, v)$;
  5. If $\beta$ is the conclusion of an $\exists$-elimination rule in $\Pi$ then there is
4 Dropping out the discharging ($E_d$) edges

We will 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 it depends on.

Let $L$ be the foundation of a derivation $\Pi$ in $M_{\supset}$. Consider a linear ordering $\mathcal{O}(L) = \{\beta_0, \beta_1, \ldots, \beta_k\}$ on $L$.

**Definition 5.** Let $L$ be a set of formulas in $M_{\supset}$ and $\mathcal{O}(\alpha)$ be a linear order $\mathcal{O}(L) = \{\beta_0, \beta_1, \ldots, \beta_k\}$. A bitstring on $\mathcal{O}(\alpha)$ is any string $b_0b_1\ldots b_k$, such that $b_i \in \{0,1\}$, for each $i = 1\ldots k$. There is a bijective correspondence between bitstrings on $\mathcal{O}(\alpha)$ and subsets of $L$, given by $\text{Set}(b_0b_1\ldots b_k) = \{\beta_i/b_i = 1\}$.

The bitstring on $\mathcal{O}(\alpha)$ will drop out the discharging edges and make explicit the information on formula dependencies in a derivation. The set of bitstrings on $\mathcal{O}(\alpha)$ is denoted by $\text{Bits}(L, \mathcal{O}(\alpha))$.

The following algorithm also produces from a proof $\Pi$ of $\alpha$ a $M_{\supset}$ greedy proof of $\alpha$. Consider that $n$ is the level of the highest branch in $\Pi$ and let $P$ be $\Pi$ in the code:

\[
j = n \\
\text{While } 0 < j \text{ do:} \\
\quad \text{For each branch } B \text{ of level } j \text{ in } P: \\
\quad \quad \text{replace each intro-app downwards by a greedy intro-app} \\
\quad \quad \text{possibly discharging more formula occurrences in } B \\
\quad j = j - 1 \\
\text{end}
\]

**Lemma 3** (Greedy $\supset$-Intro are complete). Let $\Pi$ be a proof of $\alpha$. Let $\Pi' = G(\Pi)$ be the result of the above function applied to $\Pi$. We conclude that $\Pi'$ is also a valid proof of $\alpha$ in $M_{\supset}$.

Proof. By induction on the size of the argument $\Pi$ we show that every $\supset$-intro rule in $G(\Pi)$ is a greedy application. So we are done.
We will remove the discharging function used in ND derivations to mark the formula occurrences discharged by a \( \supset \)-intro application. We will use labelling of formula occurrences in the ND derivations producing a decorated ND derivation that contains all the information regarding dependencies of any formula in the derivation. Below we define decorated greedy Natural Deduction derivations. We use the notation \( \vec{b}_\alpha \) to denote the bitstring \( b_0b_1\ldots b_k \), such that \( \text{Set}(b_0b_1\ldots b_k) = \{\alpha\} \), i.e., the only \( b_i = 1 \) is when \( i \) is the order of \( \alpha \).

**Definition 7** (Decorated Greedy N.D. derivation). Let \( \Gamma \) be a set of formulas and let \( \mathcal{O}_F \) be any linear ordering on these formulas. Let \( \mathcal{B}(\mathcal{O}_F) \) be the set of bitstrings on the ordering \( \mathcal{O}_F \). Let \( \Pi \) be a \( \supset \) derivation having an element of \( \mathcal{B}(\mathcal{O}_F) \) attached as a label to each formula occurrence in \( \Pi \). We say that \( \Pi \) is a Decorated Greedy N.D. derivation, iff, the following conditions hold to every formula occurrence \( \beta \), labelled with \( \vec{b} \), i.e., \( \vec{b}\beta \) in symbols, in \( \Pi \):

1. If \( \vec{b}\beta_1 \) is a top-formula (leaf) in \( \Pi \) then \( \vec{b} = \vec{b}_{\beta_1} \);
2. If \( \vec{b}\beta_1 \beta_2 \) is the conclusion of a \( \supset \)-Elim rule having \( \vec{b}\beta_1 \) and \( (\vec{b}_1 \supset \beta)\vec{b}_2 \) as premisses then \( \vec{b} = \vec{b}_1 \oplus \vec{b}_2 \);
3. If \( \vec{b}\beta_1 \beta_2 \) is the conclusion of a Greedy \( \supset \)-Intro rule application that has \( \vec{b}\beta_1 \vec{b}_2 \) as premiss then \( \vec{b} = \vec{b}' - \vec{b}_{\beta_1} \).

**Definition 8.** Given a derivation \( \Pi \) in \( \supset \), let \( \Gamma \) be the set of formulas occurring in \( \Pi \) and \( \mathcal{O}_F \) be any linear ordering on these formulas. Let \( L \) be the a mapping from the set of formula occurrences in \( \Pi \) to \( \mathcal{B}(\mathcal{O}_F) \). This mapping is called a labeling on its formulas.

**Definition 9** (Adequate Labeling). Let \( \Pi \) be a derivation in \( \supset \) and \( L \) be a labeling on its formula occurrences. We say that \( L \) is adequate to \( \Pi \), if and only if, for every formula occurrence \( \alpha \) of \( \Pi \), \( \text{Set}(L(\alpha)) \) is the set of formulas from which this occurrence of \( \alpha \) depends on.

**Lemma 4.** For every greedy \( \supset \) derivation \( \Pi \), there is a labeling \( L \), such that, \( L \) is adequate to \( \Pi \).

**Proof** The fact that \( \Pi \) has only greedy \( \supset \)-intro applications implies that for every conclusion \( \alpha \supset \beta \) of an \( \supset \)-intro application in \( \Pi \) has no \( \alpha \) belonging to its dependencies. Thus, by induction on the size of the derivation \( \Pi \) we prove that there is an adequate labeling for \( \Pi \).

The above lemma and lemma [3] supports the following proposition.

**Proposition 5.** For every set of formulas \( \Gamma \) and formula \( \alpha \) in \( \supset \), such that \( \Gamma \vdash \supset \alpha \), there is a decorated greedy derivation of \( \alpha \) from \( \Gamma \). In fact the label of \( \alpha \) is \( \vec{b}_\Gamma \).

**Definition 10.** A decorated tree-like (greedy) derivation with adequate labeling is a structure \( \mathcal{T} = \langle V, E_D, E_d, r, l, L \rangle \), where \( F = \bigcup_{v \in V} \{l(v)\} \) is the foundation of \( \mathcal{T} \), \( \mathcal{O}_F \) an linear ordering on \( F \) and \( L : E_D \rightarrow \mathcal{B}(\mathcal{O}_F) \) is an adequate labeling.

And using the above proposition we obtain:

**Proposition 6.** For every set of formulas \( \Gamma \) and formula \( \alpha \) in \( \supset \), such that \( \Gamma \vdash \supset \alpha \) there is a tree-like (greedy) derivation with adequate labeling \( \Pi \) of \( \alpha \) from \( \Gamma \) and a labeling \( L \) of the deduction edges, i.e. the elements of \( E_D \), by elements of \( \mathcal{B}(\mathcal{O}_S) \), for some ordering \( \mathcal{O}_S \) of the set \( S \) of formulas occurring in \( \Pi \). \( L \) is such that \( \text{Set}(L(\langle v_1, v_2 \rangle)) \) is the set of formulas from which the formula occurrence \( l(v_1) \) depends on inside \( \Pi \).
Proof We apply proposition 5 and lemma 4 to obtain a decorated greedy derivation $\Pi$ with an adequate labeling $L'$. Then, an application of Proposition 1 obtains a greedy tree-like derivation $T = \langle V, E_D, E_d, r, l, I \rangle$ of $\alpha$ from $\Gamma$. Finally, we define $L(\langle v, w \rangle) = L'(v)$, for every $\langle v, w \rangle \in E_D$ to obtain a tree-like (greedy) derivation with adequate labeling.

□

Definition 11. The tuple $\langle V, E_D, E_d, r, l, L \rangle$ and the order $O_S$ on the range of $l$ is called a decorated greedy tree-like derivation of $l(r)$ from $\Gamma$, iff, every application of $\supset$-introduction is greedy, $\set O_S(\langle v_1, v_2 \rangle)$ is the set of formulas from which the formula occurrence $v_1$ depends on, for every edge $\langle v_1, v_2 \rangle \in E_D$.

From the last two propositions, proposition 5 and 6, we obtain the following theorem.

Theorem 7. For every derivation $\Pi$ of $\alpha$ from $\Gamma$ that only has Greedy $\supset$-intro rules and $\supset$-Elim rules there is a decorated greedy tree-like derivation (DGTD) $\Pi_T$ of $\alpha$ from $\Gamma$.

Corollary 8. In the above statement, $\text{size}(\Pi) = \text{size}(\Pi_T)$, and if $\Pi$ is normal, then $\Pi_T$ is normal too.

We observe that the discharging edges $E_d$ provide information on which formula occurrence is discharged by which rule. This information is subdivised by the labeling $L$, as shown by the corollary below.

Corollary 9. Let $\langle V, E_D, E_d, r, l, L \rangle$ be a decorated greedy tree-like derivation, $O_S$ be the order on the range of $l$, and $L$ the associated labeling of dependencies. We can conclude that the information provided by $E_d$ can be determined using the other components of the tuple.

5 The Horizontal compressing procedure

This section defines the horizontal compression procedure and proves some properties about it. The compression follows the definition of horizontal collapse. A horizontal collapse applies to a dag-like decorated greedy derivation. It aims to identify two or more nodes in the rooted dag-like derivation at the same deduction level. The collapsing applies from the conclusion level, namely the zero level, towards the assumptions levels. When applied to tree-like rooted, and decorated, derivations, it yields dags instead of trees. The following definition formally defines this operation as a cases style definition. Forty rules form a case analysis that applies to a Dag-like derivation to yield a (new) dag-like derivation. The horizontal collapsing initially transforms tree-like derivations into dag-like derivations. Additional structure is needed to allow us to verify that a particular dag-like derivation is a (correct) derivation, indeed. We define a dag-like derivability structure as the underlying structure to encode dag-like derivations. Thus, a dag-like derivation is a dag-like derivability structure (DLDS) instance and a condition that should be true about this DLDS instance.

Definition 12 (Dag-like derivability structures DLDS). Let $\Gamma$ be a set of $M_\supset$ formulas and $O_\Gamma$ an arbitrary linear ordering on $\Gamma$ and $O^0_\Gamma = O_\Gamma \cup \{0, \lambda\}$ A dag-like derivability structure, DLDS for short, is a tuple $\langle V, (E^i_D)_{i \in O^0_\Gamma}, E_A, r, I, L, P \rangle$, where:

1. $V$ is a non-empty set of nodes;
2. For each $i \in O^0_\Gamma$, $E^i_D \subseteq V \times V$ is the family of sets of edges of deduction;

$\set O_\Gamma = \set{0, \ldots, n}$ for every $n \in O_\Gamma$
3. \( E_A \subseteq V \times V \) is the set of edges of ancestrality;
4. \( r \in V \) is the root of the DLDS;
5. \( l : V \rightarrow \Gamma \) is a function, such that, for every \( v \in V \), \( l(v) \) is the (formula) label of \( v \);
6. \( L : \bigcup_{i \in \mathcal{O}} E^i_D \rightarrow \mathcal{B}(\mathcal{O}_S) \) is a function, such that, for every \( \langle u, v \rangle \in E^i_D \), \( L(\langle u, v \rangle) \) is a bitstring.
7. \( P : E_A \rightarrow \{1, \ldots, ||\Gamma||\}^* \), such that, for every \( e \in E_A \), \( P(e) \) is a string of the form \( o_1; \ldots; o_n \), where each \( o_i \), \( i = 1, n \) is an ordinal in \( \mathcal{O}_\Gamma \);

Note that for each \( i \in \mathcal{O}^0 \), and, \( \langle u, v \rangle \in E^i_D \), \( i \) can be seen as the color of the edge \( \langle u, v \rangle \). Thus, each deduction edge is coloured with formulas from \( \Gamma \) or the 0 colours. A colour different from 0 is introduced every time a collapsing of nodes is performed. A tree-like greedy derivation has only 0 coloured deduction edges. The algorithm below is responsible for the horizontal compression. Afterwards we ask that the \( \text{Set}(\langle u, v \rangle) \), \( \langle u, v \rangle \in E^i_D \) is a dependency set and \( L \) is an adequate labeling, according to definition 9. Edges in \( E^i_D \) are the result of the collapse of the edges. The \( \lambda \) label should be read as an intentional absence of labelling. In the algorithm to verify that a DLDS is valid or not, the dependency set associated with an edge labelled with \( \lambda \) should be calculated dynamically, while all the other members of \( \bigcup_{i \in \mathcal{O}^0} E^i_D \) have it statically assigned.

---

**Algorithm 1** Horizontal Compression

- **Precondition:** A tree-like greedy derivation \( D \)
- **Ensure:** The DLDS that is \( D \) compressed

```
1: for \( \text{lev} \) from 1 to \( \text{h}(D) \) do
2: \hspace{3mm} for \( u \) and \( v \) at \( \text{lev} \) do
3: \hspace{6mm} \( \text{HCom}(u, v) \)
4: \hspace{3mm} end for
5: end for
```

---

### 5.1 The compression rules

Below we show 28 compression rules that define \( \text{HCom}(u, v) \). Some of the rules already used and presented in the example in section 2 are not repeated here. Some symmetric rules may be omitted. Each of them applies to a specific case, the DLDS \( D \), that matches the left-hand side. The effect of collapsing two nodes (vertexes) of a DLDS that matches the left-hand side of the rule produces a new DLDS depicted by the right-hand side of the rewriting rule. It is worth noting that every decorated greedy tree-like derivation is a DLDS having \( P(v) = \epsilon \), for every \( v \in V \). Figure [32] i.e., rule \( \text{R0IE} \) represents the rewrite rule that collapses the conclusion of an application of \( \text{\textgreater} \)-Intro with an application of \( \text{\textgreater} \)-Elim. Figure [33] i.e., rule \( \text{R0HE} \), represents the collapse of a hypothesis (assumption or top-formula) with the conclusion of a \( \text{\textgreater} \)-Elim. Figure [34] rule \( \text{R0IH} \), represents the conclusion of a \( \text{\textgreater} \)-Intro with a hypothesis. Finally, figure [35] rule \( \text{R0HH} \), represents the collapse of two hypotheses. Other combinations are: the collapse of the conclusions of two \( \text{\textgreater} \)-Elim rules; the collapse of a conclusion of a \( \text{\textgreater} \)-Elim with a \( \text{\textgreater} \)-Intro; the collapse of two conclusions of \( \text{\textgreater} \)-Intro rules; the collapse of the conclusion of a \( \text{\textgreater} \)-Elim with a hypothesis. Finally, we have the collapse of a hypothesis and a \( \text{\textgreater} \)-Intro. Of course, these are all the nine possibilities. We will see that we have three four classes

\[ \text{As an example, } \text{R}_{e}2EI \text{ is symmetric to } \text{R}_{e}2IE \]
of rules. The naming of the rules follow a schema. They are named as $\text{Rim}_i\text{m}_r$, where $i = 0 \ldots 3$ and $m_i, m_r \in \{I, E, H, \ast\}$, such that, $i$ is the type of the rule, $m_i$ and $m_r$ means the kind of the labels in the left and right nodes that will collapse. For example, the rule $\text{R0IE}$ is the official name of the rule in figure \[32\]. Note all names correspond to rules. For example there is no rule with the name $\text{R3II}$. This will be better explained in the sequel. It is worth noticing that when one of the nodes to be collapsed is already the result of a previous collapse then there are more than one rule applied to have it as conclusion. In this case we use the letter $X$ in the name formation of the rule. For example, $\text{R1XE}$ is the name of the rule that collapses a left node that is conclusion of an $\ast$-Elim. The rule $\text{R1XE}$ appears in figure \[37\]. There is a small variation in the naming of the rules. There are names such as $R_02m_1m_r$ and $R_2m_1m_r$. The indexes $v$ and $e$ indicates that the respective rule collapses only vertexes ($v$) or edges ($e$) too. See figures \[7\] and \[51\] for a better understanding of this. For example, the rule $\text{R0EE}$, used in the compressing example in figure \[3\] collapses the conclusions of two elimination rules.

Figure \[32\] shows a Type=$I$ rule, as defined below, named $\text{R0IE}$. We use this figure to re-emphasize how to read the pictorial representation of each horizontal compression rule. Both the left and the right-hand sides are subgraphs of, respectively, the DLDS $\mathcal{D}$ and $\mathcal{D}'$. The meaning of this rule is to 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.

\[\mathcal{D}' = \langle V', (E_D')_{i \in O_i'}, E_A', r', l', l'' \rangle,\]

considering $\mathcal{D} = \langle V, (E_D)_{i \in O_i}, E_A, r, l, l'', P \rangle$, and:

- $V' = V - \{v\}$,
- $(E_D')_{i \in O_i} = E_D - \{(u, \bullet_a), (v, \bullet_b), (p_2, v), (p_3, v)\}$,
- $(E_A')_{i \in O_i} = (E_A)_{i \in O_i} \cup \{(u, \bullet_a)\}$,
- $(E_D')_{i \in O_i} = (E_D)_{i \in O_i} \cup \{(u, \bullet_b)\}$,
- $(E_D')_{i \in O_i} = E_D$, for each $i > 2$,
- $E_A' = E_A \cup \{(\bullet_a, p_1), (\bullet_b, p_2), (\bullet_b, p_3)\}$,
- $l' = l'' V'$,
- $l' = l / (E_D')_{i \in O_i} [\langle u, \bullet_a \rangle \leftarrow \bar{c}_1 - \bar{p}_0; \langle u, \bullet_b \rangle \leftarrow \bar{b}_1 \lor \bar{d}_1]$, and finally,
- $P' = P[\langle \bullet_a, p_1 \rangle \leftarrow 1; \langle \bullet_b, p_2 \rangle \leftarrow 2; \langle \bullet_b, p_3 \rangle \leftarrow 2]$.  

In the left-hand side of the rule in figure \[32\], $p_i$, $i = 1, 3$, $u$ and $v$ are different nodes in the subgraph, such that $l(v) = l(u)$, 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 $\text{Sets}(\bar{c}_1)$. The absence of a label on an edge indicates that the edge is unlabeled. 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. $\bullet$ is label different nodes always. In figure \[32\] the bullets label different nodes. In the set-theoretical semantics of the rules, explained in the previous paragraph we use $\bullet_a$ and $\bullet_b$ to reference the the two different nodes. Edges that belong to $E_D^1$ have the colour $i$; this is the red ordinal number $1, \ldots, n$ 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 in
the right-hand side of R0IE. In the sequel, we will find the label \( \lambda \) assigned to some edges, i.e., those belonging to \( E^\lambda_D \). As already mentioned in the paragraph after definition 12, the members of \( E^\lambda_D \) are edges that have the dependency set calculated by the main Algorithm in appendix A that verifies if a DLDS is valid. Moreover, a node in the left-hand side shows all the edges outgoing or incoming it. If there is no incoming edge drawn then there is no drawn at all.

There is a particular case that is present in some rules, namely the rules in figures 33 to 35, 40, 42, 19, 44, 46, 47 and 49. This particular feature has to do with the fact that at least one of collapsed nodes is labeled with a top-formula, that is, an hypothesis. In each of these rules, the right-hand side contains a marking \( h \) indicating that at least of one of the collapsed nodes is an hypothesis.

Regarding to understanding the remaining rules, we do not need any additional explanation, and the reader should read them as explained in the precedent paragraphs. We advise the reader that all the rules in the graphical representation in the sequel assume that nodes and edges drawn in different positions are different always. For example, in figure 32, i.e R0IE, \( p_0, p_1, p_2, p_3, u, v \) and the two bullets (●) below them are all pairwisely different nodes. Dashed lines represent possible paths in the graph.

5.2 On the classification and type of rules

Let \( \beta \) be formula occurrence in an ND derivation in \( M^- \), it is a hypothesis or the conclusion of a \( \supset \)-application or the conclusion of a \( \supset \)-Elim. Nine is the total number of possible pairs. Consider, for example, the rules shown in figures 32, 33, 34, 35 and 50, i.e, rules R0IE, R0HE, , and together with the remaining four cases. We consider that the full subgraph of the DLDS, to which they apply, determined by the set of nodes reachable from the nodes in their respective left-hand graph, does not have any collapse yet. For example in the context of the HC-compression, when applying R0IE, any node below the two bullets that are reachable from some of them are not collapsed node. These nine rules are what we call type-0 rules. In contrast, the rules in figure 37 and 38 are type-I rules. They consider, i.e. have as a precondition, that their respective left-hand side graph represents a pair of nodes to collapse, such that exactly one of them is already the result of a previous collapse. Remember that the collapses follow the algorithm from the bottom up and left to right. There are three possible type-I rules; depending on which rule, \( \supset \)-Intro or \( \supset \)-Elim, the right node to collapse is the conclusion of or if it is a hypothesis. The rules in figures 39 to 19, 47 and 49 are of type-II. In contrast with type-0 and type-I, their precondition is that (at least one of) the nodes to be collapsed is target of an ancestor edge, i.e. a member of \( E^A \). What we stated is equivalent to saying that their respective sons has already been collapsed before, according to the order of execution of algorithm 1. The type-II rules are of two kind. Either they collapse only the vertexes or vertexes and edges of the DLDS. The letter \( v \) and \( e \) indicates what is the kind in their respective rule names. There are nine possible combinations, as in type-0, for each kind. We do not find it necessary to show the remaining six cases to avoid very similar repeating rules. The type-II rules are listed in definition below. They represent the collapse of two nodes that are targets of ancestor edges. In opposition to the type-I rules, in the next level below the nodes that will be collapsed, there is no node resulting from an already collapsed pair of nodes. These already collapsed nodes occur below this next level. The total number of rules of the type-II is 18; in the listing of rules i the sequel, some of them may be omitted, but we have to consider them too. We have also the type-III rules that collapses one node that is already collapsed with other node not collapsed yet, in the context of the existence of A-edges arriving on the collapsed nodes. These are the rules with names R\( _v \), R\( _e \), R\( _v \), R\( _e \), R\( _v \), R\( _e \), R\( _v \), R\( _e \), R\( _v \), R\( _e \).

We also consider the classification of the rules according to the effect that they have on the A-
edges, i.e., if they move ancestor edges downwards or upwards. The MDE-rules, for Moving Down Edges, are used to collapse at least one top-node in a pair of nodes that have the same formula labeling them. We have already seen some of these rules in section 2, rules $R_{v2HH}$, $R_{e2HH}$ and $R_{e2XH}$ are examples of MDE rules. The MUE rules, for Move Upwards Edges, are those used to move the A-edge up, when collapsing nodes that are target of these edges.

**Definition 13** (Types of rules). The rules of compression are classified in types. The classification is the following:

- **Type-0** $R_{0HH}$, $R_{0HI}$, $R_{0HE}$, $R_{0IH}$, $R_{0IE}$, $R_{0II}$, $R_{0EH}$, $R_{0EI}$, $R_{0EE}$;
- **Type-I** $R_{1XH}$, $R_{1XE}$, $R_{1XI}$;
- **Type-II** $R_{v2EI}$, $R_{e2EI}$, $R_{v2EE}$, $R_{e2EE}$, $R_{v2EH}$, $R_{e2EH}$, $R_{v2II}$, $R_{e2II}$, $R_{v2IE}$, $R_{e2IE}$, $R_{v2HI}$, $R_{e2HI}$, $R_{v2HE}$, $R_{e2HE}$, $R_{v2HH}$, $R_{e2HH}$;
- **Type-III** $R_{v3XH}$, $R_{e3XH}$, $R_{v3XE}$, $R_{e3XE}$, $R_{v3XI}$, $R_{e3XI}$.

Other dimension for classifying the compression rules is by the effect that they make in the A-edges.

**Definition 14** (Moving up and down A-edge rules). The rules of compression are also classified according their effects on the ancestor edges:

- **MUE- Moving Up Edges** $R_{v2EI}$, $R_{e2EI}$, $R_{v2EE}$, $R_{e2EE}$, $R_{v2II}$, $R_{e2II}$, $R_{v2IE}$, $R_{e2IE}$, $R_{v2HI}$, $R_{e2HI}$, $R_{v2HE}$, $R_{e2HE}$, $R_{v2EH}$, $R_{e2EH}$, $R_{v2IH}$, $R_{e2IH}$, $R_{v2II}$, $R_{e3XH}$, $R_{e3XH}$, $R_{v3XE}$, $R_{e3XE}$, $R_{v3XI}$, $R_{e3XI}$, $R_{v3XH}$, $R_{e3XH}$;
- **MDE- Moving Down Edges** $R_{v2HH}$, $R_{e2HH}$, $R_{v2XH}$, $R_{v2XH}$.

---

**Figure 32:** (a) $u$ and $v$ collapse

(b) After collapse $HCom(u,v)$
On the horizontal compression

\[ u = \begin{array}{c} p_2 \\ \downarrow \bar{b}_1 \\ \downarrow \bar{b}_2 = b_1 \lor d_1 \end{array} \quad \begin{array}{c} p_3 \\ \downarrow d_1 \\ \downarrow \bar{d}_1 \end{array} \]

Figure 33: (a) \( u \) or \( v \) assumptions coll.

\[ \bar{l}(\tilde{u}) \quad \bar{l}(\tilde{v}) \]

(b) After collapse \( HCom(u, v) \)

\[ \bar{c} = c_1 - p_0 \]

\[ \bar{l}(\tilde{v}) \]

Figure 34: (a) \( u \) and \( v \) collapse

\[ \bar{l}(\tilde{v}) \]

(b) After collapse \( HCom(u, v) \)

\[ \bar{c} = c_1 - p_0 \]

\[ \bar{l}(\tilde{v}) \]

Figure 35: (a) \( u \) and \( v \) collapse

\[ \bar{l}(\tilde{v}) \]

(b) After collapse \( HCom(u, v) \)

\[ \bar{c} = c_1 - p_0 \]

\[ \bar{l}(\tilde{v}) \]

Figure 36: (a) \( u \) and \( v \) collapse

(b) After collapse \( HCom(u, v) \)
Figure 37: (a) $u$ and $v$ collapse (b) After collapse $\text{HCom}(u,v)$

Figure 38: (a) $u$ and $v$ collapse (b) After collapse $\text{HCom}(u,v)$
On the horizontal compression

\[
\bar{c} = \bar{c}_1 - \bar{p}_1
\]

\[
\bar{b}_2 = \bar{b}_1 \downarrow \bar{d}_1
\]

\[
\bar{c} = i \neq j
\]

\[
\bar{b} \; [0; s_1] \quad [0; s_2]
\]

\[
\bar{d} \; [0; s_1] \quad [0; s_2]
\]

\[
\bar{c}_1 \downarrow \bar{d}_1
\]

\[
\bar{b}_1 \downarrow \bar{d}_1
\]

\[
HCom(\bar{u}, \bar{v}) \quad \Rightarrow \quad \bar{c}_1 = \bar{c}_1 - \bar{p}_1
\]

\[
HCom(\bar{u}, \bar{v}) \quad \Rightarrow \quad \bar{c}_1 = \bar{c}_1 - \bar{p}_1
\]

Figure 39: (a) \(u\) and \(v\) collapse (b) After collapse \(HCom(u, v)\)

Figure 40: (a) \(u\) and \(v\) collapse (b) After collapse \(HCom(u, v)\)
\[ \overline{c} = \overline{c_1} \lor \overline{c_2} \]

Figure 41: (a) \( u \) and \( v \) collapse (b) After collapse \( HCom(u, v) \)

\[ \overline{h} \]

\[ \lambda \]

Figure 42: (a) \( u \) and \( v \) collapse (b) After collapse \( HCom(u, v) \)
On the horizontal compression

\[ \bar{c}_1 - p_1 \downarrow \bar{b}_1 = b_1 \lor \bar{d}_1 \]

\[ \rightarrow HCom(u, v) \]

\[ HCom(u, v) \rightarrow R_v2IE \]

\[ \bar{c}_1 - p_1 \downarrow \bar{b}_1 = b_1 \lor \bar{d}_1 \]

\[ \rightarrow HCom(u, v) \]

\[ HCom(u, v) \rightarrow R_v2IH \]

Figure 43: (a) \( u \) and \( v \) collapse (b) After collapse \( HCom(u, v) \)

Figure 44: (a) \( u \) and \( v \) collapse (b) After collapse \( HCom(u, v) \)
\[ u = v \]

Figure 45: (a) \( u \) and \( v \) collapse

\( \text{R}_v2\text{XH} \)

\[ (i; s) \text{HCom}(u, v) \Rightarrow \]

\[ \text{HCom}(u, v) \]

\[ [i; s] \]

\[ [0; j; s_2] \]

\[ \bar{l}(u) \]

\[ \bar{l}(v) \]

\[ h \]

\[ i \]

\[ j \]

\[ \bar{b}_1 \]

\[ \bar{d}_1 \]

\[ p_2 \]

\[ p_3 \]

\[ s_1 \]

\[ s_2 \]

\( \text{HCom}(u, v) \)

\[ \text{R}_v2\text{HE} \]

Figure 46: (a) \( u \) and \( v \) collapse

(b) After collapse \( \text{HCom}(u, v) \)
On the horizontal compression

\[ H\text{Com}(u,v) \Rightarrow \begin{array}{l}
\bar{c}_1 = u = v \\
\bar{b}_2 = \bar{b}_1 \lor \bar{d}_1 \\
\end{array} \]

\( R_{e3XH} \)

Figure 47: (a) \( u \) and \( v \) collapse  
(b) After collapse \( H\text{Com}(u,v) \)
\begin{align*}
\overline{b}_1 &= \overline{b}_1 - \text{ant} (\tilde{\ell}(\overline{v})) \\
\ell(v) &= \frac{m+1}{k} \\
\text{HCom}(u, \overline{v}) &= \Rightarrow R_{\overline{v}3XI}
\end{align*}

Figure 48: (a) $u$ and $v$ collapse (b) After collapse $HCom(u, \overline{v})$
On the horizontal compression

Figure 49: (a) $u$ and $v$ collapse

(b) After collapse $HCom(u, v)$

**Obs:** In rules $R_e3XH$, $R_v3XH$ and $R_v2XH$ the node $u$ in the left hand-side may be marked with $h$, as in the right hand-side.

Figure 50: (a) $u$ and $v$ collapse

(b) After collapse $HCom(u, v)$

**Remark.** We have to remark that in some of the rules above, we use a shortened form of the labels assigned by the function $P$, the paths. This is due to have a lighter visualization of the rules. The paths are shortened by omitting sequences of zeros (0s) in the path. When showing concrete examples, as in the appendix, we do not use the shortened version.
6 More on DGTD, DLDS and compression rules soundness preservation

We need some definitions concerning the relation to DLDSs and the rules listed in the above subsection 5.1. They are in the sequel.

Definition 15 (INS-Incoming Deductive Edges of a node). Given a DLDS $D$ of $\alpha$ from $\Gamma$ and a node $k \in D$, the deductive in-degree of $k$ is defined as $INS(k) = \{ f : f \in E_D^i, i \in \mathcal{O}(\Gamma \cup \{ \alpha \})^0 \land target(f) = k \}$.

Definition 16 (OUTS-Outcoming Deductive Edges from a node). Given a DLDS $D$ of $\alpha$ from $\Gamma$ and a node $k \in D$, the deductive out-degree of $k$ is defined as $OUTS(k) = \{ f : f \in E_D^i, i \in \mathcal{O}(\Gamma \cup \{ \alpha \})^0 \land source(f) = k \}$.

Note that for any node $k$, both sets, $INS(k)$ and $OUTS(k)$, do not take into account the ancestor edges in their definition. We remember that the members of $E_A$ are not deductive edges. However, they play an important, although auxiliary role in the logical reading of any DLDS. A trivial observation is that there is a natural map from DGTDs to DLDSs.

Definition 17. Let $T = \langle V, E_D, E_d, r, l, L \rangle$ be a DGTD. Let $\mathcal{O}_S$ be the order on the range of $l$, provided by $T$ itself and $\Gamma$ the set of leaves in $T$. Let $\text{Dag}(T)$ be $\langle V, (E_D^i)_{i \in \mathcal{O}_S}, E_A, r, l, P \rangle$, where $E_D^0 = E_D$, $E_D^i = \emptyset$, for all $i \neq 0$ and $E_A = \emptyset$, $P = \emptyset$.

It is easy to verify that $\text{Dag}(T)$ is well-defined, and hence it is a DLDS, for every DGTD $T$. Thus, we have the mapping $\text{Dag}$ from DGTDs to DLDSs. When reading a DGTD from top to bottom in a tree-like Natural Deduction derivation, there is at most one path from any top-formula occurrence to any other formula occurrence in the derivation. The following fact is an easy consequence of $\text{Dag}$’s definition above.
Proposition 10. Let $T$ be a DGTD. For every pair of nodes $v$ and $u$ in $T$, there is a bijection between the paths from $v$ to $u$, in $T$, and 0-paths, i.e., using only members of $E_D^0$, from $v$ to $u$ in $\Dag(T)$. Moreover, the dependency sets, assigned by $L$, in both structures, are equal for every edge $\langle v,u \rangle \in E_D^0$.

From the definition of the mapping $\Dag$, we can see that there is no path in the DLDS $\Dag(T)$ with colors different from 0, due to $E_A^i = \emptyset$, for all $i \neq 0$. Moreover, there are no paths in $E_A$, for $E_A = \emptyset$.

The following definition shows how the information stored in the component $P$, the seventh one, the last, of any DLDS is used as a relative address for nodes in it. It uses:

Definition 18. $el(\{e\}) = e$ and $el(S) = \bot$, if $S$ is not $\{a\}$ for some $a$.

Definition 19 (relative address of a node). Let $D$ be a DLDS of $\alpha$ from $\Gamma$ and let $\gamma \in \mathcal{O}(\Gamma \cup \{\text{alpha}\})^*>^*$ we say that $\gamma$ is the address of a node $v \in D$ relative to a node $u \in D$ iff the following algorithm returns $v$ on input $\gamma$, $D$ and $u$. The underlying idea is that $\gamma$ provides information on every branching in the path from $u$ downwards $v$. Each ordinal from left to right in $\gamma$ indicates which branch to take in.

Algorithm 2 finding a node from its relative address and origin of the path

Precondition: $u$, the origin, $D$, the DLDS, and the relative address $\gamma$

1: $b \leftarrow u$
2: $\text{glues} \leftarrow \gamma$
3: while $\text{glues} \neq e$ do
4: if $\text{size}(\text{OUTS}(b)) = 1$ then
5: $g \leftarrow el(\text{OUTS}(b))$
6: $b \leftarrow \text{target}(g)$
7: else if $\text{size}(\text{OUTS}(b)) > 1$ and $\text{size}(\{e/(e \in \text{OUTS}(b)) \land (\text{color}(e) = \text{head}(\gamma))\}) = 1$ then
8: $g \leftarrow el(\{e/(e \in \text{OUTS}(b)) \land (\text{color}(e) = \text{head}(\gamma))\})$
9: $b \leftarrow \text{target}(g)$
10: $\text{glues} \leftarrow \text{rest}(\gamma)$
11: else
12: Return false
13: end if
14: end while
15: Return $b$

For defining when a DLDS corresponds to a valid derivation we need the definition of Deductive path below.

Definition 20 (Deductive path). Given two nodes $v_1$ and $v_2$ in a VLDS $D = \langle V, (E_D^i)_{i \in \{\lambda\} \cup \mathcal{O}_D^i}, E_A, r, l, L, P \rangle$, we call a path $e_1, e_2, \ldots, e_n$ from $v_1$ to $v_2$ a deductive path, iff, for each $p = 1, \ldots, n$, $e_p \in \bigcup_{i \in \{\lambda\} \cup \mathcal{O}_D^i} E_D^i$.

In particular, if $e_1, e_2, \ldots, e_n$ is a deductive path from $v_1$ to $v_2$ and there is $i \neq 0$, such that $e_j \in E_D^i$ or $e_j \in E_D^\lambda$, for some $0 \leq j \leq n$, then the path is a mixed deductive path from $v_1$ to $v_2$.

Given a DLDS $D = \langle V, (E_D^i)_{i \in \mathcal{O}_D^i}, E_A, r, l, L, P \rangle$ and a node $w \in V$, we define

$Pre(w) = \{v : \text{There is a deductive path from } v \text{ to } w\}$
the set of nodes that are linked to \( w \) by some deductive path. Moreover we have the set of top nodes of a DLDS.

\[
Top(w) = \{ v : v \in Pre(w) \text{ and, there is no } v' \in V, \langle v', v \rangle \in (E^i_D)_{i \in O_i} \text{ or } v \text{ is marked as hypothesis} \}
\]

Finally, the set of full deductive paths reaching to \( w \in V \) is:

\[
DedPaths(w) = \{ \langle e_1, \ldots, e_n \rangle : e_1 \ldots e_n \text{ is a deductive path, source}(e_1) \in TopNode(w) \text{ and target}(e_n) = w \}
\]

We introduce the relation \( \sim \) between dependency sets.

**Definition 21.** For any pair of dependency sets \( \bar{b} \) and \( \bar{c} \), \( \bar{b} \sim \bar{c} \) holds, if and only if, \( \bar{c} = \bar{b} \) or \( \bar{c} = \lambda \) or \( \bar{b} = \lambda \).

**Definition 22.** Given a DLDS \( D = \langle V, (E^i_D)_{i \in O_i}, E_A, r, l, L, P \rangle \) and a node \( w \in V \), we define \( Flow(D, w) \) as a function from \( Pre(w) \) into \( \wp((O^B_S)^* \times B(O^S_S)) \), such that:

\[
Flow(D, w)(v) =
\]
In the end of section 2 we use the function \textit{Flow} in a compressed \textit{DLDS} to illustrate how it works.
by calculating the dependency sets inside it. In appendix\[B\] we also illustrate how the function Flow provides the Dependency sets of a compressed DLDS. Flow is used to in a condition that is part of the validity criteria for DLDS below.

**Definition 23.** Given a structure $D = \langle V, (E^i_D)_{i \in O^i}, E_A, r, l, L, P \rangle$, we say that it is a valid DLDS, iff. the following conditions hold on it:

- **Color-Acyclicity** For each $i \in O^i$, $E^i_D$ does not have cycles;
- **Leveled-Colored** The rooted sub-dag $\langle V, (E^i_D)_{i \in O^i}, r \rangle$ is leveled;
- **Ancestor-Edges** For each $\langle v_1, v_2 \rangle \in E_A$, the level of $v_1$ is smaller than the level of $v_2$;
- **Ancestor-Backway-Information** For each $\langle v_1, v_2 \rangle \in E_A$, $P(\langle v_1, v_2 \rangle)$ is the relative address of $v_1$ from $v_2$;
- **Simplicity** The rooted sub-dag $\langle V, (E^i_D)_{i \in O^i}, r \rangle$ is a simple graph, i.e, for each pair of nodes $v_1$ and $v_2$, there is at most an $i \in O^i$, such that $\langle v_1, v_2 \rangle \in E^i_D$;
- **Ancestor-Simplicity** The sub-dag $\langle V, E_A \rangle$ is a simple graph;
- **Non-Nested-Ancestor-Edges** For each $\langle v_1, v_2 \rangle \in E_A$, there is no $w$ in the path from $v_2$ to $v_1$, determined by $P(\langle u, v \rangle \in E_A)$, such that $\langle w, z \rangle \in E_A$, for some $z \in E_A$.
- **CorrectRuleApp** For each $w \in V$, Flow$(D, w)(v)$ is well-defined for each $v \in Pre(w)$. Moreover, for each $w$ and $v$, Flow$(D, w)(v)$, with $v \in Pre(w)$, we have:
  - If Flow$(D, w)(v) = \{b, p\}$ then OUT$(v) = \{v, v'\}$ and the color of $\langle v, v' \rangle$ is head$(p)$, i.e., $\langle v, v' \rangle \in E^i_D$ and $\langle \bar{b}, \bar{v} \rangle = \bar{L} \langle \langle v, v' \rangle \rangle$; and,
  - If Flow$(D, w)(v) \neq \emptyset$ and it is not a singleton then for each $\Phi_i = \{(\bar{b}, p) \in Flow(D, w)(v) : \text{head}(p) = i\}$:
    1. If $\Phi_i \neq \emptyset$ then there is only one $v' \langle v, v' \rangle \in E^i_D$ and if $\Phi_i = \{(\bar{b}, p)\}$ then $\bar{L} \langle \langle v, v' \rangle \rangle = \bar{b}$ else $\bar{L} \langle \langle v, v' \rangle \rangle = \lambda$, and;
    2. If $\Phi_i = \emptyset$ then there is no $v' \in V$, such that, $\langle v, v' \rangle \in E^i_D$.

It is worth noting that in item CorrectRuleApp, the verification that a rule application is correct involves, among other things, finding out that the premises agree with the conclusion and checking that the dependency sets are correctly assigned, this is the main role of function Flow.

Each of the items in definition\[23\] is an invariance property that should be preserved by all compression rules applications. This is what theorem\[21\] says. The lemmas in sub-section\[6.1\] prove that the HC rules preserve each condition in the definition\[23\]. The proof of theorem\[21\] uses all of them.

### 6.1 HC rules and DLDS validity preservation

In sub-section\[5.2\] we discuss the types of the rules, namely, type-0, type-I, type-II and type-III listings point out which rule belongs to each of the types. Below we provide applicability conditions for each of the types.

**Definition 24** (Rule applicability conditions for type-0 rules). A type-0 rule is applicable to a pair $u, v \in V$ in a DLDS $D = \langle V, (E^i_D)_{i \in \Lambda} \cup O^i, E_A, r, l, L, P \rangle$, iff, $l(u) = l(v)$, there is no $e \in E_A$, such that target$(e)$ is one of the nodes in the left-hand side of the rule, and, for all $e \in (E^i_D)_{i \in \Lambda} \cup O^i$, if target$(e)$ is some node in the left-hand of the rule then $e \in E^0_D$.
Definition 25 (Rule applicability conditions for type-I rules). A type-I rule is applicable to a pair $u, v \in V$ in a DLDS $D = \langle V, \langle E_D \rangle_{i \in \{A\} \cup \{O\}}, A, r, l, L, P \rangle$, iff $l(u) = l(v)$, (1) there is no $e \in E_A$, such that $\text{target}(e)$ or $\text{source}(e)$ is one of the nodes linked to $v$ in the left-hand side of the rule, and; (2) $u$ is a node that it is already the result of a collapsing, i.e., the nodes linked to $u$ are sources or targets of an ancestor rule $(E_A)$, and, for each $e \in \text{OUTS}(u)$, there is $i \neq 0$, such that $e \in E^i_D$.

The rules of type-I are rules in figures 37, 38 and 36 are all the rules of type-I. Only these rules satisfy the condition above.

Definition 26 (Rule applicability conditions for type-II rules). A type-II rule is applicable to a pair $u, v \in V$ in a DLDS $D = \langle V, \langle E_D \rangle_{i \in \{\lambda\} \cup \{O\}}, A, r, l, L, P \rangle$, iff $l(u) = l(v)$, both $u$ and $v$ are target of an edge $e \in E_A$, i.e., $\text{target}(e) = v$ or target$(e) = u$.

For example, rules in figures 39 to 44 are of this kind.

Definition 27 (Rule applicability conditions for type-III rules). A type-II rule is applicable to a pair $u, v \in V$ in a DLDS $D = \langle V, \langle E_D \rangle_{i \in \{\lambda\} \cup \{O\}}, A, r, l, L, P \rangle$, iff $l(u) = l(v)$, exactly one of $u$ or $v$ are target of an edge $e \in E_A$, i.e., target$(e) = v$ or target$(e) = u$. If $v$ is the target of an $A$-edge, then $u$ is a node that is already collapsed by a rule of type-III.

All type-III rules satisfy above condition.

In this section, we show the preservation of each condition in the definition of validity of DLDS, definition 22.

We should observe that the HC algorithm applies the HC rules from the lowest to the highest levels and in each level from left to right.

Definition 28 (Application of HC rule in algorithmic position). Given a DLDS $D$, we say that an application of a HC rule in $D$ is in algorithmic position iff this application collapses two nodes, $v$ and $u$, in level $j$, such that, above $j$ there is no sub-graph originated from the collapse of two nodes by any application of any HC rule and, there is no application that collapses two nodes by any HC rule in level $j$ righter than both $v$ and $u$ either.

Remark. Due to the fact that the HC algorithm applies in any level from left to right, if a HC application in a DLDS $D$ is in algorithmic position with respect to the nodes $v$ and $u$ in level $j$, then only the leftmost of $u$ and $v$ can be the result of a previous HC rule application. The lemma below illustrates how the mechanism of the ancestor edges updating works. Before this lemma statement, we need some definitions.

Definition 29 (Partially Compressed DGTD). We say that a DLDS $D = \langle V, \langle E_D \rangle_{i \in \{\lambda\} \cup \{O\}}, A, r, l, L, P \rangle$ is a partially compressed DGTD, iff, there is a DGTD $T$, such that, $D$ is the result of the application of some steps in line 3 of the algorithm 7 to $T$.

Any DGTD is a partially compressed DGTD, and the final result of the algorithm, the so-called totally compressed DGTD is also a partially compressed DGTD. We observe that the horizontal compression algorithm 1 halts when submitted to any DGTD, see theorem 24. Any partially compressed DGTD that is not the final result of the application of the algorithm 1 to it has an algorithmic position to collapse two nodes $u$ and $v$. For technical reasons, the following lemma considers only rules that are not MDE. The format of a DLDS that cannot serve as input to a non-MDE rule has every level, but the last one, with no two nodes labelled with the same formula. This is what we obtain as an almost compressed DLDS. This kind of DLDS is called MUE$^+$-compressed DLDS. Only the top-formulas of the derivation may have repetitions in a same level.
Lemma 11. Let $D = (V, (E_D^i)_{i \in \{0\} \cup I}, E_A, r, I, L, P)$ be a valid partially compressed DGTD and $u_{ap}$ and $v_{ap}$ nodes in level $j > 0$, such that, there is at least one compression rule that is not MDE that can be applied to $D$ in algorithmic position to collapse $u_{ap}$ and $v_{ap}$. Consider a node $w$ in level smaller than $j$, such that, there is a mixed deductive path from the node $u_{ap}$ to $w$. Hence, if $w = source(e)$ and $e \in E_A$ then $target(e) = u_{ap}$ or there is $e' \in E_D^0$, such that $target(e) = source(e')$ and $target(e') = u_{ap}$. Moreover, if $u_{ap}$ is marked as hypothesis then there is $e' \in E_D^0 \cup E_D^1$, such that, $target(e) = target(e')$ and $source(e') = u_{ap}$.

We should observe that the statement of the lemma involves hold for every lefthand side of the compression rules.

Proof of lemma \[ D \] is a partially compressed DGTD, so there is a DGTD $T$, such that, $D$ is the result of the application of some steps in line 3 of the horizontal compression algorithm to $T$. We prove the lemma by induction on the lexicographic pair $(I, II)$, where $I$ is the number of ancestor edges in $D$, and $II$ is the number of steps executed by the horizontal compression algorithm on $T$ to obtain $D$. The base is $card(E_A) = 0$ when there is no ancestor edge and no horizontal compression rule was applied. In this case, the statement trivially holds. For the inductive case, $card(E_A) = n > 0$, there is a rule that can be applied to $D$ in algorithmic position to collapse $u_{ap}$ and $v_{ap}$, such that $u_{ap}$ and $v_{ap}$ are nodes in level $j > 0$. Let $(card(E_A), II_D)$ be the lexicographic pair of $D$. Thus, $II_D$ is the number of horizontal compression rules that results in $D$. So, some rule $R$ was applied to a previous DLDS $D'$ resulting into $D$. $R$ may or may not have created an ancestor edge in $D$. Thus, we have two cases:

1. $card(E_A) = card(E_A')$, where $E_A'$ is the set of ancestor edges in $D'$. In this case, the inductive hypothesis holds for $D'$, with pair $(card(E_A), II_D - 1)$. Since $D'$ is valid, due to the validity of $D$, by inductive hypothesis $D'$ satisfies the property in the lemma statement.

2. $card(E_A') < card(E_A)$. In this case, $R$ adds at least one ancestor edge to $D'$ to have $D$, so the inductive complexity of the former is smaller than the latter. We observe that $D'$ is valid as a consequence of the validity of $D$. Thus, by the inductive hypothesis, the property stated by the lemma holds for $D'$

It remains to analyse the changes that the application of the rule $R$ makes in the ancestor edges of $D'$, case 1 and the insertion of new ancestor edges by $R$, i.e. case 2. Separating the analysis according to the rules, we have to consider the type-0 and type-I rules, since they are rules that only add ancestor edges; hence, we are in the case 2 analysis. All the ancestor edges inserted in $D'$, resulting in $D$ satisfy the property stated in the lemma. The type-II rules that are not MDE change existing ancestor rules, they concern case 1 analysis. We can inspect the changes determined by each of these rules and conclude that the property stated in the lemma also hold in this case. For the sake of showing the need for the hypothesis that $D$ is valid, we describe in more detail rules $\text{R01E (case 2 above)}$ and $R_{q2IH}$ (case 1 above) argumentations, in the following two paragraphs.

By inductive hypothesis, the property stated by the lemma holds for $D'$ and suppose that rule $R$ is $\text{R01E}$ below in figure 52. We can observe that the dag in the lefthand side of the rule is a subgraph in $D'$, that $D'$ satisfies all validity conditions in definition 23 and the righthand side is subgraph of $D$. It is easy to see that the lemma property holds for $D$. The addition of the ancestor edges, in blue, after the application of $\text{R01E}$ satisfies this condition. By the validity of $D'$ and the lemma property holding on $D'$, we can affirm that there is no $w$ in a level below $u$ (and $v$) and a mixed deductive path from either one of $p_1$, $p_2$ or $p_3$, such that there is $e \in E_A$ and $source(e) = w$. If this is the case then $target(e) = u$, or $target(e) = \bullet$ or $target(e) = p_i$, for some $i$, since $D'$ satisfies the lemma property.
However, by the conditions of applicability of \textbf{R01E} all the conditions on the existence of \( w, e \) and their consequences are contradictory to the fact that \textbf{R01E} is applied to \( D' \) resulting in \( D \). Thus, there is no \( w \) and \( e \in E_A \), such that \( \text{source}(e) = w \). From this we conclude that the application of \textbf{R01E} to \( D' \) results in \( D \) satisfying the lemma.

\[
\bar{c} = c_1 - p_0 \parallel \bar{b}_2 = b_1 \lor d_1
\]

Figure 52: (a) \( u \) and \( v \) collapse (b) After collapse HCom\((u, v)\)

By the inductive hypothesis, the property stated by the lemma holds for \( D' \) and suppose that rule \( R \) is \textbf{R\_e\_2IH} below in figure 53. We can observe that the dag in the lefthand side of the rule is a subgraph in \( D' \), that \( D' \) satisfies all validity conditions in definition 23, and the righthand side is a subgraph of \( D \). By an argument analogous to the case of \textbf{R\_e\_2IH} application above, we can conclude that there is no \( w \) below and \( e \in E_A \) below the \( u \) (and \( v \)) level with \( \text{source}(e) = w \), but the ancestor edges (blue in the picture) in the lefthand side of figure 53. This is also a consequence of the applicability conditions of rule \textbf{R\_e\_2IH}. Due to this, we can conclude that the application of \textbf{R\_e\_2IH} to \( D' \) results in \( D \) satisfying the lemma. The ancestor edges new in the righthand side of \textbf{R\_e\_2IH}, both of them satisfy the property of the lemma. In particular, the part of the property related to deductive edges labelled with the \( \lambda \) holds.

Finally, the argumentation for proving that all type-I, type-II and type-0 rules that are not \textbf{MDE} also obtain \( D \) holding the lemma’s property are a combination of the cases 2 and 1 analysis. Thus, we have that the property stated by the lemma holds in the inductive case too. This concludes the proof of the lemma. Moreover the type-III rules \textbf{R\_e\_3XE}, \textbf{R\_e\_3XE}, \textbf{R\_e\_3XI} and \textbf{R\_e\_3XI} that are \textbf{MUE} rules have arguments that are also combination of the cases 2 and 1 analysis.

Q.E.D.

The following lemmas correspond to the preservation of each condition in definition 23.

\textbf{Lemma 12 (Color-Acyclicity).} Given a valid \textbf{DLDS} \( D = \langle V, (E_D^l)^{\lambda \in \{\lambda\} \cup \Omega_l}, E_A, r, l, L, P \rangle \), and \( u \) and \( v \) in level \( j \), such that \( l(u) = l(v) \). Let \( R \) be any \textbf{HC} rule, such that, \( R \) applies in algorithmic position to \( u \) and \( v \) resulting in a \textbf{DLDS} \( D' \). Thus, condition Color-Acyclicity holds on \( D' \).

\textbf{Proof.} Let \( D = \langle V, (E_D^l)^{\lambda \in \{\lambda\} \cup \Omega_l}, E_A, r, l, L, P \rangle \) be a valid \textbf{DLDS}, \( j \) be a level in \( D \) and \( R \) a \textbf{HC} rule in algorithmic position that collapses \( u,v \in V \). We prove that the resulting \( D' \) \textbf{DLDS} satisfies Color-Acyclicity by observing that:

- If \( R \) is a type-0 rule, then both nodes, \( u \) and \( v \) do not result from collapses by any previous \textbf{HC} rule application. If \( u \) or \( v \) were a result of any \textbf{HC} rule application, then there would
be ancestor edges arriving at least at one of them or at some of the nodes linked to them from above. Other case to consider is when \( v \) or \( u \) are marked with \( h \) due to be the result of application of rules \( R0HE, R0IH, R0HH, R1XH, R2IH, R2HE, R3HE, R3XH \).

Observing that since \( D \) is a valid DLDS then it satisfies Color-Acyclicity, so by inspection on each rule’s resulting \( D' \) we observe that the new edges, inserted by the right-hand side of the rule, cannot form a cycle. The main reason is that the direction of the coloured deduction edges is from top to bottom, and they do not link a node to itself either. Moreover, any edge in \( E_A \) direction is from the bottom-up. They do not form a cycle either. The same argumentation is used to type-I rules that also creates A-edges.

- The rules of type-II and type-III are similarly treated. We need to observe only that the left-hand side has no cycle for each colour \( i \), then the graph resulting from the insertions and changings described by the right-hand side is also acyclic for each \( i \). This argumentation also serves to the MUE and MDE that moves A-edges.

Q.E.D.

**Lemma 13 (Leveled-Colored).** Given a valid DLDS \( D = \langle V, (E_D^i)_{i \in \{\lambda\} \cup O^p}, E_A, r, l, L, P \rangle \), and \( u \) and \( v \) in level \( j \), such that \( l(u) = l(v) \). Let \( R \) be any HC rule, such that, \( R \) applies in algorithmic position to \( u \) and \( v \) resulting in a DLDS \( D' \). Thus, condition \textit{Leveled-Colored} holds on \( D' \).

**Proof.** This is a straightforward consequence: all rule applications only create coloured edges between subsequent levels. Moreover, the root is not affected by any rule application.

Q.E.D.

**Lemma 14 (Ancestor-Edges).** Given a valid DLDS \( D = \langle V, (E_D^i)_{i \in \{\lambda\} \cup O^p}, E_A, r, l, L, P \rangle \), and \( u \) and \( v \) in level \( j \), such that \( l(u) = l(v) \). Let \( R \) be any HC rule, such that, \( R \) applies in algorithmic position to \( u \) and \( v \) resulting in a DLDS \( D' \). Thus, condition \textit{Ancestor-Edges} holds on \( D' \).
Proof. This follows immediately from the fact that all rule application only creates ancestor edges from a node in level \( l_1 \) to a node in level \( l_2 \) and \( l_1 > l_2 \).

Q.E.D.

Lemma 15 (Ancestor-Backway-Information). Given a valid DLDS \( D = (V, (E_D^i)_{i \in I} \cup O, E_A, r, l, L, P) \), and \( u \) and \( v \) in level \( j \), such that \( l(u) = l(v) \). Let \( R \) be any HC rule, such that, \( R \) applies in algorithmic position to \( u \) and \( v \) resulting in a DLDS \( D' \). Thus, condition Ancestor-Backway-Information holds on \( D' \).

Proof. This is a straightforward consequence of all rule applications: (1) When it creates an ancestor-edge \( e \), it labels it with the relative address of source\((e)\) from target\((e)\) as the label provided by the component \( P \) of \( D \), and; (2) When it modifies the target or source of an ancestor edge, it updates the component accordingly \( P \) of that labels the modified edge.

Q.E.D.

We provide the following definition based on the analysis that each one of the listed rules does not preserve simplicity of A-edges.

Definition 30 (A-edge simplicity destroyer rule). The following rules in some application cases can destroy the simplicity of A-edges in the graph. \( R_v 2HH, R_v 2XH, R_v 2IH, R_v 2EH, R_v 2XH, R_v 2IH, R_v 2EH, R_v 3XH \).

We abbreviate A-edge simplicity destroyer rule as AESD-rule.

Lemma 16 (Simplicity and Ancestor-Simplicity). Given a valid DLDS \( D = (V, (E_D^i)_{i \in I} \cup O, E_A, r, l, L, P) \), and \( u \) and \( v \) in level \( j \), such that \( l(u) = l(v) \). Let \( R \) be any HC rule, such that, \( R \) is not AESD and it applies in algorithmic position to \( u \) and \( v \) resulting in a DLDS \( D' \). Conditions Simplicity and Ancestor-Simplicity holds on \( D' \).

Proof. Simplicity is a consequence of the applicability conditions for each type of rule. Note that the lefthand side of each compression rule figures out every deductive edge present in \( D \) in the algorithmic position determined by \( j, u \) and \( v \). On the right-hand side, do not place more than one deductive edge of the same colour between the same pair of nodes. This holds for every compression rule and ensures Simplicity in each application of the rules. A similar argument holds for Ancestor-Simplicity. We have to use lemma [11] however, to ensure that there is no ancestor edge with source below \( u \) or \( v \) and target in some node of the lefthand side. With this property provided by the lemma [11] we can ensure that the modifications and insertions of the ancestor edges in the righthand side of a compression rule application produce no more than one ancestor edge between the same pair of nodes.

Q.E.D.

We note that lemma [16] above does not hold, in general, if we relax the restriction that the applied HC-rule is not a AESD rule. MDE and AESD rules apply on top-nodes that are target of A-edges, moving the A-edges down to have other node as target or keeping it as is. In the case of a MDE-rule, the new target may be target of other A-edge. Thus, the inductive hypothesis when applying MDE rules cannot hold. Observe the example in section [2] figure [27] where many applications of MDE rules produce the same number of A-edges between the pair of nodes. Concerning the AESD rules that are not MDE, when two or more applications of AESD are used in sequence, the corresponding A-edges that are kept may have equal sources and, with the collapse, they will have the same target. This is enough to destroy A-simplicity. However, we can observe that in this sequential application,
the configuration in the second application of the AESD rule, the path that labels the A-edge, by the
P labelling function, is equal to the path that labels the A-edge that has the collapse node as target,
just the previously collapsed. Withe the purposes of using the A-edge as source of information for
reading correctly the resulting DLDS, we do not need more than one A-edge. Thus, we can remove all,
but only one of the A-edges. The final DLDS is A-simple. Moreover, we have the following lemma.
In figure [54] we have this situation, but we show both A-edges, instead of removing one of them. The
final DLDS in the example in section [2] shows also this phenomenon.

Lemma 17 (Unicity of A-edges labeling). Consider any sequential application of a pair of AESD
rules to an A-simple DLDS. After the applications of these rules the DLDS can be taken as a A-
simple graph.

Proof The proof is based on a discussion of one specific case. Consider that the sequence of
AESD rules Re2IH and Re3XH when applied to the configuration depicted by the leftmost graph
in figure [54] inside a DLDS. After the application of the mentioned rules, it produces the rightmost
graph. The two arrows convey the same information, that is, the path s2 that is the address of w from u.
Thus, we only need one of the A-edges after application of the rules. This is automatically
implemented by the fact that the set A-edges is a set, has no repetitions. There are ten AESD rules
and one hundred of pairs. They are analogous to the case we illustrated here. For the purposes of this
article the lemma is proved.

Q.E.D.

Look at the example used in the lemma above, i.e., figure [54]. Let w[5] be the source of the possibly
many A-edges, say v1, . . . , vn that have top-nodes in the same level of v and are premises of the same
conclusion, the sequence of AESD rule applications is as long as the number of top-nodes that are
target of A-edges. Moreover, since the source node w is shared by every A-edge, then the paths that
go from each v, i = 1, . . . , n and labels the many A-edges from each vi to w, carry the same logical
information. Thus, as in lemma above, we can remove all, but only one of them, ending up with an
A-simple graph, and hence a DLDS. The full formalization of lemma [17] is in progress and [CSH22]
reports what has already been done. The content of this paragraph can be seen as a proof for the
following corollary of lemma [17] above.

Corollary 18 (AESD rules preserve Flow). Consider any sequential application of a pair of AESD
rules to an A-simple DLDS. After the applications of these rules the function Flow on the resulted
Flow is the restriction of the original Flow function before the application of the sequence of AESD
rules.

Lemma 19 (Non-Nested-Ancestor-Edges preservation). Given a valid DLDS \( D = (V, (E_D^i)_{i \in \{\lambda\} \cup \Omega}, E_A, r, l, L, P) \),
and u and v in level j, such that l(u) = l(v). Let R be any HC rule, such that, R applies in algorithmic
position to u and v resulting in a DLDS \( D' \). Condition Non-Nested-Ancestor-Edges holds on \( D' \)

Proof. Let \( D = (V, (E_D^i)_{i \in \{\lambda\} \cup \Omega}, E_A, r, l, L, P) \) be a valid DLDS, j be a level in \( D \) and R a HC
rule in algorithmic position that collapses u,v \( \in V \). We prove that the resulting \( D' \) DLDS satisfies
Non-Nested-Ancestor-Edges by observing that lemma [11] ensures the property that any node w in
a mixed deductive path from u (v), below the level j that is source of an ancestor edge \( e \in E_A \),
source(\( e \)) = w, target(\( e \)) is u (v), or linked to u (v) by a deductive edge of color 0 or \( \lambda \). The
application of any HC rule either moves the target above or below one level or, adds a new edge in a

\[ \text{the right and lower bullet in 54} \]
mixed deductive path without sources of \( w \). This was already argued in the proof of lemma \[ \text{I} \] Hence, the modifications caused by the application of any Rule do not introduce nesting. The same observation holds for the AESD rules too. Specifically, for the MDE rules, since when they move downwards an A-edge, the new target takes part in the old path. The indicative hypothesis ensures that this path has no nesting.

Q.E.D.

Lemma 20 (CorrectRuleApp). Given a valid DLDS \( D = \langle V, (E^D_i)_{i \in \{\lambda, \land, \lor\}}, E_A, r, I, L, P \rangle \), and \( u \) and \( v \) in level \( j \), such that \( I(u) = I(v) \). Let \( R \) be any HC rule, such that, \( R \) applies in algorithmic position to \( u \) and \( v \) resulting in a DLDS \( D' \). Thus, condition CorrectRuleApp holds on \( D' \)

Proof. By hypothesis, \( D \) is valid. Thus, \( Flow(D, w)(\nu) \) is well-defined, for every \( w \in D \) and \( \nu \in Pre(w) \). Moreover, item [CorrectRuleApp] holds, that is, for every \( w \) and \( \nu \in Pre(w) \) both items in CorrectRuleApp holds. We have to prove that \( D' \) also satisfies these two items of CorrectRuleApp, for each possible application of \( R \) in \( u \) and \( v \) in \( D \), yielding \( D' \). Due to readability and article’s size concerns, we will not show all 28 cases detailed case-analysis proof to catch and agree with a validity of this mathematical result. At a higher-level of presentation, we show here a typical and complex enough case in detail. Let us consider that \( R \) is \( R_v \cdot \text{3XE} \) as it is shown in figure 55 below, and that \( w_u \) is the conclusion node which has \( u \) as a premiss, and analogously, for \( w_v \) regarded to \( v \). Thus, in this particular case, we have that \( Flow(D', w_v)(\nu) = (b_2, [0; k]; s_3) \), and we can verify that \( b_2 = L(v, w_v) \). \( Flow(D, w_u)(u) = (f_1 \lor f_2, [0; \ldots; i; s_1], (c_1 - I(p_0), [0; \ldots; j, s_2]) \}, considering that \( D \) is valid and hence, \( I(u) = I(p_0) \lor I(p_1) \) and \( (p_1, u) \in \gamma_i \), as well \( (p_1, u) \in \gamma_E \). The item CorrectRuleApp holds in \( D \), since \( (u, w_u) \in E^D_0 \) and \( L((u, w_u)) = \lambda \), for \( \text{card}(Flow(D, w_u)(u)) = 2 \), and then, due to subitem 1 of the second item of [CorrectRuleApp] holds. After the application of \( R_v \cdot \text{3XE} \), see the right-hand side of figure 55 we have the following analysis concerning the validity

![Figure 54: Applying rules Re2IH and Re3XH in sequence, lemma 17](image-url)
of $\mathcal{D}'$, remembering that $l(u) = l(v)$.

$$\text{Flow}(\mathcal{D}', w_u)(u) = \{(\bar{f}_1 \lor \bar{f}_2, [0;\ldots;i;3]), (\bar{c}_1 - l(\bar{p}_0), [0;\ldots;j;2]), (\bar{b}_1 \lor \bar{d}_1, [1;1;3])\}$$

What is justified by:

$$\text{Flow}(\mathcal{D}', w_u)(p_{1a}) = \{(l(\bar{p}_{1a}), [0;0;\ldots;i;1])\} \quad \text{and} \quad \text{Flow}(\mathcal{D}', w_u)(p_{1b}) = \{(l(\bar{p}_{1b}), [0;0;\ldots;i;1])\}$$

$$\text{Flow}(\mathcal{D}', w_u)(p_1) = \{(\bar{c}_1, [0;0;\ldots;j;2])\} \quad \text{and} \quad \text{Flow}(\mathcal{D}', w_u)(p_2) = \{(l(\bar{p}_2), [1;1;3])\} \quad \text{and} \quad \text{Flow}(\mathcal{D}', w_u)(p_3) = \{(l(\bar{p}_3), [1;1;3])\}$$

With, $\bar{f}_1 = l(\bar{p}_{1a})$, $\bar{f}_2 = l(\bar{p}_{1b})$, $\bar{b}_1 = l(\bar{p}_2)$ and $\bar{d}_1 = l(\bar{p}_3)$. Thus, the conditions on $\text{CorrectRuleApp}$ hold for $\mathcal{D}'$, observing the the nodes other then $w_v, u, w_u$ and their node premisses are not changed. So, the conditions stated by $\text{CorrectRuleApp}$ hold on them, since $\mathcal{D}$ satisfies it.

![Figure 55](image-url)

The other cases, concerning the other 28-1 rules, take part in the forthcoming formalization of this compressing algorithm and formal proof of the theorem in this article, see [CSH22]. We are working in a more formal presentation, with an integral and more detailed argument with a formalization in the LEAN ITP. [CSH22] shows initial steps in the formalization of what we have in this article/report.

Q.E.D.

Lemmas [16] is the only lemma that does not work for every HC-rule. Ancestor simplicity is not preserved, in general, by MDE rules. It is preserved by all other HC-rules. Thus, if we do not consider MDE rules, we have the following theorem [21]

---

---

---
Theorem 21. The set of compression rules that are not ASED preserves validity of any valid DLDS.

Proof. We show for each one of the rules presented in the previous section that if a DLDS D is valid, then so is the D after the application of the rule, by lemmas 12, 13, 14, 15, 16, 19, 20, 22. Observe that lemma 16 is the only in the list of lemmas that is restricted to rules that are not MDE.

Q.E.D.

A formal proof of the above lemma would be a very long case analysis, such that, each case has to check each of the items in the definition of validity of DLDS in definition 23. Due to the tedious repetition of cases, we postponed this formalization to a further article, conveyed with the help of an Interactive Theorem Prover. In [CSH22] it is report the initial step forward this formalization.

Look at the example at the beginning of this article, figures 28 and 28. Due to the order of HC-rules application, the compression algorithm applied all the (appliable) MUE rules before the first MDE rule application. This is a consequence of the fact that no MDE rule can be applied before all the other rules cannot be applied anymore. Remember that the order of application of the rules is bottom-up and right to left. The pattern for the application of any of the MDE rules appears in the DLDS only after some rule has been applied.

A MDE rule is only applied when the two nodes that will be collapsed by the application are both A-edge targets and top-nodes. The MUE rules repeatedly move the ancestor edges up until they do not apply anymore when only some MDE rule can be applied, i.e., we reach firstly a MUE+ -compressed DLDS. We can state the following theorem that uses the definition below.

Definition 31 (Partially MUE+ -compressed DGTD). A partially MUE+ -compressed DGTD is a partially compressed DGTD obtained by the sole application of MUE HC-compression rules to a DGTD.

Theorem 22. Let \( D = \langle V, (E_D)_{i \in \{\lambda\} \cup \Omega}, E_A, r, l, L, P \rangle \) be a valid partially MUE+ -compressed DGTD. Consider the HC compression algorithm, restricted only to MUE rules, applied to \( D \). The algorithm stops after a finite number of steps and obtains a valid DLDS \( D' \) that has no level with a pair of top-nodes labelled with the same formula, except possibly for top-nodes occurring in the same level and labelled with the same formula.

Proof. Since DGTDs are DLDS too, by theorem 21 \( D' \) is a valid DLDS. Each time, a MUE rule applies to a valid DLDS, it yields a DLDS with one pair of nodes labelled with the same formula nodes less. Type-0 rules create A-edges, Type-I rules creates A-edges too. Type-II rules, but the MDE rules, move the A-edges up along the derivation. The combination of these MUE rules, plus the Type-III rules, for example \( R_v 3XE, R_e 3XE, R_v 3XI, R_e 3XI \); collapse all nodes labelled with the same formula in levels lower than the top-nodes connected to them. At this stage no MUE rule can be applied, for there is no configuration that matches the left handside of these rules. The only nodes labelled with the same formula in the same level are the top-nodes. Thus, \( D' \) is a valid MUE+ -compressed DLDS.

Q.E.D.

Corollary 23. If \( D \) is a MUE+ -compressed DLDS the only targets of A-edges in \( D \) are the top-formulas.

Finally, using lemma 17 and corollary 18 we note that we can obtain a fully-compressed DLDS by the application of the MDE rules from bottom-up and left-to-right. MDE rules collapse the top-nodes labelled with the same formula that occur in the same level. Thus, we have the followin theorem.
Theorem 24. If algorithm\[\text{extension}\] extended with a second round of MDE rules applications is applied to a valid DLDS then it eventually halts providing a DLDS that has no level with two nodes labeled with a same formula.

The root is labelled with the conclusion of the valid DLDS. To have the information about the dependency set from which this conclusion depends, we have to consider the effect of the last rule on dependency sets represented by labels on the deductive edges incoming in \( r \). Remember that the last rule can be a \( \supset \)-Intro or a \( \supset \)-Elim. We add a new node linked to the root \( r \) with a (new) distinguished edge with the sole purpose of having it labelled with the final dependency set. We call this a grounded DLDS.

Definition 32. Let \( \mathcal{D} = \langle V, (E_D^i)_{i \in \mathcal{O}}, E_A, r, l, L, P \rangle \) be a valid DLDS. Let \( \mathcal{D}' \) extend \( \mathcal{D} \) by adding a new node \( g \), such that, \( l(g) = \frac{1}{n} \), a new deductive edge of color 0 \( \langle r, g \rangle \), such that \( L(\langle r, g \rangle) = \vec{b}(S) \), where:

1. If \( l(r) = \alpha_1 \supset \alpha_2 \) and \( \langle v, r \rangle \) is its only incoming edge, a deductive edge of color 0, then \( r \) is the conclusion of a \( \supset \)-intro and \( S = \text{Set}(L(\langle v, r \rangle)) - \{\alpha_1\} \); and

2. If \( r \) is the conclusion of a \( \supset \)-Elim and \( \langle v_1, r \rangle \) and \( \langle v_2, r \rangle \) are its only incoming edges then \( S = \text{Set}(L(\langle v_1, r \rangle)) \cup \text{Set}(L(\langle v_2, r \rangle)) \)

We call \( \mathcal{D}' \) a grounded DLDS.

Note that only valid DLDS can be grounded, by the force/will of the definition. The root of a grounded DLDS, is, again, by the the force of definition, \( r \), where \( \langle r, g \rangle \in E_D^0 \) and \( l(g) = \frac{1}{n} \). The node \( g \) is the support for the ground only. We have the following corollary. We say that a DLDS is compressed iff it has no collapsible pair of node in any level.

Corollary 25. Let \( \alpha \) be any tautology in \( M_\supset \). There is a compressed grounded DLDS \( \mathcal{D} = \langle V, (E_D^i)_{i \in \mathcal{O}}, E_A, r, l, L, P \rangle \), such that \( l(r) = \alpha \), \( \Gamma \) is the set of sub-formulas of \( \alpha \) and \( L(\langle r, \frac{1}{n} \rangle) = \vec{b}(\emptyset) \).

Proof. If \( \alpha \) is a \( M_\supset \) tautology then there is a Natural Deduction proof of it. By the normalization of \( M_\supset \) there is a normal ND proof. This proof has only occurrence of subformulas of \( \alpha \). By lemma\[\text{extension}\] there is a greedy ND derivation of \( \alpha \) having also only sub-formulas of \( \alpha \) occurrences. Proposition\[\text{extension}\] and corollary\[\text{extension}\] ensure the existence of a DGTD \( T \) that is a proof of \( \alpha \). Finally, from proposition\[\text{extension}\] we can easily see that \( \text{Dag}(T) \) is a valid DLDS and its extension to a grounded DLDS has \( L(\langle r, \frac{1}{n} \rangle) = \vec{b}(\emptyset) \). Thus, by theorem\[\text{extension}\] and theorem\[\text{extension}\] we have a compressed and grounded DLDS with the conclusion depending of no assumption, i.e., it is a proof of \( l(r) \).

Q.E.D.

Finally, we adjust the compression algorithm to obtain fully-compressed DLDS from valid DGTD.
Algorithm 3 Horizontal Compression

Precondition: A tree-like greedy valid derivation $D$

Ensure: The valid DLDS that is $D$ fully-compressed

1: for $lev$ from 1 to $h(D)$ do
2: for $u$ and $v$ at $lev$ do
3: $HCom_{MUE}(u,v)$
4: end for
5: end for
6: for $lev$ from 1 to $h(D)$ do
7: for $u$ and $v$ at $lev$ do
8: $HCom_{MDE}(u,v)$
9: end for
10: end for

7 On the size of compressed Natural Deduction proofs

We start by a simple consequence of the validity of any DLDS.

Lemma 26. Let $D = \langle V, (E^i_D)_{i \in O}, E_A, r, l, L, P \rangle$ be a compressed grounded DLDS, obtained by the application of the Horizontal Compression algorithm 3 to a valid DGTD. The underlying graph of $D$ with colors in $O_T$ and $E_A$ is a simple graph.

Proof. This is a trivial consequence of validity definition, since the fifth and sixth conditions on the validity is the simplicity of the full colored subgraph of $D$, with colors in $O_T$ and $E_A$, respectively.

Q.E.D.

In lemma 26 above, we consider only the full subgraph coloured with ordinals and the ancestor edges ($E_A$). However, some rules, such as rule $R_{39}$, create coloured edges labelled with the colour $\bar{\lambda}$, although no more than one by pair of vertexes. Thus, we should note that rules $R_{39}$, $R_{54}$, $R_{51}$, $R_{??}$, $R_{47}$ do not create more than one edge labeled with $\bar{\lambda}$ between any pair of nodes. As already said, We consider $\bar{\lambda}$ as a colour. Besides that, the rules mentioned above are rules that collapse edges in a way that anytime a pair of edges is collapsed, the algorithm generates at most one $\bar{\lambda}$ coloured edge. The collapse of edges is a consequence of the collapse of their respective source and targets in a two-step action. Thus, the previous lemma extends to the following proposition 27, where the only addition to the previous lemma 26 is to consider $\bar{\lambda}$ as a colour.

Proposition 27. Let $D = \langle V, (E^i_D)_{i \in O}, E_A, r, l, L, P \rangle$ be a compressed grounded DLDS, obtained by the application of the Horizontal Compression algorithm 3 to a valid DGTD. The underlying graph of $D$ with colors in $O_T$ and $E_A$ is a simple graph.

Proof. This is a trivial consequence of the validity of the DGTD, formalized in the lemmas above, and the fact that we can only apply the rule in figure 19 when all the node in its graphical representation are pairwisely different. The focus is on the two bullets being different.

Q.E.D.

Lemma 28. Let $D = \langle V, (E^i_D)_{i \in O}, E_A, r, l, L, P \rangle$ be a compressed grounded DLDS, obtained by the application of the Horizontal Compression algorithm 3 to a valid DGTD, with height $h$. For each $(u, v) \in E_A$, $\text{len}(P((u, v)))$ if defined is smaller than $h$. 
Corollary 31. E.H.Haeusler, J.F.C. Barros Jr. and R.C.M. Brasil Filho

Proof. In a valid DLDS, each ancestor edge \( (u, w) \in E_A \) has its source \( u \) in a level strictly smaller than \( w \)'s level, i.e., the target. Since \( P((u, w)) \), according the validity of \( \mathcal{D} \) is the relative address of \( u \) from \( w \), which goes downwards always, then the size of \( P((u, w)) \) cannot be bigger than the difference between the levels of \( w \) and \( u \). Thus \( \text{len}(P((u, w))) \leq \text{h} \). Observe that no compression rule increases the height of the DLDS.

Q.E.D.

We state the following lemma without proof, since it is an easy consequence of the definition of \( l \).

Lemma 29. Let \( \mathcal{D} = \langle V, (E_D^i)_{i \in \mathcal{O}}, E_A, r, l, L, P \rangle \) be a compressed grounded DLDS, obtained by the application of the Horizontal Compression algorithm \( 3 \) to a valid DGTD. For each \( (u, w) \in E_D^i \), the size of \( l((u, w)) \) the length of the bitstring that is size of \( \Gamma \) plus one.

Definition 33. Size of a DLDS Let \( \mathcal{D} \) be a grounded DLDS of \( \alpha \) from \( \Gamma \). The size of \( \mathcal{D} \) is the length of the string obtained by the juxtaposition of the the strings (words) in the alphabet of ordinals derived from \( \Gamma \) plus \( \lambda \), and the punctuation marks: \( \{, \}, \langle \rangle \) representing each component of \( \mathcal{D} \).

We observe that in fact, the punctuation marks do not need to be taken into account in the complexity analysis. Thus, the followin results do not take them into account.

Proposition 30 (Upper-bound on simple DLDS). Let \( \mathcal{D} \) be a DLDS with height \( h \), with \( m \) as the size of the set of node labels, such that, has a simple graph as the underlying colored graph. The size of \( \mathcal{D} \) has upper-bound \( O(h \times m^4) \).

Proof. Recall that \( \mathcal{D} = \langle V, (E_D^i)_{i \in \mathcal{O}}, E_A, r, l, L, P \rangle \), \( \text{size}(\Gamma) = m \); \( \text{size}(V) \leq h \times m \); considering the labelling of vertexes \( \text{size}(\text{labeled}(V)) \leq h \times m^2 \); i.e., lemma 29. Considering the colors of the the deduction edges, for each color \( i = 0, m \), \( \text{size}(E_D^i) \leq \text{size}(V) \); hence \( \text{size}(\text{labeled}(E_D^i)) \leq \text{size}(V)^2 \times m = (h^2 \times m^2) \times m = h^2 \times m^3 \). Considering all colors we have \( \text{size}(E_A) \leq \text{size}(V) \times \text{size}(V) \times h \leq (h \times m)^2 \times h = (h \times m)^3 \). Finally we have that \( \text{size}(\mathcal{D}) \leq O(h \times m^4) \)

Q.E.D.

Corollary 31 (Upper-bound on compressed grounded DLDS). Let \( \Pi \) be a proof of \( \alpha \) in \( M_2 \) with height \( h \). Let \( m \) be the number of formulas of occurring in \( \Pi \). Then there is a compressed grounded DLDS \( \mathcal{D} = \langle V, (E_D^i)_{i \in \mathcal{O}}, E_A, r, l, L, P \rangle \), such that \( \text{I}(r) = \alpha \), \( L(\langle r, 1 \rangle) = b(\emptyset) \) and \( \text{size}(\mathcal{D}) \leq O(h \times m^4) \).

Proof. From corollary 25 we have the existence of \( \mathcal{D} \) and from proposition 30 we have the upperbound.

Q.E.D.

8 On the experimental compression rate of the algorithm

In 2019, one of the authors implemented horizontal compression (HC) using Python2.7.15 and the GRAPHVIZ library to store and manipulate graphs. In his master dissertation, he implemented Natural Deduction proofs and DLDS in GRAPHVIZ. Some classes of formulas with huge proofs were submitted to the HC algorithm. We developed a program to generate purely implicational Natural
Deduction proofs of non-hamiltonicity of graphs to have specimens submitted to the HC implementation. Fibonacci based Natural Deduction purely implicational formulas with exponential size were compressed by the HC algorithm and compared to Huffman compression algorithm, one of the benchmarks in compression of strings. In 2019 we could not find a robust compression algorithm for proofs. Thus, we report here the comparison with a string-based algorithm in the case of Huffman compression. We show a qualitative comparison of a non-hamiltonicity proof of size almost $3^3$ formulas with its HC compressed DLDS graph. We can see in figures 56 and 58. We know that a graph with three nodes is quite small. Its proof of non-hamiltonicity is a Natural Deduction proof with almost $3^3$ formulas. Anything bigger cannot be shown for the sake of qualitative analysis. The set of HC-rules in 2019 was not the same that is shown here. The current set is an optimzed and more adequate set of rules, mainly to have a simpler proof of soundness preservation.

Figure 56: Natural Deduction brute force proof of non-hamiltonicity of the graph in figure 57.

Figure 57: Graph G3
In the compressed proof shown in figure 57, we can see only three collapse nodes, the red ones, which generated some ancestor edges, and the blue ones. In general, as big the proof is, as redundant it is, see [Hae22] and the HC algorithm works better. Figure 59 shows the comparison of the compression rate of the Huffman with the horizontal compression (HC) rate.
On the horizontal compression

Figure 59: Compression comparison between Huffman and HC compression for big tautologies
We can see in figure 59 how HC is better than Huffman when the proofs are bigger, and more redundant henceforth. In figure 60 we can effectively compare the compression rate between both algorithms.

![Graph comparing compression rate between Huffman and HC compression for big tautologies](image)

Figure 60: Compression rate comparison between Huffman and HC compression for big tautologies

Finally, in figure 61 we compare the time used by both algorithms for compressing the proofs of the class Fibonacci(n) that was used in this basic quantitative study.
66

On the horizontal compression

Figure 61: Comparison between Huffman and HC compression time for big tautologies of the class Fibonacci(n)

9 Conclusion and further research

We think we succeeded to a high degree in almost all of the contributions listed in the second paragraph of the introduction. One of the primary purposes of this report is to provide a comprehensive technical presentation of the Horizontal Compression method to compact propositional proofs in $M_\subseteq$. Using this report, other researchers or we can use a prover assistant to formalize the proof of soundness preservation, termination and even the polynomial upper bound. We think that maybe the last one because having traditional estimative techniques on data structures might not need formal proof. The experimental part lacks a comparison with Ziv-Lempel method. It will follow soon. We advocate that the compression of proofs and their corresponding efficient verification is a research goal per se. It can be taken independently from our theoretical proofs already published. Concerning them, we currently work to improve our arguments to have shorter and more intuitive ones. The fact that nowadays we are concerned with assisted proofs on the compression method shown here is already a research goal per se.

Acknowledgments This work arose in the context of term- and proof-compression research supported by the ANR/DFG projects HYPOTHESES and BEYOND LOGIC [DFG grants 275/16-1, 16-2, 17-1] and the CNPq project Proofs: Structure, Transformations and Semantics [grant 402429/2012-5]. The first author, Haehsler, wants to thank very much prof. Lew Gordeev, his co-author and co-responsible for many of the ideas in a first version of HC compression method on Natural Deduction proofs. Nowadays we intend to use the HC algorithm in a alternative proof, more concrete, way to prove what we proved in [GH20, GH22c, GH22b] and that we presented in Logic Coloquium 2021, see abstract in [GH22a]. We would like to thank L. C. Pereira and all colleagues in PUC-Rio for their contribution as well as P. Schroeder-Heister (EKUT) and M. R. F. Benevides (UFRJ) for their support of these projects. Special thanks go to S. Buss and R. Dyckhoff for their valuable
comments during the preparation of [GH19]. We thank Robinson Callou for accepting the challenge to formalize the results presented in this article in $L∀∃N$. One of the authors is grateful to Leonardo de Moura for the constant motivation to have the formalization of the HC compression in some ITP. There is a huge group of people that should be listed in the acknowledgements. There is no list of names in this part of the article to avoid any mistaken omission.

References

[CSH22] Robinson Callou, Jefferson Santos, and E. H. Haeusler. Towards a proof in lean about the horizontal compression of dag-like derivations in minimal purely implicational logic. In Pre-Proceedings of LSFA2022, 2022.

[GH19] L. Gordeev and E. H. Haeusler. Proof compression and np versus pspace. Studia Logica, 107(1):55–83, 2019.

[GH20] L. Gordeev and E. H. Haeusler. Proof compression and np versus pspace ii. Bulletin of the Section of Logic, 49(3):213–230, 2020.

[GH22a] L. Gordeev and E. H. Haeusler. On proof theory in computational complexity. In Report on the Logic Coloquium 2021, 2022. The Logic Coloquium Proceedings of Abstracts is published as an article in [?].

[GH22b] L. Gordeev and E. H. Haeusler. On proof theory in computational complexity: overview, 2022.

[GH22c] Lew Gordeev and Edward Hermann Haeusler. Proof Compression and NP Versus PSPACE II: Addendum. Bulletin of the Section of Logic, page 9 pp., Jan. 2022.

[Hae22] Edward Hermann Haeusler. Exponentially huge natural deduction proofs are redundant: Preliminary results on $M\supset$. FLAP, 9(1):287–326, 2022.

[Hud93] J. Hudelmaier. An $o(n\log n)$-space decision procedure for intuitionistic propositional logic. Journal of Logic and Computation, 3:1–13, 1993.

[Pra65] D. Prawitz. Natural deduction: A proof-theoretical study. Almqvist & Wiksell, 1965.

A polynomial time algorithm to verify validity of DLDS

In this section we present a (polynomial in time) algorithm to verify whether a DLDS is valid or not. In fact, the algorithm verifies whether its input, a DLDS, is a (partial sometimes) result of the compressing method shown in [5] when applied, even partially, to any valid DLDS. As a greedy tree-like derivation is a trivially valid DLDS by the soundness of Natural Deduction for $M\supset$, the correctness of the algorithm follows. However, any discussion on correctness is postponed to section 6. The algorithm below computes the function Flow from top downwards. Returning an error, if the Flow cannot be defined in some node of the DLDS or returns the dependency set of the root, that should be empty in order to the DLDS be valid.
Algorithm 4 Verification that a DLDS $\mathcal{D}$ is a valid derivation from $\Delta \subseteq \Gamma$

**Precondition:** $\mathcal{D} = \langle V, (E_D)_{i \in O^1}, E_A, r, I, L, P \rangle$ and $\Delta \subseteq \Gamma$

**Precondition:** $\langle V, \bigcup_{i \in O^0} E^i_D \rangle$ with root $r$

Uses $S_1, S_2$ : Flow

1. $S_1 \leftarrow \emptyset$
2. $S_2 \leftarrow \emptyset$
3. $\text{AddConclusionDummyEdge}(\langle r, \text{conc} \rangle; \mathcal{D}; L(\langle r, \text{conc} \rangle) = \overline{\Delta})$
4. $\text{CheckRootedDagLevel}(V, (E_D)_{i \in O^1}, E_A, r)$
5. $\text{Lev} \leftarrow \text{height}(\mathcal{D})$
6. for $\text{ln} = \text{Lev}$ downto $0$
7. for $p$ at level $\text{ln}$ do
8. for $c \in \text{colors}$ do
9. try $\text{Premisses} \leftarrow \text{Premiss}(S_1, p, c)$ end try
10. catch (exception InvalidPremisses) end catch
11. if $\text{Hyp}(p)$ then
12. if $(c==0)$ then
13. if $\text{Premisses} == \emptyset$ then
14. $S_2(0) \leftarrow (I(p), e)$
15. else
16. thrown InvalidException(“Vertex” $p$ “as a regular (not related to $\lambda$-Edges) hypothesis cannot have incoming edges or premisses ”)
17. else
18. if $\text{Premisses} == \emptyset$ then
19. if $\exists g \in E_A(\text{target}(g) = p)$ then
20. for $g \in \{ g : \text{target}(g) = p \}$ do
21. try $S(\text{hd}(P(g)) \leftarrow (I(p), \text{tail}(P(g)))$ end try catch (exception Invalidcolor / path) end catch
22. end for
23. else
24. $S_2(c) \leftarrow \emptyset$
25. else
26. $S_2(c) \leftarrow \text{Update}(\text{Premiss}, p, c, S_1)$
27. else
28. if $(c==0)$ then
29. if $\text{Premisses} == \emptyset$ then
30. thrown InvalidException(“Hypothesis should not have premisses”)
31. else
32. $S_2(c) \leftarrow \text{Update}(\text{Premiss}, p, c, S_1)$
33. else
34. if $\text{Premisses} == \emptyset$ then
35. $S_2(c) \leftarrow \emptyset$
36. else
37. $S_2(c) \leftarrow \text{Update}(\text{Premiss}, p, c, S_1)$
38. end for
Algorithm 4 Verification that a DLDS $\mathcal{D}$ is a valid derivation

39:   for $f \in \text{out}(p)$ do
40:      if $\text{color}(f) == 0$ then
41:         $S_1(f)(0) \leftarrow S_2(0)$
42:      elseif $\text{color}(f) > 0$ then
43:         if $S_1(f)(\text{color}(f)) == \emptyset$ then
44:            $S_1(f)(\text{color}(f)) \leftarrow S_2(\text{color}(f))$
45:         else
46:            thrown InvalidException(“$S_1$ already defined, should not pass by initial check of decorated rooted
dag”)
47:      elseif $\text{color}(f) == \lambda \land \text{out}(p) == 1$ then HERE HERE
48:         for $c' \in \text{colors}$ do
49:            $S_1(f)(c') \leftarrow S_2(c')$
50:         end for
51:      else
52:         thrown InvalidException(“undefined color of edge or more than one $\lambda$-edge going out a vertex”)
53:      end for
54:      $S_2 \leftarrow \emptyset$
55:   end for
56:   Checking whether the assumption set for the conclusion of the derivation contains only the hypothesis (non-
discharged assumptions)
57:   if $S_1(\langle r, \text{conc} \rangle)(0) == (\Delta, \epsilon) \land \forall c(S_1(\langle r, \text{conc} \rangle)(c) == \emptyset)$ then
58:      print(“$\mathcal{D}$ is a valid derivation of $l(r)$ from $\Delta$”)
59:   else
60:      print(“$\mathcal{D}$ is NOT a valid derivation of $l(r)$ from $\Delta$”)

The following algorithm is used as a procedural unit in algorithm ?? As a matter of better readability we call DedEdges the set $(\bigcup_{i \in \text{colors}} E^i_D)$, i.e., the set of deductive edges of any color, but $\lambda$. 
Algorithm 5 Retrieves premisses related to color $c$ of a rule that has node $p$ as conclusion

Premiss$(S,p,c)$

Precondition: $D = (V,(E_D)_{i \in O_0},E_A,r,I,L,P)$ Global

Precondition: $S : Flow, p \in V$ and $c \in \text{colors}$

Ensure: A mapping $F : \text{colors} \setminus \lambda \rightarrow (\text{DedEdges})^2 \cup \text{DedEdges}$

1: if $c > 0$ then
2: if $\text{Suitable}\{f : (S(f)(c) \neq \bot) \land \text{target}(f) = p\}$ then
3: $\text{Return}(\langle c,\{f : (S(f)(c) \neq \bot) \land \text{target}(f) = p\}\rangle)$
4: else
5: thrown $\text{InvalidException}$(“incoming edges are not suitable premisses”)
6: elseif ($c == 0$) then
7: if $\text{Hyp}(p)$ then
8: if $\{f : (S(f)(c) \neq \bot) \land \text{target}(f) = p\} == \emptyset$ then
9: $\text{Return}(\emptyset)$
10: else
11: thrown $\text{InvalidException}$(“A hypothesis has no premisses of color 0”)
12: else
13: if $\{f : (S(f)(c) \neq \bot) \land \text{target}(f) = p\} == \emptyset$ then
14: thrown $\text{InvalidException}$(“Only hypothesis have empty set of premisses”)
15: else
16: $\text{Return}(\langle c,\{f : (S(f)(c) \neq \bot) \land \text{target}(f) = p\}\rangle)$
17: elseif $c == \lambda$ then
18: $F \leftarrow \emptyset$
19: for $co \in \text{colors} - \{\lambda\}$ do
20: $F(co) \leftarrow \{f : (S(f)(co) \neq \bot) \land \text{target}(f) = p\}$
21: end for
22: $\text{Return}(F)$
23: else
24: thrown $\text{InvalidException}$()
The following algorithm is used as a functional unit in algorithm [22] too. It returns the partial Flow associated to the color $c$, using the premisses $Ps$ to update the Flow $S$ resulting in the partial Flow. Verification of conformity (validity) of $Ps$, $S$ and the DLDS in the context (global data) is also performed.

**Algorithm 6** Given premisses $Ps$, a color $c$ and a Flow $S$ returning the partial flow defined on $c$ obtained by updating the effect of the associated inference from the given data, including the case of $\lambda$-edge.

```
UpdatePartialFlow(Ps : Premisses, p : node, c : color, S : Flow)
Precondition: $D = (V, (E_D)_i \in O^0, E_A, r, I, L, P) \text{ Global}$
Precondition: $S : Flows, p \in V$ and $c \in colors, Ps \in Set(DedEdges)$
Ensure: A pair $F : B(O_S) \times (O^0_1)^*$
1: if $\exists! x((source(x) = p) \land x \in E_D \land c \neq \lambda)$ then
2: $g \leftarrow ix((source(x) = p) \land x \in E_D \land c \neq \lambda)$
3: if $\text{card}(Ps) == 2$ then
4: if $Is - \supset -E(Ps, g, S)$ then
5: $F(c) \leftarrow (\pi_1(S(\text{minor}(Ps))(c)) \oplus \pi_1(S(\text{major}(Ps))), \text{tail}(\pi_2(S(\text{minor}(Ps))(c))))$
6: else throw InvalidException (“Wrong Application of $\supset$-E rule. Cannot Update Flow ”)
7: elseif $\text{card}(Ps) == 1$ then
8: if $Is - \supset -I(Ps, g, S)$ then
9: $F \leftarrow (\pi_1(S(\text{el}(Ps))(c)) - \text{Antecedent}(I(p)), \text{tail}(\pi_2(S(\text{el}(Ps))(c))))$
10: else throw InvalidException (“Wrong Application of $\supset$-I rule. Cannot Update Flow ”)
11: else throw InvalidException (“Wrong Application of rule. Cannot Update Flow ”)
12: elseif $\exists! x((source(x) = p) \land x \in E_D)$ then
13: $g \leftarrow ix((source(x) = p) \land x \in E_D)$
14: if $\text{card}(Prem) == 2$ then
15: if $Is - \supset -E - \lambda(Prem, g, S)$ then
16: $F \leftarrow (\pi_1(S(\text{minor}(Prem))(c)) \oplus \pi_1(S(\text{major}(Prem))), \text{tail}(\pi_2(S(\text{minor}(Prem))(c))))$
17: else throw InvalidException (“Wrong Application of $\supset$-E rule. Cannot Update Flow ”)
18: elseif $\text{card}(Prem) == 1$ then
19: if $Is - \supset -I - \lambda(Prem, g, S)$ then
20: $F \leftarrow (\pi_1(S(\text{el}(Prem))(c)) - \text{Antecedent}(I(p)), \text{tail}(\pi_2(S(\text{el}(Prem))(c))))$
21: else throw InvalidException (“Wrong Application of $\supset$-I rule. Cannot Update Flow ”)
22: else throw InvalidException (“Wrong Application of rule. Cannot Update Flow ”)
23: else throw InvalidException (“Wrong Application of rule. Cannot Update Flow ”)
```

**Observation.** The following definitions are used in algorithm [22]
• $\text{el}(\{e\}) = e$
• $\text{Is} - \text{Intro}(\text{Ps}, g, S) : \text{EqStrings}(l(\text{source}(\text{minor}(\text{Ps}))) \supset l(\text{source}(g)), l(\text{source}(\text{major}(\text{Ps})))) \land (\pi_1(S(\text{minor}(\text{Ps}))(co)) \supset \pi_1(S(\text{major}(\text{Ps}))(co)) \equiv L(g)) \land \pi_2(S(\text{major}(\text{Ps}))(co)) = \pi_2(S(\text{minor}(\text{Ps}))(co)))$
• $\text{Is} - \text{Intro} - \lambda(\text{Ps}, g, S) : \text{EqStrings}(l(\text{source}(\text{minor}(\text{Ps}))) \supset l(\text{source}(g)), l(\text{source}(\text{major}(\text{Ps})))) \land (\pi_2(S(\text{major}(\text{Ps}))(co)) = \pi_2(S(\text{minor}(\text{Ps}))(co)))$
• $\text{Is} - \text{Intro} - I(\text{Ps}, g, S) : \text{EqStrings}(l(\text{source}(\text{el}(\text{Ps}))), \text{Sucedent}(l(\text{source}(g)))) \land (\pi_1(S(\text{el}(\text{Ps}))(co)) - \text{Antecedent}(l(\text{source}(g)))) = L(g)$
• $\text{Is} - \text{Intro} - I(\text{Ps}, g, S) : \text{EqStrings}(l(\text{source}(\text{el}(\text{Ps}))), \text{Sucedent}(l(\text{source}(g))))$

B  An example using the function $\text{Flow}$

Consider the portion of the (compressed) DLD $D$ shown in figure 62. Note that the vertical positions of the nodes may have nothing to do with the usual downwards way of reading N.D. derivations. For example, the node labeled with “$X2v1 \rightarrow q$”, in the highest vertical position in the figure, is the conclusion of an $\supset$-Intro rule that has the node labeled with $q$ as premise. Observe that $q$ appears in a position below its conclusion. To illustrate the use of the function $\text{Flow}$, definition /refdef:Flow, we consider the node $w$, labeled with $ORX3 \rightarrow q$. We advice the reader that we only illustrate the definition of the function $\text{Flow}$ in this appendix. The verification depends on the labeling function $L$ that is not represented in the picture below due to a clearer visualization of the DLD. The computer program that generates this example produces a messy graph if we show the labeling function $L$.

By definition, we have the following values for $\text{Flow}(D, w)$, considering that the name of the nodes are displayed near each one of them in figura 62. With the sake of showing a cleaner explanation on this example, instead of using bitstrings for representing the dependency sets we use the informal and shorter set representation.

$$
\text{Flow}(D, w)(u_1) = \{(X2v1, 00002)\} \\
\text{Flow}(D, w)(u_2) = \{(X2v1 \supset (X3v1 \supset q)), 00002\} \\
\text{Flow}(D, w)(u_6) = \{(X1v1 \supset (X3v1 \supset q)), 00001\} \\
\text{Flow}(D, w)(u_3) = \{(X1v1, 00001)\} \\
\text{Flow}(D, w)(u_3) = \{(X1v1, X1v1 \supset (X3v1 \supset q)), 00002\}, \{(X2v1, X2v1 \supset (X3v1 \supset q)), 00002\}
$$

For each node $u \in \text{Pre}(w)$, each $(\vec{b}, p) \in \text{Flow}(D, w)(u)$ of type $(O^*_1)^* \times B(O_S)$ is such that, $\vec{b}$ is or represents the dependency set that should label, using $L$, the unique deductive edge $\langle u, v \rangle$ of color $\text{head}(p)$ in $D$. The function $\text{Flow}$ recursively calculates this dependency set along the set of nodes that are in all deductive paths from the top nodes reaching to $w$. For example, in $D$, $u_1$ is a top node labeled with $X2v1$ and is target of an ancestor edge labeled with path 00002. This label, in a valid DLD, is the relative address of the source of the ancestor edge, the node just above left $u_1$ in figure 62 and labeled with $X2v1 \supset q$. Thus, the pair $\langle \{X2v1\}, 00002\rangle$ carries the information that the edge $\langle u_1, u_3 \rangle$ of color $0 = \text{head}(00002)$ is or should be labeled by $L$ with the bitstring that represents $\{X2v1\}$. An important observation is that when the DLD is a tree-like deduction, $\text{Flow}(D, w)(v) = \{(\vec{b}, \emptyset)\}$ for every $w$ and $v$ and $\vec{b}$ the usual dependency set for tree-like deductions. However, when we have already
collapsed nodes, as it is the case of \( u_3 \), we have to create a consistent flow of information. Note that \( \text{Flow}(D, w)(u_3) \), above, is not a singleton, it has two elements, they are respectively related to each derivation that share the node \( u_3 \) after the collapse. The pair \( \{\{X_1v_1, X_1v_1 \supset (X_3v_1 \supset q)\}\},00001 \) has to do with the subderivation that has \( u_1 \) and \( u_2 \) as premises and \( u_3 \) as conclusion, and relative address going to the formula \( X_2v_1 \supset q \), while the pair \( \{\{X_2v_1, X_2v_1 \supset (X_3v_1 \supset q)\}\},00002 \) goes to \( X_2v_3 \supset q \) and has premises in \( u_5 \) and \( u_6 \) and shared conclusion \( u_3 \). The set \( \text{Flow}(D, w)(u_3) \) is, hence, a two-element set. The number of elements of \( \text{Flow}(D, w)(v) \) reflects how many different ways of deducting \( v \in \text{Pre}(w) \) there are in \( D \).

**About the label \( \lambda \).** We noted, in the above paragraph, that \( \text{Flow}(D, w)(u_3) \) has two elements and hence is related with two different dependency sets, one is \( \{X_1v_1, X_1v_1 \supset (X_3v_1 \supset q)\} \) and the other is \( \{X_2v_1, X_2v_1 \supset (X_3v_1 \supset q)\} \). In this case, the DLDS cannot have both as labels assigned by \( L \) to the edge \( \langle u_3, u_4 \rangle \), we use the label \( \bar{\lambda} \) to indicate that the dependency set should be considered dynamically to be used downwards in a verification of validity. As a consequence, the DLDS validity conditions that involve the function \( \text{Flow} \), item \([\text{CorrectRuleApp}]\) in definition \([23]\) has some validity verification cases related on the \( \lambda \). We have to observe that the label \( \lambda[0] \) that appears in figure \([62]\) labeling some edges is used as an aggregation of the labels \( \lambda \) and the 0. This is due only to a better visualization and has not to do with any technical or any other deeper meaning. The following values of \( \text{Flow} \) show the results in \( \text{Pre}(w) \). Observe that the local validity of the rule, i.e., whether the node represents a valid \( \supset \) introduction or elimination rule is verified by the \( \text{Flow} \) function. In the case that the rule is not correctly applied the function \( \text{Flow} \) is not well-defined.

\[
\begin{align*}
\text{Flow}(D, w)(d_{10}) &= \{\{X_2v_3 \supset (X_3v_3 \supset q)\},00001\} \\
\text{Flow}(D, w)(d_{11}) &= \{\{X_2v_3\},00001\} \\
\text{Flow}(D, w)(d_9) &= \{\{X_1v_3 \supset (X_3v_3 \supset q)\},00002\} \\
\text{Flow}(D, w)(d_7) &= \{\{X_1v_3\},\emptyset\} \\
\text{Flow}(D, w)(d_{8}) &= \{\{X_1v_3, X_1v_3 \supset (X_3v_3 \supset q)\},00001\},\{\{X_2v_3, X_2v_3 \supset (X_3v_3 \supset q)\},00001\}\} \\
\text{Flow}(D, w)(d_{12}) &= \{\{X_3v_3 \supset q\} \supset ((X_3v_1 \supset q) \supset ((X_3v_2 \supset q) \supset ((\text{OR}X_3 \supset q))))\},\emptyset\} \\
\text{Flow}(D, w)(u_8) &= \{\{X_1v_3, X_1v_3 \supset (X_3v_3 \supset q)\}, \\
& \quad (X_3v_3 \supset q) \supset ((X_3v_1 \supset q) \supset ((X_3v_2 \supset q) \supset ((X_3v_3 \supset q))]\},0002), \\
& \quad \{\{X_2v_3, X_2v_3 \supset (X_3v_3 \supset q)\}, \\
& \quad (X_3v_3 \supset q) \supset ((X_3v_1 \supset q) \supset ((X_3v_2 \supset q) \supset ((X_3v_3 \supset q))]\},0001\} \}
\end{align*}
\]

\[
\begin{align*}
\text{Flow}(D, w)(u_4) &= \{\{X_2v_1, X_2v_1 \supset (X_3v_1 \supset q)\},X_1v_3, X_1v_3 \supset (X_3v_3 \supset q)\}, \\
& \quad (X_3v_3 \supset q) \supset ((X_3v_1 \supset q) \supset ((X_3v_2 \supset q) \supset ((X_3v_3 \supset q))]\},002), \\
& \quad \{\{X_1v_1, X_1v_1 \supset (X_3v_1 \supset q)\}, \\
& \quad X_2v_3, X_2v_3 \supset (X_3v_3 \supset q)\}, \\
& \quad (X_3v_3 \supset q) \supset ((X_3v_1 \supset q) \supset ((X_3v_2 \supset q) \supset ((X_3v_3 \supset q))]\},001\} \\
& \} \\
\end{align*}
\]

The value returned by \( \text{Flow}(D, w)(u_4) \), in line \([13]\) represents two different deductions of \( \text{OR}X_3 \supset q \). In the sequel, each deduction is associated to its respective place in the original (uncompressed)
Natural Deduction derivation, by following the paths, here 001 and 002. The first is a deduction of $X2v3 \supset q$ and the second is a deduction of $X2v1 \supset q$. The node labeled with $q$ is from where the mentioned paths diverge to each respective conclusion.

Figure 62: Part of the compressed DLDS from figure 57 used in the explanation of unfold.