Instituto Superior Técnico

Bachelor's degree project
Proof-search for non-classical logics

João Afonso Batista

Supervisors:
Carlos Caleiro
Sérgio Marcelino

2019/2020

1 Introduction

For millennia mathematicians made all their calculations by hand. Meticulously writing their answer on papers, spending hours repeating the same process over and over again, simply because they had to.
With the rise of computers and automation, thinkers from all areas could simply stop doing these repetitive chores and use their time to do what really mattered. From the simplest calculator giving a helping hand to the day to day calculations, to the computer that could automatically compute integrals and derivatives, innovation brought upon a new age where tasks that don't require creativity to solve are done by a computer. This project is another step in this big and slow process that changes modern science.

This project's main objective is the development of a proof-search algorithm based on analytic calculi by capitalizing on the work done in [1]. The implementation can be found in [3].
This report starts by introducing the necessary theory one needs to understand how and why the algorithm works. Then it explains the algorithm and its implementation, along with some of its direct applications. After stating and proving an interesting proposition about analyticity, the report ends with some examples.

Given two sets of formulas, Γ and Δ, how can one know if Γ derives Δ? This is the main question tackled in this project. In fact, in some cases, it is even possible to have a computer find the solution. To understand all these concepts, one must first see:

2 The theory behind the curtains

2.1 Logics over propositional signatures

Definition 1 (Propositional signature) *A propositional signature Σ is a $\mathbb{N}$-indexed family of sets $\Sigma = \{\Sigma^{(k)}\colon k \in \mathbb{N}\}$ where $\Sigma^{(k)}$ contains the k-ary* connectives *of Σ. Let $L_\Sigma(P)$ be the set of formulas using* connectives *of Σ and variables in P.*

Example 1 (Propositional signature) *Σ might be the classical signature, where $\Sigma^{(1)} = \{\neg\}$, $\Sigma^{(2)} = \{\vee, \wedge, \Rightarrow\}$, $\Sigma^{(k)} = \emptyset$ for $k = 0$ and $k \geq 2$. An example of a formula in this signature is $p \to (q \wedge \neg p)$, $p, q \in P$. Note that these connectives are merely symbols, they have no intrinsic value or interpretation. This interpretation is given by a logic.*

Defining what a logic is is not an easy task. Logicians have struggled and came up with some solutions. The definition used in this project is the one studied by mathematicians like Shoesmith and Smiley.

Definition 2 (Multiple conclusion logic) *Given a propositional signature Σ, a multiple conclusion logic over Σ is a* consequence relation*, between set of formulas in $L_\Sigma(P)$, satisfying the following four properties. In this project, such a relation will be represented by $\rhd$.*

Let Γ, Γ', Δ and Δ' be any sets of formulas in L_Σ(P):

- (O) if $\Gamma \cap \Delta \neq \emptyset$ then $\Gamma \rhd \Delta$
- (D) if $\Gamma \rhd \Delta$ then $\Gamma, \Gamma' \rhd \Delta, \Delta'$
- (C) if $\Gamma, \Omega \rhd \bar{\Omega}, \Delta$ for every set of formulas Ω, then $\Gamma \rhd \Delta$ ($\bar{\Omega}$ is the set of formulas not in Ω)
- (S) if $\Gamma \rhd \Delta$ then, for any substitution σ, $\Gamma^\sigma \rhd \Delta^\sigma$

A substitution is a map σ: P$\to L_\Sigma$(P). Γ^σ represents the application of σ to all the variables of all formulas in Γ.

Intuitively, $\Gamma \rhd \Delta$ can be read as "every time every formula in Γ is True, then at least one formula in Δ is true".

One simple but important idea is that a logic reflects a way to synthesize what is True and what is False and how one can know which is which. If it is known that every formula in Γ is True and if $\Gamma \rhd \Delta'$ where Δ' consists of one formula Δ'={A}, then A is True. If, for instance, $\emptyset \rhd \Delta'$, then one can understand that A is always True.
It is simple to realize that truth is not absolute. It depends on the logic being used.
The meaning of these properties will be clearer once one analyzes the two lenses through which logicians study logics: semantics and symbolic calculi. Both try to interpret the connectives in Σ and provide easy to work with tools to interact with the logic.

2.2 Semantics, Monadic non-deterministic matrices and valuations

Semantics interpret the connectives by providing a *logical matrix*. One can then proceed to study the logic at hand through valuations.

Definition 3 (Logical matrix) *Given a propositional signature Σ, a* logical matrix M *over Σ is a tuple (V,D,$\circ$) where V is the set of truth-values, D is a subset of V containing the designed truth-values (the designated truth-values may be thought of as the ones that are True, while the other ones are False) and $\circ$ is the* interpretation function. *It contains all the information about how to interpret the connectives in Σ. It is called a* non-deterministic matrix *if its interpretation function is non-deterministic, meaning every connective, given some combination of truth-values, may have more than one possible interpretation.*

A possible generalization is the *partial non-deterministic matrix* where the interpretation function may interpret a connective, given a combination of truth-values, as the empty set. This means that particular combinations of truth-values and connective can never exist.

Example 2 (Deterministic logical matrix) *Let Σ_1 be the classical propositional signature and M_1 be the* logical matrix *($\{0,1\}$,$\{1\}$,$\circ$), where $\circ$ interprets the connectives as follows:*

$\neg$	
0	1
1	0

$\Rightarrow$	0	1
0	1	1
1	0	1

$\vee$	0	1
0	0	1
1	1	1

$\wedge$	0	1
0	0	0
1	0	1

Example 3 (Non-deterministic logical matrix) *Let Σ_2 be formed by a single binary connective $\Rightarrow$ and M_2 be the* logical matrix *($\{0,1,2\}$,$\{2\}$,$\circ$), where $\circ$ interprets $\Rightarrow$ as follows:*

$\Rightarrow$	0	1	2
0	2	1,2	0
1	1,2	1	0,1
2	0	0,1	2

Notice how $\Rightarrow$ connective may be interpreted in various ways, even for the same combination of truth-values. For example 1$\Rightarrow$0 may be 1 or 2.

A valuation over M is a function v: L_Σ(P) $\rightarrow$ V that maps every formula in L_Σ(P) into a truth-value. Valuations have to respect the interpretation function $\circ$ of matrix M. Given a valuation v, let the set Ω_v=$\{A : A \in L_\Sigma(P), v(A) \in D\}$ and its complementary set $\bar{\Omega}_v$=$\{A : A \in L_\Sigma(P), v(A) \notin D\}$.

Example 4 (Valuations) *Let $A_1 = (p \Rightarrow q)$ be a formula in $L_{\Sigma_1}(P)$ and $A_2 = (p \Rightarrow q)$ be a formula in $L_{\Sigma_2}(P)$. Let v_i be valuations such that $v_1(p) = v_1(q) = 0$, $v_2(p) = 0$ and $v_2(q) = 1$.*
Since M_1 is deterministic*, then by simply defining the values of its variables, the value of A_1 is determined, in this case: $v_1(A_1) = v_1(0 \Rightarrow 0) = 1$ and $v_2(A_1) = v_2(0 \Rightarrow 1) = 0$.*
As M_2 is non-deterministic*, a valuation may have to declare the value of A_2, since by simply defining the*

values of its variables, its value may not be determined. For example: $v_1(A_2) = v_1(0 \Rightarrow 0) = 2$ *is determined, but* $v_2(A_2) = v_2(0 \Rightarrow 1) \in \{1, 2\}$*, so for* v_2 *to be well defined,* $v_2(A_2)$ *has to be 1 or 2.*

In the classical signature where there's only two truth-values, 1 (True) and 0 (False), 1 is designated and 0 is not (1 ∈ D, 0 ∉ D), and so, given a truth-value x, it is possible to know which one it is simply by verifying if $x \in D$ or $x \notin D$. If $x \in D$ then x=1, x=0 otherwise.
Generally this is not the case. This is where the notion of a *monadic separator* becomes an important one.

Definition 4 (Monadic separator) *Given a logical matrix M = (V, D, ∘) over a propositional signature Σ and two truth values x,y ∈ V, a formula* $F \in L_\Sigma(P)$ *is said to be a Monadic separator for x and y if F has only one variable, p, and for every pair of valuations v, v' that satisfy v(p) = x and v'(p) = y, v(F) ∈ D and v'(F) ∉ D, or vice versa.*

This means that formula F is useful to distinguish the truth-values x and y.

Definition 5 (Monadic logical matrix and Discriminator) *We say that M is a* Monadic logical matrix *if there is a monadic separator for every pair of truth-values. Let Θ be a family of sets of formulas with only one variable, indexed by x. Θ is called a* discriminator *for M if, for every* $y \in V \backslash \{x\}$*, there is a formula F in* Θ_x *that separates x and y, meaning F(x) ∈ D and F(y) ∉ D, or vice-versa.*

A discriminator is an important tool in distinguishing truth-values. Given a truth-value x, a discriminator for M has the necessary information to uniquely know which truth-value x is.

Example 5 (Discriminator) *Let Σ be composed by a binary connective □, and let M matrix be ({0,1,2},{2},∘), where ∘ interprets □ as follows:*

□	0	1	2
0	1	2	2
1	2	2	2
2	1	2	2

The formula F_1 = p separates *0 and 2 and also 1 and 2, because* $F_1(0) = 0 \notin D$, $F_1(1) = 1 \notin D$ *and* $F_1(2) = 2 \in D$*. The formula* F_2 = p□p *separates 0 and 1 since* $F_2(0) = 0 \square 0 = 1 \notin D$ *and* $F_2(1) = 1 \square 1 = 2 \in D$*. So the set* $\{F_1, F_2\}$ *is a set of separators for M. In this case,* $\Theta_0 = \Theta_1 = \{F_1, F_2\}$ *and* $\Theta_2 = \{F_1\}$*. Given a truth-value* x*, one can know which it is using the formulas in the discriminator. If* $F_1(x) \in D$ *then x=2. If* $F_1(x) \notin D$ *and* $F_2(x) \in D$ *then* x = *1 and* x = *0 otherwise.*

The purpose of a discriminator may be oblivious at this time, but it will be clear that it is a crucial ingredient in the algorithm developed in this project.

Knowing the idiosyncrasies of semantics, it is easy to understand the meaning of the logical relation ⊳. Given sets of formulas in L_Σ(P), Γ and Δ, then Γ ⊳ Δ if, for every possible valuation v, $\Gamma \cap \bar{\Omega}_v \neq \emptyset$ (some formula in Γ is false) or $\Delta \cap \Omega_v \neq \emptyset$ (some formula in Δ is true). One can check that the four properties are verified, thus proving that the relation ⊳ is a logic as defined before:

- (O) if $\Gamma \cap \Delta \neq \emptyset$ then $\Gamma \rhd \Delta$
- (D) if $\Gamma \rhd \Delta$ then $\Gamma, \Gamma' \rhd \Delta, \Delta'$
- (C) if $\Gamma, \Omega \rhd \bar{\Omega}, \Delta$ for every set of formulas Ω, then $\Gamma \rhd \Delta$ ($\bar{\Omega}$ is the set of formulas not in Ω)
- (S) if $\Gamma \rhd \Delta$ then, for any substitution σ, $\Gamma^\sigma \rhd \Delta^\sigma$

(O) - If Γ and Δ have at least one formula in common, ($A \in \Gamma \cap \Delta$) then, for every valuation v , v(A) is either designated and thus $\Delta \cap \Omega_v \neq \emptyset$, or v(A) is undesignated, case where $\Gamma \cap \bar{\Omega}_v \neq \emptyset$.

(D) - Every valuation v that satisfies $\Gamma \cap \bar{\Omega}_v \neq \emptyset$ or $\Delta \cap \Omega_v \neq \emptyset$ also satisfies $(\Gamma \cup \Gamma') \cap \bar{\Omega}_v \neq \emptyset$ or $(\Delta \cup \Delta') \cap \Omega_v \neq \emptyset$ because $\Gamma \cap \bar{\Omega}_v \subseteq (\Gamma \cup \Gamma') \cap \bar{\Omega}_v$ and $\Delta \cap \Omega_v \subseteq (\Delta \cup \Delta') \cap \Omega_v$.

(C) - By contradiction, assume $\Gamma \not\rhd \Delta$, then exists a valuation v that makes every formula in Γ and no formula in Δ designated. Let Ω be the set of all formula v designates, then $\bar{\Omega}$ is set of all formulas v turns into undesignated truth-values. Since v turns every formula in Γ and Ω into designated truth-values and every formula in Δ and $\bar{\Omega}$ into undesignated truth-values, $\Gamma, \Omega \not\rhd \bar{\Omega}, \Delta$, proving the desired proposition.

(S) - One can see that making a substitution to the formulas of Γ and Δ makes no real difference, because one must simply change the valuation v that satisfy the conditions for $\Gamma \rhd \Delta$ for v^σ. Where, if v turn the variable a into the truth-value x, v^σ turns the variable $\sigma(a)$ into x

2.3 Symbolic calculi, inference rules and derivations

Symbolic calculi is another way to interpret the connectives in a propositional signature. By providing a set of inference rules (a calculus) one can understand how to interpret a logical relation $\rhd$ in a completely different way from the semantic lens.

Definition 6 (Inference rule) *A inference rule is a pair composed by a set of formulas, called the premises, and a formula, called the conclusion.*

In this project only multiple conclusion inference rules are considered, which is a generalization of the previous definition, where the conclusion is a set of formulas instead of a single formula.

Example 6 *The following rules give the foundations for understanding and using the classical interpretation for the connective $\Rightarrow$ (the set of formulas above the line is the set of premises and the one below the line is the set of conclusions):*

$$\frac{}{p, p \Rightarrow q} \qquad \frac{p, p \Rightarrow q}{q} \qquad \frac{q}{p \Rightarrow q},$$

Intuitively, these 3 rules, r_1, r_2 and r_3 state, respectively, that:
It is always the case that either p *or* $\mathrm{p} \Rightarrow \mathrm{q}$ *are True.*
If p *and* $\mathrm{p} \Rightarrow \mathrm{q}$ *are True, so is* q.
If q *is True, then* $\mathrm{p} \Rightarrow \mathrm{q}$ *is as well.*

Intuitively, inference rules are, in a way, similar to semantic, since they state that, if the premises are True then at least one of its conclusions is True as well. There is a nuance here, that lies in the fact that in this syntactic lens, no *logical matrix* was given. Meaning there is nothing defining what is Truth. So inference rules have to be interpreted in a slightly different way. That is where derivations enter the picture.

Definition 7 (Derivation) *A derivation is a labeled tree. The label of the root is a set of formulas Γ and every node (son) branching from another node (father) results from the application of a inference rule and its label is the set of formulas of its father united with a conclusion of the applied rule.*

If one has a set of inference rules, one may produce derivations from a certain set of premises Γ to a certain set of conclusions Δ. These start with the set of formula Γ and, in each step, one rule is used, adding its conclusion to the obtained set of formulas until now. Only rules for which every premise belongs to the set of obtained formulas until now can be used. The derivation ends when a formula in Δ is obtained. This idea becomes clearer in the following examples.

Example 7 (Derivation) *With the previous rules and adding the following r_4, r_5 and r_6*

$$\frac{p, q}{p \wedge q} \qquad \frac{p \wedge q}{p} \qquad \frac{p \wedge q}{q}$$

it is possible to produce a derivation from $\Gamma = \{p \wedge (p \Rightarrow q)\}$ to $\Delta = \{q\}$.

$$\{p \wedge (p \Rightarrow q)\}$$
$$|$$
$$\{p \wedge (p \Rightarrow q)\} \cup \{p\}$$
$$|$$
$$\{p \wedge (p \Rightarrow q)\} \cup \{p\} \cup \{(p \Rightarrow q)\}$$
$$|$$
$$\{p \wedge (p \Rightarrow q)\} \cup \{p\} \cup \{(p \Rightarrow q)\} \cup \{q\}$$

In the first and second steps, r_5 and r_6 were used, and then r_2 was applied to conclude q*. Notice that the rules are not directly used, they are instanced first, meaning the variables in the rules are substituted by a formula. In the first step, r_5 was applied as $\frac{p \wedge (p \Rightarrow q)}{p}$, and then r_6 was applied as $\frac{p \wedge (p \Rightarrow q)}{(p \Rightarrow q)}$. Also realize that rules were only used if its set of premises in contained in the set of formulas of the node the rule is being applied to.*

In an attempt to simplify the representation, from now on, derivations will only show the formula obtained in that step. The previous example turns into:

$$p \wedge (p \Rightarrow q)$$
$$|$$
$$p$$
$$|$$
$$(p \Rightarrow q)$$
$$|$$
$$q$$

One can obtain the set of formulas in a node simply by computing the union of all nodes above it.

The previous example was a linear derivation. They are useful and easy to work with. However, when working with multiple conclusion inference rules, a challenge arises: the application of this type of rules will result in the branching of the derivation, producing a tree instead of a line. To end the derivation, each branch of this tree must end with a leaf containing a formula in Δ. (Given a node, its branch is every node that has a direct downwards path to it. By stating a general "branch" then it is implied that it is a branch of a leaf. A leaf is a node that has no nodes branching out from it.)
This is easy to understand intuitively: if a rule states "if these premises are satisfied, then at least one of these conclusions is as well", it is not known which conclusion is satisfied, so every possibility must be tested.

Example 8 (Non-linear Derivation) *A derivation for $\emptyset \rhd \Delta$, where $\Delta = \{p \Rightarrow p\}$, consisting of a application of rule r_1 and then rule r_3, is the following.*

$$\begin{array}{ccc} & \emptyset & \\ & \swarrow \quad \searrow & \\ p & & p \Rightarrow p \\ | & & \\ p \Rightarrow p & & \end{array}$$

Again, although a rule is applied in each step, it must first be instanced. r_1 was used but only after replacing p *for* p *and* q *for* p *in the form of $\frac{}{p, p \Rightarrow p}$. In this example, the nodes with the formula $p \Rightarrow p$ are leaves and, for example, the branch of the left leaf is the left node with the formula $p \Rightarrow p$, the node with the formula p and the node with the empty set.*

Given a propositional signature Σ and a set of inference rules R, one can define a logical relation $\rhd$ as

follows:
$\Gamma \rhd \Delta$ if there is a derivation from Γ to Δ. It is easy to prove this relation satisfies properties (O), (D) and (S). Proving property (C) involves the axiom of choice and Zorn's lemma.

- (O) if $\Gamma \cap \Delta \neq \emptyset$ then $\Gamma \rhd \Delta$
- (D) if $\Gamma \rhd \Delta$ then $\Gamma, \Gamma' \rhd \Delta, \Delta'$
- (C) if $\Gamma, \Omega \rhd \bar{\Omega}, \Delta$ for every set of formulas Ω, then $\Gamma \rhd \Delta$ ($\bar{\Omega}$ is the set of formulas not in Ω)
- (S) if $\Gamma \rhd \Delta$ then, for any substitution σ, $\Gamma^\sigma \rhd \Delta^\sigma$

(O) - If there is a formula in both Γ and Δ then there is a trivial derivation from Γ to Δ consisting only in the first node, the one with the formulas in Γ. That is a derivation since the last node has a formula in Δ.

(D) - If there is a derivation D from Γ to Δ then it is also a valid derivation from $\Gamma \cup \Gamma'$ to $\Delta \cup \Delta'$ since every formula applied in D can also be applied since the set $\Gamma \subseteq \Gamma \cup \Gamma'$. Every branch in D ends with a formula in Δ, which is also a formula in $\Delta \cup \Delta'$

(S) - Since the rules applied in the derivation from Γ to Δ are instanced first, then one must simply change the instances to get a derivation from Γ^σ to Δ^σ. Every instance that turns a variable p into a formula A, one changes for a instance where p turns into the formula $\sigma(A)$

Notice the subtle difference between sematics and symbolic calculi. In the latter, one can make conclusion about logical relation without defining what is truth, as was done in semantics.

When faced with a logic based on a set of rules, one might think about trying to develop a algorithm to automatically create derivations. One big problem instantly arises: the number of possible rules to apply in each step is potentially infinite, since $L_\Sigma(\mathrm{P})$ is infinite. So the algorithm might never end.
If one wants do implement such an algorithm, this problem must be diverted.

2.4 Analytical calculus

Definition 8 (Analytical calculus) *A calculus is said to be analytical if it enjoys the generalized subformula property.*
This property states: if there is a derivation from Γ to Δ, then there is a derivation from Γ to Δ with these two properties:

- *The rules applied in each step are instanced with formulas in the set sub$(\Gamma \cup \Delta)$ (the set of all subformulas of all formulas in Γ and Δ)*
- *The formulas that show up throughout the derivation are in the set sub$(\Gamma \cup \Delta) \cup \{F(a) : F \in \Theta, a \in$ sub$(\Gamma \cup \Delta)\}$ (Θ is a set of formulas with only one variable, called a discriminator)*

Just like that, the problem is solved. Although the number of formulas to check grows exponentially with the number of variables of the inference rules in the calculus, it is always finite, thus, when working with analytical calculi, it is possible to provide an algorithm to provide derivations from sets of formulas.

2.5 Relating semantics and symbolic calculi

As proven in [1], from a *monadic non-deterministic* matrix with *discriminator* Θ, it is always possible to produce a set of inference rules that enjoys the *generalized subformula property* that have the following property: every valuation that turn every formula in Γ into a designated truth-value also turns at least one formula in Δ into a designated truth-value, meaning $\Gamma \rhd \Delta$ (in the semantic sense), if and only if there is a derivation from Γ to Δ using rules in the set of inference rules, meaning $\Gamma \rhd \Delta$ (in the syntatic sense). This means the logic created from the logical matrix is the same as the one created by the analytical calculus.

This amazing results is a bridge between semantics and symbolic calculi. In this project, an algorithm that works with a set of inference rules was implemented. Even in the case someone is working with semantics, this algorithm may be used by first creating a set of inference rules from the *logical matrix*.

If the algorithm finds a derivation from Γ to Δ, that means that $\Gamma \rhd \Delta$, and so, every valuation that makes every formula in Γ designated, also does so for some formula in Δ.
On the other hand. If the algorithm ends without a derivation, then there is no derivation from Γ to Δ, meaning that there is at least one valuation that makes every formula in Γ designated and every formula in Δ not designated. Furthermore, in this case, the algorithm is also able to compute a set of formulas that may provide such valuation.

3 The algorithm

Given a set of inference rules $r_1, r_2, ..., r_n$ (enjoying the generalized subformula property) and a *discriminator* Θ, one can provide a derivation from a set of formulas Γ to another set of formulas Δ (or state that no derivation exists if that is the case) by following these steps:

- 1 - Calculate $\Upsilon = sub(\Gamma \cup \Delta)$ and $\Omega = \Upsilon \cup \{F(A) : F \in \Theta, A \in \Upsilon\}$.
- 2 - Compute D, the set with the instances of all the rules using all the formulas in Υ. From those, eliminate the rules from which appear formulas not in Ω.
- 3 - Start the derivation: (every node is a set of formulas.)
- 3.1 - The first node, also called the root (since the derivation is tree-shaped), is the set Γ.
- Follow these 4 steps until the derivation ends (main cycle):
- 3.2 - Choose an open node (this is, a not expanded node) to expand. Let Ψ be the set of formulas in this node.
- 3.3 - Verify if Ψ contains any formula in Δ. If it does, then close that node and return to step 3.2. If it doesn't, continue to step 3.4.
- 3.4 - Choose, from D, a instantiation of a rule (with set of conclusions E, $\#(E) = n$) to apply that satisfies these 2 conditions: its premises are contained in Ψ and $E \cap \Psi = \emptyset$.
- 3.5 - For each conclusion of the chosen rule, $C_1, C_2, ..., C_n$, add a node to the tree, branching out from the selected node, with set of formulas $\Psi_i = \Psi \cup C_i$, for $i \in$ {1,2,...,n}, return to step 3.2.
- The derivation ends if:
- 4 - All branches are closed, case in which every leaf contains a formula in Δ and one can conclude that there is a derivation from Γ to Δ. Note that the application of a rule with no conclusions also closes a branch. So, actually, every leaf may contain a formula in Δ OR it may be a node to which a rule with no conclusions was applied. (no choice can be made in step 3.2)
- 5 - There is an open node with no possible rules to apply. Case in which there is no derivation from Γ to Δ. (no choice can be made in step 3.4)

3.1 Soundness of the algorithm

This algorithm is being applied to an analytical calculus with a finite number of inference rules, therefore, if there is a derivation from Γ to Δ, then there is a derivation from Γ to Δ where the rules are instanced with formulas in $\Upsilon = sub(\Gamma \cup \Delta)$ and all formulas that appear in the derivation are in $\Omega = \Upsilon \cup \{F(A) : F \in \Theta, A \in \Upsilon\}$.

One must first understand why, in step 3.4, the algorithm does not apply a rule to a node whose set of formulas contains a conclusion of said rule.
Imagine the node chosen in step 3.2 has a set of formulas Ψ, and in step 3.4 a instantiation of a rule is chosen, with set of conclusions E (let #(E)=n=1), such that $\Psi \cap E \supseteq \{C_1\}$. Then in step 3.5 the algorithm will produce n nodes branching out from the selected node, with set of rules $\Psi \cup \{C_i\}, C_i \in E$. Note that $\Psi \cup \{C_1\} = \Psi$ since $C_1 \in \Psi$, therefore, the node created has the same set of formulas as the selected node, meaning the algorithm did not progress one bit towards its destination, it is exactly in the same state as it was before applying this rule. If $\#(E) = n > 1$ then the algorithm is actually in a worse place than it was before, because one of the resulting nodes is the same as the selected one, and then there are n-1 extra nodes to further explore and close.

One consequence of the way the algorithm chooses the rules to apply in step 3.4, is that no instantiation of a rule is used twice in the same branch. This is quite trivial to realize. Given an open node with set of formulas Ψ, every rule r_i with set of conclusions E_i applied to the nodes in the branch of the selected node are such that $\Psi \cap E_i \neq \emptyset$, so r_i will never be applied to the selected node.

Notice that, as a consequence of the previous statement, this algorithm will end in finite time. The number of applicable rules, N, is finite, so the tree will have, in the worst case, a depth of N. (since no instantiation of a rule is used twice, after the N's step, there is no more rules to apply, thus satisfying condition 5 and ending the main cycle.)
If the used calculus has k inference rules, the i'th rule has n_i variables and $\#(sub(\Gamma \cup \Delta)) = z$, then there are $\sum_{i=1}^{k} z^{n_i}$ instanced rules.

It is settled that the algorithm will always terminate in finite time, but is it correct? To see that it is, one must break down the two possible cases.

In the case where $\Gamma \rhd \Delta$.
As the logical relation $\rhd$ was based on symbolic calculi, this means there is a derivation from Γ to Δ. As stated before, this means there is a derivation D from Γ to Δ where the rules are instanced with formulas in $\Upsilon = sub(\Gamma \cup \Delta)$ and all formulas that appear in the derivation are in $\Omega = \Upsilon \cup \{F(A) : F \in \Theta, A \in \Upsilon\}$.
This is proved by contradiction. Start by assuming that, in fact, there is a derivation from Γ to Δ, but the algorithm reached a node η that satisfies condition 5. This can never happen and here is why.

There are two derivation that are going to be mentioned: D, and the derivation that is being computed by the algorithm. Remember that D has a formula from Δ in every leaf (or a leaf is a node where a rule with no conclusions was applied to). Let the branch of η be B (B is the set of all nodes that lead to η) and remember that the set of formulas in η is the union of all formulas that appear throughout branch B, let it be called Ψ_η. Obviously $\Gamma \subseteq \Psi_\eta$ because the first node (which set of formulas is Γ) appears in every branch of the derivation.
The first instanced rule applied in D, r_1, is such that all of its premises are in Γ (otherwise it could not be used). The assumption made is that every possible usable instantiation of a rule was applied in branch B, so r_1 was applied in branch B, therefore, at least one of r_1 conclusions, C_1, is in Ψ_η.
In a similar line of thinking, the rule r_2 applied in D, to the node that is formed after applying r_1 (the application of rule r_1 forms n new nodes, where n is the number of conclusions of r_1, I am referring to the node formed by the conclusion C_1), has its premises contained in $\Gamma \cup \{C_1\}$ (otherwise it could not be applied). Again, as every possible usable instantiation of a rule was applied in branch B, r_2 was applied in branch B.
The idea is repeated. Now one can conclude that one of r_2's conclusions, C_2, is in Ψ_η, and thus rule r_3 (the rule applied in D to the node formed by the application of rule r_2) was used in branch B.
So every rule of a certain branch in D was used in branch B, but in every branch of D, there is a rule r_n that concludes only formulas in Δ (or the $\emptyset$) so Ψ_η contains a formula in Δ (or a rule with no conclusions

was applied to η) and thus η would be closed in some step 3.3. This is a contradiction since the assumption was that η would satisfy condition 5.
This proves condition 5 will never be met. As proven above, the algorithm always ends, so the algorithm will eventually reach condition 4, concluding there is a derivation from Γ to Δ.

In the case where $\Gamma \not\triangleright \Delta$.
Since Γ and Δ are not in the relation, then there is no derivation from Γ to Δ. It is easy to see that condition 4 of the algorithm will never be satisfied (if it was, then the algorithm would have found a derivation, but none exist), so the algorithm will only stop when condition 5 is met. As shown before, the algorithm always ends in finite time so condition 5 will eventually be met, and so, as stated in step 5, the algorithm concludes there is no derivation from Γ to Δ.

3.2 The implementation

There are a lot of ways to implement this algorithm. For the same pair (Γ, Δ), with the same calculus, one implementation may produce a very small tree with 10 nodes while another one may produce a tremendous tree with thousands of nodes. Obviously there is an interest in producing an implementation that is efficient, meaning it has to explore the minimum number of nodes and does not require the process of extensive computations.
The most important factors in the efficiency of the implementation are:

3.2.1 The Node

The choice made in step 3.2 of the algorithm can alter greatly the number of iteration of the main cycle. There are two cases to analyze:

$\Gamma \triangleright \Delta$: In this case the algorithm will end when condition 4 is satisfied. Meaning every branch has to close, every branch must be analyzed. Since that's the case, it does not matter which node is chosen because it is inevitable to visit all open nodes eventually.

$\Gamma \not\triangleright \Delta$: In this case the algorithm ends when condition 5 is satisfied. There's a big difference between this case and the previous one. Being that in the previous, every branch had to meet a condition: having a formula of Δ in its leaf (or ending with the application of a rule with no conclusions). In this case only one branch has to meet a condition, the first branch where there is no possible rule to apply ends the algorithm, so the node's choice matters greatly. A good way to tackle this difficulty is with a LIFO (Last In First Out) approach. By doing a depth search the "bad branch" will be found without having to spent computation time with other "healthy branches". If there's a way to understand, given a tree, which node will most likely lead to a "bad branch", then that's the best choice. ("bad branch" means a branch that will end the proof, that will eventually have no applicable rules. "healthy branch" is one that will never meet that condition.)

3.2.2 The Rule

The choice made in step 3.4 is by far the most important, and a very difficult one to optimize. There is no efficient process or algorithm that determines which is the single best rule to be used. Even for a human mind, this choice is not obvious and requires a lot of creativity. Nevertheless, there are certainly some choosing criteria that are better than others. Here are some thoughts helpful in solving this conundrum.

One may take the easiest approach and select a random rule. And although that solution might make the smallest possible derivation, the probability of such an event occurring is exponentially smaller the bigger the derivation. This solution is inconsistent and overall awful. Nevertheless, the algorithm will end and it

will provide the correct answer, and that's the beauty in all of this.

There is a theoretical function $\tilde{f}$ that, given a tree, a node chosen to expand and a instantiation of a rule to apply to it, returns a real number that rates how good that rule is. If one had access to this function, he could simply find the best rule and apply it. Producing the best possible derivation tree. The sad truth is: no one knows how to implement $\tilde{f}$ efficiently, at least at the moment. One can only try to approximate this function, implementing the best heuristic function h possible. When faced with the choice in step 3.4, h returns a rating for every rule, and the obvious choice is the rule with the highest rating. This function h is the hardest and most important piece concerning the efficiency of the implementation. Note that in different nodes, the same instantiation of a rule might be very good or very bad.

One possible criteria to choose the rules is to give higher priority to rules with fewer conclusions first. This will lead to a smaller branching rate, diminishing the computational effort of the algorithm. If the algorithm first applied a single conclusion rule and then one with two conclusions, the tree has 4 nodes. If the algorithm applied the rules on the other order, then the tree would have 5 nodes.

The more information h uses, the better its approximation of $\tilde{f}$. From all instantiations of rules computed in step 2, it is known which ones will end a branch. It might be useful to compute the set containing these *finishing rules*, which have one of two possible characteristics: it has no conclusions, or all its conclusions are in Δ. The application of any rule with one of these properties will close the branch it was applied to.
When expanding a node where all the premises of a *finishing rule* are contained in its set of formulas, then that rule can be applied and its branch closes.
Furthermore, instead of trying to reach a formula in Δ, when choosing a rule to apply, the algorithm only needs to try to reach the premises of one of these *finishing rules*. Bringing the finish line one step closer.

Thinking backwards, instead of bringing the finish line closer, one might try to get closer to the finish line as well. It is possible to compute all possible trees branching out from the node being expanded, to a depth of n. And after picking the best outcome, choose the rule that began that tree. As these trees grow exponentially, this is a very costly heuristic function. For n bigger than 3, it slows the algorithm a lot. But for a n=2, the heuristic becomes considerably better without compromising the overall efficiency of the algorithm.

The core idea in my implementation of h is the following. Instead of trying to rate rules, h first rates formulas, all formulas computed in step 1. The real number associated with the rule r is the minimum of the rate of its conclusions. This makes sense since the application of a rule will result in the chosen node to branch out into n branches, where n is the number of conclusions of the rule. The algorithm will have to close every resulting branch, so the criteria that a rule is as bad as the worst branch it opens makes sense. The formulas are rated by how much closer they bring this branch to a *finishing rule*.

Deterministic Heuristic functions tend to be worse than stochastic ones. So in case of ties, the choice is made randomly.
This may be counter intuitive, since the idea of perfection is never associated with randomness. Nevertheless, areas like artificial intelligence, that use a lot of heuristic functions, support these theses. Reasoning that a deterministic function will always act the same way, and when confronted with problem it struggles with, it will always be stuck. This lies in the fact that it is very very hard, most times simply impossible, to make a deterministic heuristic function that's satisfactory in every single case.

3.3 Some examples

The purpose of this subsection is to show some examples where the algorithm was put to test.

These first three examples are based on the classic Hilbert calculus, consisting of these 11 rules and discriminator $\Theta = \{p\}$:

$$\frac{p,\neg p}{}\,r_1 \qquad \frac{}{p,\neg p}\,r_2 \qquad \frac{p\wedge q}{p}\,r_3 \qquad \frac{p\wedge q}{q}\,r_4 \qquad \frac{p,q}{p\wedge q}\,r_5 \qquad \frac{p}{p\vee q}\,r_6$$

$$\frac{q}{p\vee q}\,r_7 \qquad \frac{p\vee q}{p,q}\,r_8 \qquad \frac{}{p,p\Rightarrow q}\,r_9 \qquad \frac{p,p\Rightarrow q}{q}\,r_{10} \qquad \frac{q}{p\Rightarrow q}\,r_{11}$$

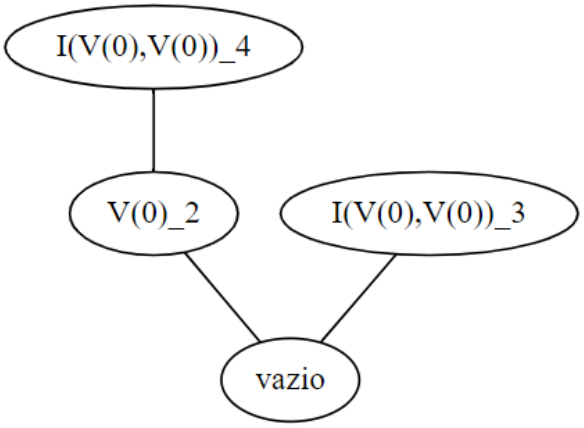
The implementation is initialized as follows, stating the symbols used for the connectives (N for $\neg$, I for $\rightarrow$, E for $\wedge$ and O for $\vee$), their arity and the discriminator, followed by the rules and the semantic interpretation of the connectives (the semantic part is not mandatory but it is necessary to compute, in the case where $\Gamma \not\triangleright \Delta$, a valuation v that satisfies $\Gamma \cap \bar{\Omega}_v = \emptyset$ and $\Delta \cap \Omega_v = \emptyset$):

```
CH = logica(["N","I","E","O"],[1,2,2,2],["V(0)"])

CH.adiciona_regra_inferencia(["V(0)","N(V(0))"],[],["V(0)"])
CH.adiciona_regra_inferencia([],["V(0)","N(V(0))"],["V(0)"])
CH.adiciona_regra_inferencia(["E(V(0),V(1))"],["V(0)"],["V(0)","V(1)"])
CH.adiciona_regra_inferencia(["E(V(0),V(1))"],["V(1)"],["V(0)","V(1)"])
CH.adiciona_regra_inferencia(["V(0)","V(1)"],["E(V(0),V(1))"],["V(0)","V(1)"])
CH.adiciona_regra_inferencia(["V(0)"],["O(V(0),V(1))"],["V(0)","V(1)"])
CH.adiciona_regra_inferencia(["V(1)"],["O(V(0),V(1))"],["V(0)","V(1)"])
CH.adiciona_regra_inferencia(["O(V(0),V(1))"],["V(0)","V(1)"],["V(0)","V(1)"])
CH.adiciona_regra_inferencia([],["V(0)","I(V(0),V(1))"],["V(0)","V(1)"])
CH.adiciona_regra_inferencia(["V(0)","I(V(0),V(1))"],["V(1)"],["V(0)","V(1)"])
CH.adiciona_regra_inferencia(["V(1)"],["I(V(0),V(1))"],["V(0)","V(1)"])

dicionario = {"N(t)":["f"],
 "N(f)":["t"],
 "I(t,t)":["t"],
 "I(t,f)":["f"],
 "I(f,t)":["t"],
 "I(f,f)":["t"],
 "O(t,t)":["t"],
 "O(f,t)":["t"],
 "O(t,f)":["t"],
 "O(f,f)":["f"],
 "E(t,t)":["t"],
 "E(f,t)":["f"],
 "E(t,f)":["f"],
 "E(f,f)":["f"]}

CH.adiciona_atribuiçoes(dicionario)
CH.adiciona_valores_de_verdade(["t","f"],["t"])
```

The first example is trivial, where $\Gamma = \emptyset$ and $\Delta = \{p \Rightarrow p\}$. The algorithm does succeed in producing a valid derivation from Γ to Δ, and does so in 1 millisecond, by first applying r_9 and then r_{11}:

```
CH.prova([],["I(V(0),V(0))"],True, tempo = True)
```

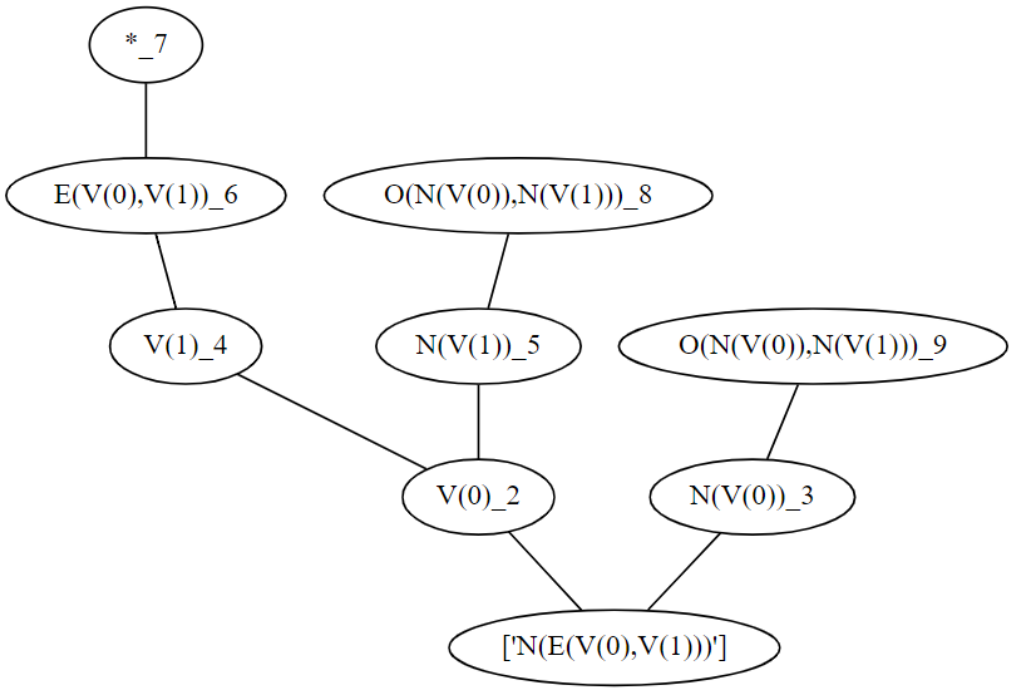
```
tempo de execução (nao conta com o desenhar da arvor): 0.0009980201721191406
```



As shown by the next example, it is possible to prove the De Morgan rules (only $\{\neg(p \wedge q)\} \rhd \{(\neg p \vee \neg q)\}$is shown) using the Hilbert Calculus. Notice how the application of rule r_1, on the last leaf, results the creation of zero nodes. Since r_1 has zero conclusions, this is represented by a node with the * symbol.

```
CH.prova(["N(E(V(0),V(1)))"],["O(N(V(0)),N(V(1)))"],True, tempo = True)
```

```
tempo de execução (nao conta com o desenhar da arvor): 0.008975505828857422
```



The third application of the algorithm is $\Gamma = \{p \Rightarrow (q \Rightarrow x), q \wedge x\}$ and $\Delta = \{\neg(x \Rightarrow q), \neg p\}$. These sets of formulas are not in relation ($\Gamma \not\sim\!\!\!\rhd \Delta$), therefore, there is a valuation v such that $\Gamma \cap \bar{\Omega}_v = \emptyset$ and $\Delta \cap \Omega_v = \emptyset$. When called with the *contra_modelo* argument as True, the algorithm calls its semantic part and produces such a valuation.

```
CH.prova(["I(V(0),I(V(1),V(2)))","E(V(1),V(2))"],["N(I(V(2),V(1)))","N(V(0))"],False, tempo = True,contra_modelo=True)
```

```
tempo de execução (nao conta com o desenhar da arvor): 0.008976221084594727

{'V(0)': 't',
 'V(1)': 't',
 'V(2)': 't',
 'I(V(1),V(2))': 't',
 'I(V(0),I(V(1),V(2)))': 't',
 'E(V(1),V(2))': 't',
 'I(V(2),V(1))': 't',
 'N(I(V(2),V(1)))': 'f',
 'N(V(0))': 'f'}
```

The calculus used in the previous examples is *deterministic*. To show how broad this algorithm is, it was applied to the following $non-deterministic$ calculus which is based on a propositional signature with a single unary connective ($\neg$, represented by N) and two binary connectives ($\wedge$, represented by E, and $\vee$, represented by O). There are four $truth-values$: T, t, f and F, where T and t are the only designated. The interpretation function interprets the connectives as follows:

$\neg$	
f	t
F	F
T	T
t	f

$\vee$	f	F	T	t
f	f,T	t,F	T	t
F	t,F	t,F	t	t
T	T	t	T	t
t	t	t	t	t

$\wedge$	f	F	T	t
f	f	f	f	f
F	f	f,F	f	f,F
T	f	f	T	T
t	f	f,F	T	t,T

With this calculus, the De Morgan rule used in the second example is actually false, and the algorithm gives a valuation v that proves it.

```
S.prova(["N(E(V(0),V(1)))"],["O(N(V(0)),N(V(1)))"],True, tempo = True,contra_modelo=True)
```

```
tempo de execução (nao conta com o desenhar da arvor): 0.005983591079711914

{'V(0)': 'F',
 'V(1)': 'F',
 'E(V(0),V(1))': 'f',
 'N(E(V(0),V(1)))': 't',
 'N(V(0))': 'F',
 'N(V(1))': 'F',
 'O(N(V(0)),N(V(1)))': 'F'}
```

Notice how stating the $truth-value$ to the 2 variables is not enough. $v(V(0)) = \text{F}$ and $v(V(1)) = \text{F}$ and so $v(E(V(0), v(1)))$ could either be f or F: if $v(E(V(0), v(1))) = \text{F}$ then $v(N(E(V(0), v(1)))) = \text{F}$ and thus v would not satisfy the conditions required. When confronted with a $non-deterministic$ calculus, a valuation must evaluate formulas as well as variables.

4 Other uses

Logic has been a mathematical branch with a relatively slow computational development, since most work requires the human mind to come up with proof, shortcuts and solutions. But recent development in the area has shown that that's not the case.
Once this algorithm is implemented, its range of possibilities go beyond its direct application. There are a number of tedious, and most times hard processes that mathematicians had to do that can now be automated as well.

4.1 Comparing sets of rules

Given a set of inference rules R_1, one may think of all the derivations possible using rules in R_1. Let C_{R_1} be the set of pairs $\{\Gamma,\Delta\}$ for which there is a derivation to Δ from Γ using only rules in R_1. Given a analytical set of inference rules R_2, one question may arise. It is possible to, using only rules in R_2, produce a derivation for every pair in C_{R_1}? (R_1 and R_2 are based on the same propositional signature)
Using the implemented algorithm, this seemingly hard question has a trivial answer.
One simply has to check, for every rule $r \in R_1$, if there is a derivation from r's premises to r's conclusions using only rules in R_2. If that turns out to be the case, then the answer is positive and $C_{R_1} \subseteq C_{R_2}$, (negative otherwise and $C_{R_1} \not\subset C_{R_2}$). Why is this the case?

Imagine there is a derivation D from Γ to Δ where, in each step i, a instantiation of a rule $r_i \in R_1$ is applied. If there is a derivation D_i from r_i's instanced premises to r_i's instanced conclusion, using only rules in R_2, then it is possible to replace step i in derivation D for derivation D_i. When this process is done for every step in D, the resulting derivation D' still goes from Γ to Δ, but using only rules in R_2.
On the other hand, if there is at least one instantiation of a rule $r \in R_1$ for which there is no derivation from r's instanced premises to r's instanced conclusions using only rules in R_2. Then one can produce a derivation from r's instanced premises to r' instaced conclusions using only rules in R_1, by a simple application of r, and can't using only rules in R_2.

This idea may be used to prove either two distinct sets of rules R_1 and R_2 have the same strength in the sense that $C_{R_1} = C_{R_2}$. Simply by calling the algorithm twice, once to conclude $C_{R_1} \subseteq C_{R_2}$ and another time for $C_{R_2} \subseteq C_{R_1}$.

4.2 Simplifying axiomatizations

Given an analytical set of inference rules R, one may, using the previous result, obtain subsets $R_i \subseteq R$ for which $C_R = C_{R_i}$. One might be interesting in finding the smallest possible R_i that satisfies this, as it is easier to work with. Since R_i is a subset of R, then it is obvious that $C_{R_i} \subseteq C_R$. Therefore, there is only one relation to check: $C_R \subseteq C_{R_i}$.
In the worst case, one could simply check every subset of R, which is finite if R is finite.
One way to avoid a mindless search is to implement a recursive program that, given a set of rules R with n rules, creates every subset R_i of R with n-1 rules, checks which ones satisfy $C_R \subseteq C_{R_i}$, repeating the process for the ones that do.
Since the relation $\subseteq$ is *transitive*, if $C_R \subseteq C_{R_1}$ and $C_{R_1} \subseteq C_{R_2}$ then $C_R \subseteq C_{R_2}$.

Concerning this algorithm, one curious mind might wonder: if R is an analytical calculus and $R_i \subseteq R$ such that $C_R = C_{R_i}$, what can be said about the analyticity of R_i? To understand this matter, one must first dive into:

5 Θ-analiticity vs ∅-analiticity

This project works with aximatizations that satisfy the *generalized subformula property.* But what certainties does one have about the simplified calculus obtained with the previous algorithm? To answer this, one must first understand the subtle but important difference between a Θ-analytical derivation and a ∅-analytical derivation.

Definition 9 *Given a set of rules and a* discriminator Θ*, a derivation from* Γ *to* Δ *is said to be*

Θ*-analytical if the rules applied in each step are instanced with formulas in the set sub(*Γ ∪ Δ*) and the formulas that appear in the nodes are in the set sub(*Γ ∪ Δ*) ∪ {F(A) : F ∈ Θ, A ∈ sub(*Γ ∪ Δ*)}.*

∅*-analytical if the rules applied in each step are instanced with formulas in the set sub(*Γ ∪ Δ*) and the formulas that appear in the nodes are also in the set sub(*Γ ∪ Δ*).*

One of the main statements from this project

Proposition 1 *Given a analytical calculus R, there are subcalculus* $\tilde{R}$ *of R such that* C_R=$C_{\tilde{R}}$ *but* $\tilde{R}$ *is not an analytical calculus.*

The proof to this result is presented in the form of an example later in this report.

From a intuitive perspective: $\tilde{R} = R\backslash\{r_1, r_2, ...r_n\}$ and there is a derivation D_i using only rules in $\tilde{R}$ that goes from r_i's premises to r_i's conclusions. If there is a derivation *D'* from Γ to Δ using rules in *R*, by replacing, in *D'*, every step where r_i was applied by D_i, then one get a derivation $\tilde{D'}$ from Γ to Δ only using rules in $\tilde{R}$. Obviously $\tilde{R}$ is *complete*, since it can provide a derivation for every pair of sets of formulas which *R* can provide a derivation.

The problem lies in the fact that D_i may have formulas in {F(A), $F \in \Theta, A \in \Upsilon_i$} (where Υ_i is the set of subformulas of r_i's premises and r_i's conclusions). So it may use a *F(A)* for an *A* that is not in sub(Γ ∪Δ), so *F(A)* could not appear in $\tilde{D'}$, if $\tilde{D'}$ was Θ-analytical. This fact will become clearer in the example.

One can divert this problem by introducing a broader characterization of Θ-analyticity.
It is true that the smaller set of rules $\tilde{R}$ may not be able to provide a Θ-analytical derivation for Γ ▷ Δ. On the other hand, if one relaxes the restrictions set by Θ-analyticity, then it is also true that there is a derivation from Γ to Δ using rules in $\tilde{R}$ where their instances are made with formulas in Ω = sub(Γ ∪ Δ) ∪ {$F(A) : F \in \Theta, A \in sub(\Gamma \cup \Delta)$} and that the formulas that appear throughout the derivation are in the set sub(Γ ∪ Δ) ∪ {$F(A) : F \in \Theta, A \in sub(\Gamma \cup \Delta)$} ∪ {$F(B) : F \in \Theta, B \in \{F(A) : F \in \Theta, A \in sub(\Gamma \cup \Delta)\}$} because the formulas needed to make the derivations D_i are present. In a sense, the calculus is Θ^2-analytical. Applying the algorith to a Θ^2-analytical calculus might result in an even weaker restriction, here called Θ^3-analytical, where the instantitations of the rules are done with formulas in sub(Γ∪Δ) ∪ {$F(A) : F \in \Theta, A \in sub(\Gamma \cup \Delta)$} ∪ {$F(B) : F \in \Theta, B \in \{F(A) : F \in \Theta, A \in sub(\Gamma \cup \Delta)\}$} and the formulas that appear on the nodes are in the set sub(Γ ∪ Δ) ∪ {$F(A) : F \in \Theta, A \in sub(\Gamma \cup \Delta)$} ∪ {$F(B) : F \in \Theta, B \in \{F(A) : F \in \Theta, A \in sub(\Gamma \cup \Delta)\}$} ∪ {$F(C) : F \in \Theta, C \in \{F(B) : F \in \Theta, B \in \{$F(A):F∈ Θ, $A \in sub(\Gamma \cup \Delta)\}$}.

On the other hand, it is easy to realize that, if the derivation from r_i's premises to r_i's conclusions using rules in $\tilde{R}$ are ∅-analytical then $\tilde{R}$ maintains the analyticity .
If D_i are ∅-analytical then all formulas that appear throughout D_i belong in the set sub(Υ_i) (where Υ_i is the set of subformulas of r_i's premises and r_i's conclusions). It is true that some of these formula might not be in sub(Γ∪Δ) ∪ {$F(A) : F \in \Theta, A \in sub(\Gamma \cup \Delta)$}, but that is not a problem because if that was the case,

then rule r would have not been applied with such a instantiation.
For example. Image one is trying to get rid of the rule that, from p and q, concludes $p \wedge q$. Then in derivation D_i $p \wedge q$ might appear. Now think that $p \wedge q$ does not belong in $\text{sub}(\Gamma \cup \Delta) \cup \{F(A) : F \in \Theta, A \in sub(\Gamma \cup \Delta)\}$, then obviously rule r was not applied in the derivation D' from Γ to Δ.

Proposition 2 *Given a analytical calculus with set of rules R and a subcalculus $\tilde{R} = R \backslash \{r_1, r_2, ...r_n\}$, if there are $\emptyset$-analytical derivations D_i, that only use rules in $\tilde{R}$, that go from r_i's premises to r_i's conclusions (for i=1,2,...,n), then C_R=$C_{\tilde{R}}$ and $\tilde{R}$ is an analytical calculus.*

5.1 The example

This example proves *proposition 1.* It starts with a analytical calculus with 53 inference rules R and *discriminator* $\Theta = \{p, \nabla p, \neg p, \nabla(\neg\, p)\}$, based of a propositional signature with two 1-ary connective, $\neg$ and ∇, and two 2-ary connectives, $\vee$ and $\wedge$. The algorithm explained in section 5.2 was applied to R and it found a subset with 50 rules, R'. Then the algorithm found a subset with 49 rules, R''. Then I show that R'' is not an analytical calculus:

$$\frac{}{\nabla p, \neg p}\, r_1 \quad \frac{}{\nabla p, \neg \nabla p}\, r_2 \quad \frac{}{\nabla p, \neg(p \wedge q)}\, r_3 \quad \frac{p}{\nabla p}\, r_4 \quad \frac{p}{\neg\neg p}\, r_5 \quad \frac{p}{p \vee q}\, r_6 \quad \frac{q}{p \vee q}\, r_7 \quad \frac{p \wedge q}{p}\, r_8 \quad \frac{p \wedge q}{q}\, r_9 \quad \frac{p \wedge q}{\nabla p, \neg q}\, r_{10}$$

$$\frac{\nabla(p \wedge q)}{\nabla p}\, r_{11} \quad \frac{\nabla(p \wedge q)}{\nabla q}\, r_{12} \quad \frac{\nabla(\nabla p)}{\nabla p}\, r_{13} \quad \frac{\nabla(\neg(\nabla p))}{\neg p}\, r_{14} \quad \frac{\nabla(\neg(\neg p))}{\nabla p}\, r_{15} \quad \frac{\neg p}{\neg(p \wedge q)}\, r_{16} \quad \frac{\neg q}{\neg(p \wedge q)}\, r_{17} \quad \frac{\neg q}{\nabla p, \neg(p \vee q)}\, r_{18} \quad \frac{\neg(p \wedge q)}{\neg p, \neg q}\, r_{19} \quad \frac{\neg(\nabla p)}{\neg p}\, r_{20}$$

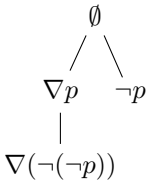
$$\frac{\neg(\neg p))}{p}\, r_{21} \quad \frac{\neg(p \vee q)}{\neg p}\, r_{22} \quad \frac{\neg(p \vee q)}{\neg q}\, r_{23} \quad \frac{p \vee q}{p, q}\, r_{24} \quad \frac{p \vee q}{q, \nabla p}\, r_{25} \quad \frac{p, q}{p \wedge q}\, r_{26} \quad \frac{\nabla(\neg p)}{q, \nabla(\neg(p \vee q))}\, r_{27} \quad \frac{p, \nabla(\neg(\nabla p))}{}\, r_{28} \quad \frac{\nabla(\neg(p \vee q))}{\nabla(\neg p)}\, r_{29} \quad \frac{p}{\nabla(p \wedge q), \neg q}\, r_{30}$$

$$\frac{p, \nabla(\neg p)}{}\, r_{31} \quad \frac{\nabla(\neg p)}{q, \nabla(\neg(p \vee q))}\, r_{32} \quad \frac{\nabla(\neg q)}{p, \nabla(\neg(p \vee q))}\, r_{33} \quad \frac{\nabla(\neg(p \vee q))}{\nabla(\neg q)}\, r_{34} \quad \frac{q}{\nabla(p \vee q), \neg p}\, r_{35} \quad \frac{\nabla p, \nabla(\neg \nabla(p))}{}\, r_{36} \quad \frac{\nabla p}{\nabla(\neg(\neg p))}\, r_{37} \quad \frac{\nabla p}{\nabla(p \wedge q), \neg(q)}\, r_{38}$$

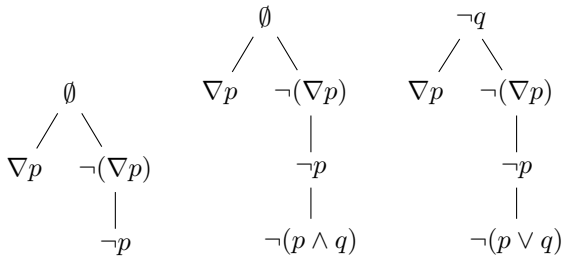
$$\frac{\nabla q}{\nabla(p \wedge q), \neg(p)}\, r_{39} \quad \frac{\nabla p}{\neg(\nabla p)}\, r_{40} \quad \frac{\nabla q}{\nabla(p \vee q)}\, r_{41} \quad \frac{\nabla p}{\nabla(p \vee q)}\, r_{42} \quad \frac{\neg p}{q, \nabla(\neg(p \vee q))}\, r_{43} \quad \frac{\neg q}{p, \nabla(\neg(p \vee q))}\, r_{44} \quad \frac{\nabla(p \vee q)}{\nabla p, \nabla q}\, r_{45} \quad \frac{\neg p, \neg q}{\neg(p \vee q)}\, r_{46}$$

$$\frac{\nabla(\neg p)}{\nabla(\neg(p \wedge q))}\, r_{47} \quad \frac{\nabla(\neg q)}{\nabla(\neg(p \wedge q))}\, r_{48} \quad \frac{\nabla(\neg(p \wedge q))}{\nabla(\neg p), \nabla(\neg q)}\, r_{49} \quad \frac{\nabla q}{\nabla(p \wedge q)}\, r_{50} \quad \frac{\nabla p}{\nabla(p \wedge q)}\, r_{51} \quad \frac{\nabla(\neg p), \nabla(\neg q)}{\nabla(\neg(p \vee q))}\, r_{52} \quad \frac{\nabla(\neg q), \neg p}{\nabla(\neg(p \vee q))}\, r_{53}$$

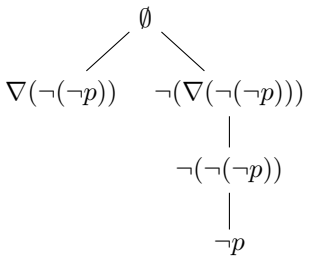
With the 53 rules in R, the algorithm can find a derivation from the empty set to $\{\nabla(\neg(\neg p)), \neg p\}$ by first applying r_1 and then r_{37}. Notice how the derivation is Θ-analytical since ∇p is no in the set $\text{sub}(\{\nabla(\neg(\neg p)), \neg p\})$ but is a F(A) for F=$\nabla p \in \Theta$ and $A = p \in sub(\{\nabla(\neg(\neg p)), \neg p\})$



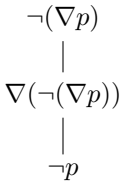
There is a subset with 50 rules $R' = R \backslash \{r_1, r_3, r_{18}\}$. R' can, Θ-analytically derive r_1, r_3 and r_{18}, here are the derivations found by the algorithm. The rules used in the first derivation are r_2 and r_{20}, in the second are r_2, r_{20} and r_{16}, in the third are r_2, r_{20} and r_{46}.



Note that R' can find a Θ-analytical derivation from the empty set to $\{\nabla(\neg(\neg p)), \neg p\}$ (using r_2, r_{20} and r_{21}), this does not mean R' is an analytical calculus (although I have a feeling it is):

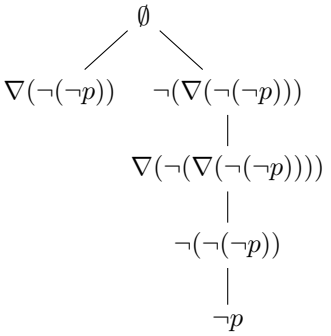


Now there is a subset with 49 rules $R'' = R'\backslash\{r_{20}\}$. R'' can, in fact, derive Θ-analytically r_{20} (using r_4 and r_{14}) notice how r_{20} was used in the 3 derivations used to prove that R' could derive r_1, r_3 and r_{18}:



And so, the example ends, as the implementation of the algorithm states that R'' can not derive Θ-analytically $\{\nabla(\neg(\neg p)), \neg p\}$ from the empty set. R'' was obtained from R by the algorithm described in section 5.2. In each step of such algorithm, the next subset was obtained Θ-analytically and in the end, R'' is not Θ-analytical.

Understand that R'' can derive $\{\nabla(\neg(\neg p)), \neg p\}$ from the empty set, simply not Θ-analytically. Here is a derivation where rule r_4 was instantiated with the formula $\neg(\nabla(\neg(\neg p)))$, which is not in sub($\{\nabla(\neg(\neg p)), \neg p\}$) = $\{\nabla(\neg(\neg p)), \neg(\neg p), \neg p, p\}$ (using r_2, r_4, r_{14} and r_{21}):



Using the notation introduced earlier, this is a Θ^2-analytical derivation, since the rule 4 was intantiated with a $F(B)$, $F{=}(\neg p) \in \Theta$, $B = \neg(\nabla(\neg(\neg p))) \in \{F(A) : F \in \Theta, A \in sub(\Gamma \cup \Delta)\}$.

I would like to end this section with my opinion on how the analyticity evolves as rules are taken from a calculus.
If there is an analytical set of rules R and the algorithm finds a subset $R_1{=}R\backslash\{r_1\}$, then there are derivations D_{1i} from r_1's premises to r_1's conclusions using rules in R_1. One can imagine that the derivations D_{1i} are connections between R_1 and R.
In future iterations of the algorithm, some rules might be taken from R_1, lets say $R_2{=}R_1\backslash\{r_2\}$ and that there is a derivation D_2 from r_2's premises to r_2's conclusions using rules in R_2. If r_2 was not used in any D_{1i},

then these are still connections from R_2 and R. If r_2 was used in some D_{1i}, then there are four possible cases:

- D_{1i} and D_2 are Θ-analytical, in which case D_{1i} is no longer a connection from R_2 to R.
- D_{1i} is Θ-analytical but D_2 is $\emptyset$-analytical, in which case D_{1i} is still a connection from R_2 to R, one simply changes every step in D_{1i} where r_2 was applied with D_2.
- D_{1i} is $\emptyset$-analytical but D_2 is Θ-analytical, in which case D_{1i} is still a connection from R_2 to R, but the steps where r_2 was changed by D_2 are Θ-analytical, meaning that, if in some future iteration, some rule from R_2 that was applied in D_2, is taken Θ-analyticaly, then this connection breaks.
- D_{1i} and D_2 are $\emptyset$-analytical, in which case D_{1i} is still a connection from R_2 to R.

After a few steps on the algorithm, one can imagine a layered structure where each layer is a set of rules, resulting from eliminating one rule from the previous layer. From each layer there are connections to the previous one, meaning that there are derivations to the rule that was taken. My final statement is: to understand a level of analyticity of a subset of R, $\tilde{R} \subseteq R$, travel through the structure from $\tilde{R}$ to R, every time there is no connection from one level to the previous one, then the analyticity of $\tilde{R}$ worsens.
A quick example:
R_1=$R\backslash\{r_1\}$, there is only one derivation, that is Θ-analytical, from r_1's premises to r_1's conclusions using rules in R_1, that uses r_3, among others.
R_2=$R_1\backslash\{r_2\}$, there is only one derivation, that is Θ-analytical, from r_2's premises to r_2's conclusions using rules in R_2, that uses r_4, among others.
R_3=$R_2\backslash\{r_3\}$, there is only one derivation, that is Θ-analytical, from r_3's premises to r_3's conclusions using rules in R_3.
R_4=$R_3\backslash\{r_4\}$, there is only one derivation, that is Θ-analytical, from r_4's premises to r_4's conclusions using rules in R_4.
I would say that, if R is Θ-analytical, then R_1 and R_2 are as well, R_3 is Θ^2-analytical and R_4 is Θ^3-analytical.

This is what my intuition, after exploring some examples, tells me.

6 The connection between the algorithm and semantics

As explained before, if one has a *monadic non-deterministic logical matrix*, there is a process to obtain a analytical calculus, thus making this algorithm applicable.
In the case where there is a derivation from Γ to Δ, the algorithm return a derivation from Γ to Δ, so one can conclude that $\Gamma \rhd \Delta$ and thus, every valuation that turns every formula in Γ into a designated truth-value also turns at least one of the formulas in Δ into a designated truth-value.
If there is no derivation from Γ to Δ, then the algorithm reaches a node η where every possibles usable instantiation of a rules was already used in its branch. One concludes that $\Gamma \not\rhd \Delta$, meaning there is at least one valuation that turns every formula in Γ into a designated truth-value and every formula in Δ into an undesignated truth-value. The algorithm is powerful enough that it is able to produce such a valuation. One must simply make a valuation that turns every formula in node η into a designated truth-value.
This valuation always exists. Every possible usable instantiation of a rule was used in the branch of the node *eta*, let Γ^+ be the set of formulas in *eta* (Γ^+ only has formulas of Γ, formulas in $sub(\Gamma \cup \Delta)$ and in $\{F(A) : F \in \Theta, A \in sub(\Gamma \cup \Delta)\}$), then it is true that there is no Θ-analytical derivation from Γ^+ to Δ, as the rules form a analytical calculus, there is no derivation from Γ^+ to Δ, thus concluding the proof, because there is a valuation that makes every formula in Γ^+ designated and no formula in Δ designated.
Follows a big difference between *monadic logical matrices* and *non-monadic logical matrices*. If the calculus

comes from a *monadic logical matrix*, then there is a discriminator Θ, and one can find such a valuation. For every formula A, $v(A) = x$ where x is designated if $A \in \Gamma^+$ and x is undesignated if $A \notin \Gamma^+$. Utilizing Θ, that has all the information necessary to uniquely identify a truth-value, one can then find the truth-value x for every formula in $sub(\Gamma \cup \Delta) \cup \{F(A) : F \in \Theta, A \in sub(\Gamma \cup \Delta)\}$.

Out of curiosity and as a term of comparison, another algorithm based on semantic was implement. Given a *logical matrix* and two sets of formulas Γ and Δ, to check if Γ is in relation with Δ, $\Gamma \rhd \Delta$, one simply has to check for every possible valuation, if that valuation turns at least one formula in Γ into a undesignated truth-value or if that valuation turns at least on formula in Δ into a designated value. If that is the case for every valuation, then $\Gamma \rhd \Delta$. If there is at least one valuation that turns every formula in Γ in a designated truth-value and no formulas in Δ into a designated truth-value, then $\Gamma \not\rhd \Delta$.
If there are a finite number of truth-values n and the number of variables in Γ and Δ m is also finite, then there are n^m possible valuations to try (this is for a *deterministic matrix*, for a *non-deterministic matrix* there are even more valuations since it must define what choices it does when faced with multiple possibilities given a connective and a combination of truth-values).

In fact, this algorithm works, it is correct, and it is very fast when Γ and Δ are relatively small. Nevertheless, it is easy to see this is impractical for cases with more variables. Even in the simplest case with 2 truth-values, if this algorithm is faced with a problem with 20 variables, then it has to check at least 2^{20} valuations. It will end, but it will require a huge computational effort.

7 Putting the algorithm to the test

After implementing the algorithm in Python, it was put to the test with real examples. These can be found on the Jupiter notebook in [3].
When faced with calculus with a small number of rules (less than 60), the implementation tends to be very fast. For example, the derivation for the De-Morgan rules in the Hilbert calculus are found in miliseconds.

To test the limits of the algorithm, it was put against a SAT solver.

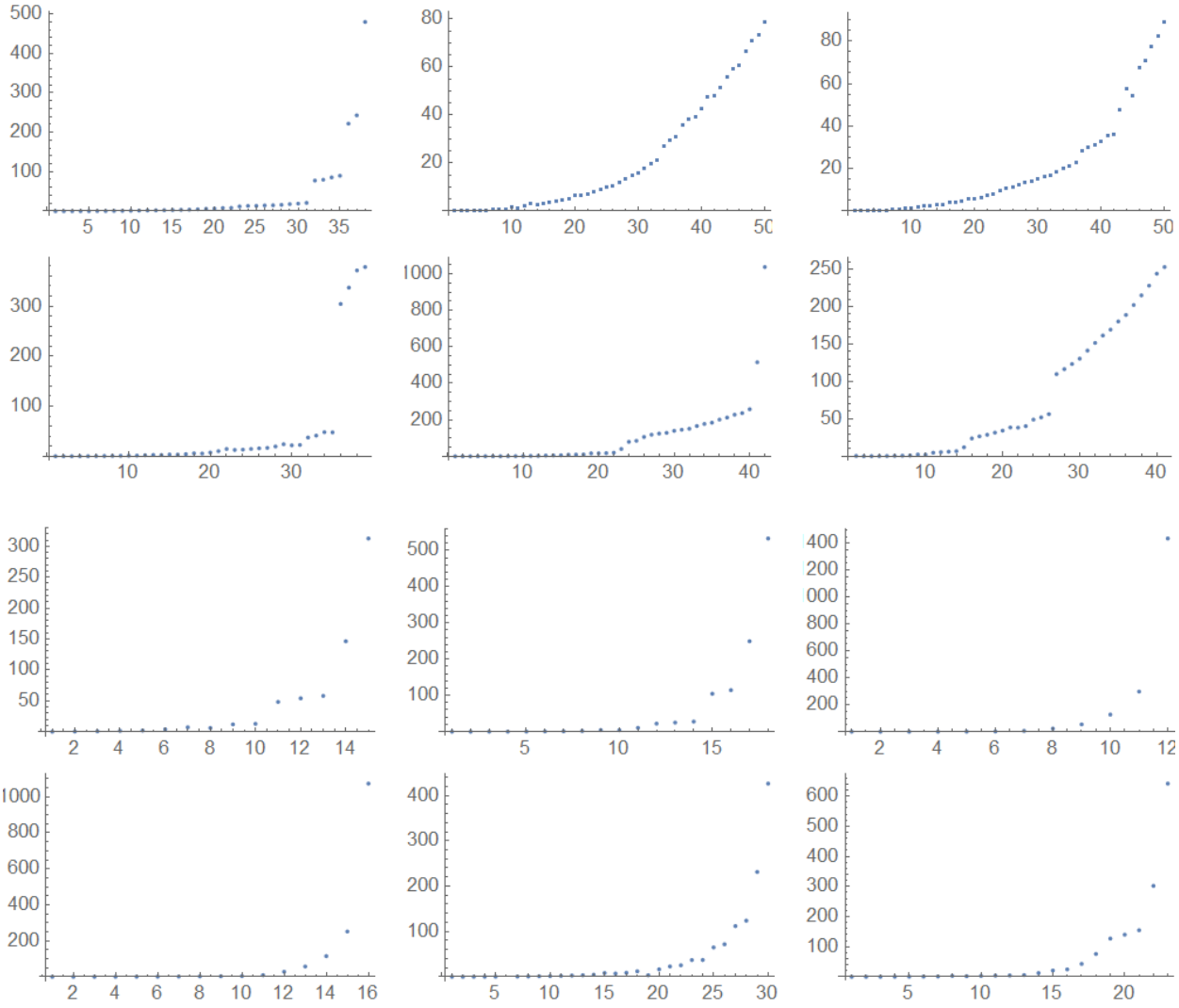
Definition 10 (SAT problem) *A SAT problem is a set of formulas A, in the classic Hilbert logic, with a specific form: Each formula is $p_1 \vee p_2 \vee p_3$ where p_1, p_2 and p_3 can either be a variable or its negation. A SAT problem is said to be satisfiable is there is a valuation that makes every formula in A True. It is said to be unsatisfiable otherwise. The problem is, given an A, knowing if it is satisfiable or not.*

SAT problems have been studied intensely since the 70's decade due to their extensive usages, and because of this, there are very sophisticated and optimized programs, called SAT solvers, to tackle these types of problems.
This is obviously a semantical problem, but remember that the classic Hilbert logic can be seen by the syntatic lens with a set of 11 rules, as explained in [1].
As explained in section 2.2, $\Gamma \rhd \Delta$ if and only if, for every valuation v, $\Gamma \cap \bar{\Omega}_v \neq \emptyset$ or $\Delta \cap \Omega_v \neq \emptyset$ ($\bar{\Omega}_v$ is set set of formulas v turns into a undesignated truth-value and Ω_v is the set of formulas v turns into designated truth-values). If $\Delta = \emptyset$, then $\Delta \cap \Omega_v = \emptyset$ and so $\Gamma \rhd \emptyset$ if and only if, for every valuation v, $\Gamma \cap \bar{\Omega}_v \neq \emptyset$, meaning every v turns at least one formula in Γ into a undesignated truth-value.
Note that this means the SAT problem A is satisfiable if and only if $A \not\rhd \emptyset$.

Example 9 (SAT problem) *Here is a simple example of a SAT problem with 5 variable and 6 formulas: $A = \{(5 \vee \neg 2 \vee \neg 3), (\neg 3 \vee \neg 1 \vee \neg 2), (\neg 3 \vee \neg 2 \vee 4), (1 \vee \neg 5 \vee \neg 3), (2 \vee 5 \vee 3), (\neg 1 \vee \neg 4 \vee \neg 2)\}$. This is a satisfiable SAT problem since the valuation* v *makes every formula True.*
$v(1) = False, V(2) = False, v(3) = True, v(4) = False, v(5) = False$

Follows a Plot that shows the running time of the algorithm for six satisfiable SAT problems, randomly

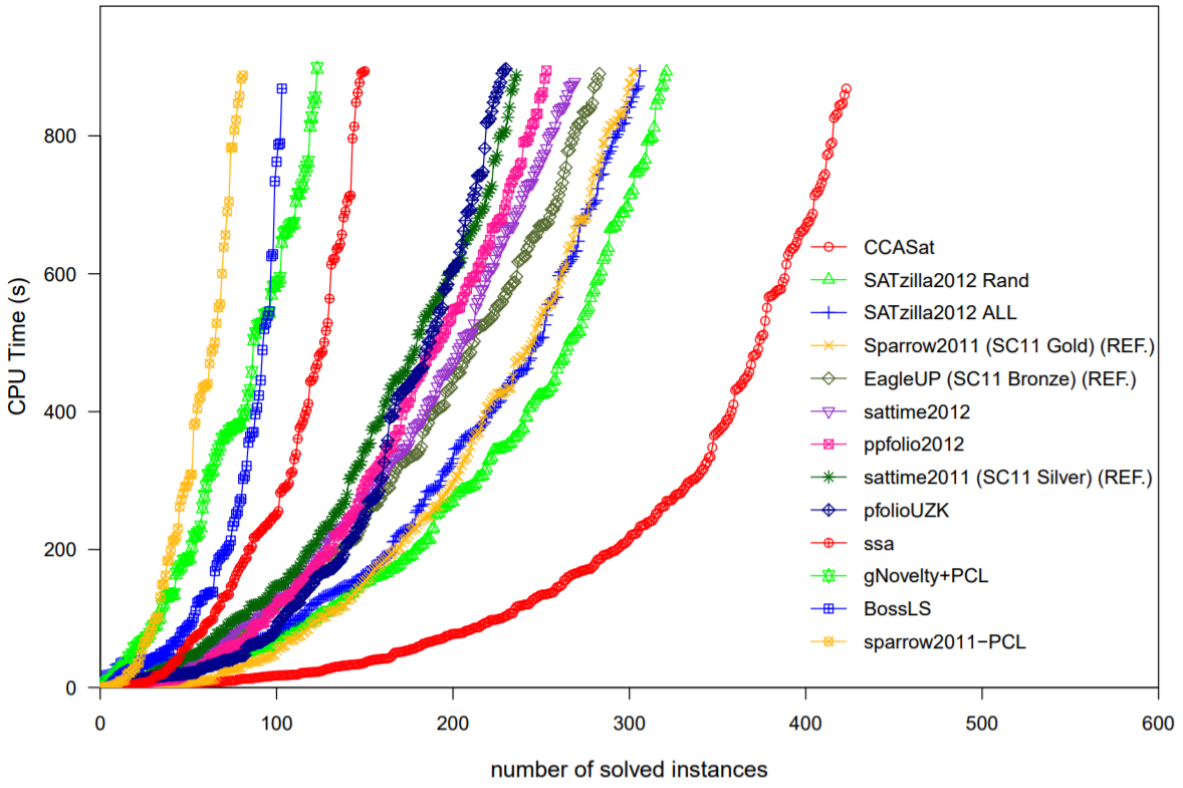
generated with 20 variables. To see how the running time evolves, the algorithm solved the problem consisting of the first formula, then the first and the second, then the first 3, and so on and so on. The x axis represents the number of formulas and the y axis the running time. (and then for six SAT problems with 50 variables)



Those jumps happen when a formula that makes the problem extra harder is introduced. With more computational power, plots with more data could be done and the result would be clearer: the time tends to grow exponentially. When the semantic algorithm is faced with problem of theses sizes it simply does not end in a practical time.

This next plot is taken from a SAT solver competition in 2012 [2]. This competition used instances with a maxinum number of variables ranging from 100 to 2000.

It is true that my implementation is worse than these SAT-solvers. Nevertheless, these programs have totally different objectives and range of application. SAT solvers have been improved for more than 40 years and are specialized in a certain logic and a certain problem while my implementation can be applied to any analytical calculus over any finite propositional signature.



Cactus plot for the Random SAT track. The total number of benchmarks in the track was 600.

8 Conclusion

After reading this report one now better understands multiple conclusion logics, seen through two different lenses, semantics and symbolic calculi, as well as how to work with logical relations.
It is still unknown how to explore the analyticity of a complete subset of an analytical calculus. With the algorithms introduced and developed in this project, this and other questions will be easier to solve.
The human mind is a brilliant tool, maybe one of the most exquisite and mysterious in the universe. Nevertheless, it is limited in various ways. Computers still lack a lot of what the human mind hides, but they can make calculations on a completely different scale. Every algorithm that shines light into what computers are capable of, is a step in advancing humanity: analytical calculi can now be more easily manipulated by computers.

References

[1] S. Marcelino and C. Caleiro. *Axiomatizing non-deterministic many-valued generalized consequence relations, 2019.*

[2] A. Balinta, A. Belovb, M. Järvisaloc, C. Sinzd. *Overview and Analysis of the SAT Challenge 2012 Solver Competition.*
`https://www.cs.helsinki.fi/u/mjarvisa/papers/bbjs.aij15.pdf`

[3] J. Batista. Repository with the implementation.
`https://github.com/JoaoAfonsoBatista/Projeto-de-Licenciatura`