Master Internship's report

Vladislas de Haldat
Supervised by Simon Guilloud and Viktor Kunčak, EPFL

August 27, 2025

General context

With the advent of computers in the second half of the XXth century, the question whether machines are able or not to *reason* and to automatically prove mathematical statements rised from its ashes. In the 60s, a convenient method has been exhibited by Davis and Putnam [DP60] and has proved itself in term of efficiency so far. It consists in taking the negation of the formula one wants to prove and to transform it into conjunctive normal form (CNF for short). This normal form allows the computer to efficiently derive a proof using a couple of logical rules only. Then, if the algorithm finds an assignement to the derived formula, it means that the original problem is not valid. Otherwise, it is valid. Doing all that, we are in fact reducing the problem to the SAT problem that is known to be NP complete. This implies that, supposing $P \neq NP$, no deterministic algorithm is able to decide the problem in polynomial time. However, the use of heuristics, that is to say, the use of non-deterministic algorithms helps to decrease the time spent to solve a SAT problem. Nowaday, most of the top SAT solvers rely on that method and do optimizations on the data structures they use. The conjunctive normal form is, of course, not the only normal form that has been tried to get better performance to automatic reasonning. In the beginning of the 2000s, researchers tried to take advantage of the negation normal form (NNF for short), arguing that the transformation to CNF could induce a lot a redundancy in the resulted formula [JC09]. The singularity of the NNF is that the negation operator is only applied on litterals. According to rewriting rules of boolean algebra, one can derive the negative normal form of a given formula in linear time.

Research problem

In the following, we tackle a generalization of classical logic that is called *orthologic* (OL for short) and we try to take advantage of some of its properties to preprocess formulas but also to derive an efficient proof search algorithm on some specific class of problems. In particular, we try to improve an existing proof search algorithm in the framework of OL that has a cubical worst case complexity. Indeed, the SAT solvers are good in

average, but they can be very slow on classes of problems that are intuitively easy or that have enough structure to be efficiently decided by an algorithm. We improve the state of the art solvers by using alternative technique to solve these specific classes.

Your contribution

The contributions of this report are two fold. First, we show that the use of another normal form, before transforming a formula into CNF, can improve the solving performances of modern SAT solvers. Second, we improve an algorithm for orthologic proof search. Despite having the same theoretical complexity, the original version would essentially always hit the worst case whereas our implementation avoids this. On problems of interest, our improved algorithm performs better than both preexisting OL algorithm and SAT solvers.

Arguments supporting its validity

To support our claims, we evaluated our algorithms on different benchmarks, borrowed or generated by ourselves. Concerning the proof search algorithm, we provide, in addition to the evaluation on benchmarks, proofs of its correctness and of its worst case complexity. Furthermore, let us note that our results do not rely on any kind of assumptions but rather on work of previous peers on the topic.

Summary and future work

To sum up, we propose two ways to improve the current state of the art of SAT solvers. One way handles the preprocessing of the formulas while the other handles the proof search algorithm for certain class of problems. Next, the immediate step to do is to implement our preprocessing and our proof search algorithm within an existing SAT solver, such as Minisat or Kissat, and test whether the performances are improved, compared to the original version. There are many questions that stand further. The main one is to study the lifting to first order logic and the way to adapt our results to specific theories such as bit vectors, lists, arrays etc.

Acknowledgements

Before anything else, I'd like to warmly thank Simon and Viktor for having had me in their laboratory at EPFL and, moreover, for the support they provided me to study that concealed, yet rich topic of orthologic. Many thanks also to Auguste, Samuel and Sankalp for the help and the nice discussions we had.

1 Introduction

In this section, we shall introduce the key concepts and the notations that will be mainly used in the present report. Let us go back in the time. Few years after the advent of quantum mechanics, a rigorous mathematical foundation has been proposed by von Neumann [vNB18] leading, after some observations [BN36], to a non-classical logical structure that would get rid of the distributivity identities that do not behave well in the framework of quantum mechanics. The underlying algebra to such a structure is the class of the so called *orthomodular lattices*, rather than the classical boolean algebras. This gave birth to a new field of logic called the *quantum logic*. The term of *orthologic* has first been introduced by Goldblatt [Gol74] to refer to the logic that corresponds to the algebraic class of *orthocomplemented lattices* or *ortholattices* for short. Those are a generalization of the boolean algebras and, in particular, give up the axiom of distributivity. For ortholattices are weaker than orthomodular lattices, it is also often called the *minimal quantum logic*. First, we shall give some basic notions about lattices, after what, we will introduce the notion of ortholattices. Then we will take a closer look to a normal form of the orthologic and, finally, we will present a proof system for orthologic.

Definition 1 (Lattice)**.** *A lattice $\langle L, \wedge, \vee \rangle$ is an algebraic structure where L is a set and $\wedge$ and $\vee$ are two binary, commutative and associative operations satisfying the following axiomatic identities, also called absorption laws ;*

$$a \vee (a \wedge b) = a$$
$$a \wedge (a \vee b) = a$$

From those follow the idempotent laws, $a \wedge a = a$ and $a \vee a = a$. Moreover, the lattice is said to be bounded when there exists two elements $\bot$ and $\top$ such that $x \vee \top = \top$, $x \wedge \bot = \bot$ and $x \vee \bot = x \wedge \top = x$.

Remark. *A lattice can also be seen as a partial order, in which case, we associate it the following order relation.*

$$a \leq b \Leftrightarrow a = (a \wedge b)$$

Or, equivalently, if and only if $b = (a \vee b)$.

Definition 2 (Generated lattice)**.** *A lattice is said to be generated by a family of elements (X_i) if its elements consist in the X_i's and their finite combinations by $\wedge$ and $\vee$.*

Definition 3 (Free lattice)**.** *A free lattice generated by a family of elements (X_i) is a lattice in which there is no other laws of equality than the ones derived by the axiomatic identities of the lattice.*

1.1 Ortholattices

As for classical logic with boolean algebras or, similarly, intuitionnistic logic with Heyting algebras, orthologic also has an underlying algebraic structure that we call *orthocomplemented lattices* or *ortholattices* for short. To introduce it, we need a very last notion that is the *orthocomplement*.

Definition 4 (Orthocomplement)**.** *An orthogonal complementation (or orthocomplement for short) on a bounded lattice L is a function $\neg : L \to L$ that respects the complement law $\neg a \vee a = \top$ and $\neg a \wedge a = \bot$, the involution law $\neg\neg a = a$ and, finally, the order-reversing law $a \leq b$ implies $\neg b \leq \neg a$.*

We now have all the ingredients to properly define the ortholattices. An *ortholattice* is a bounded lattice that comes with an *orthocomplement* $\neg$. We show, in the table 1, an axiomatization of ortholattices that is a sum up of all the axiomatic identities we have seen so far. With that set, one can show that boolean algebras are special cases of ortholattices, in particular, the cases where distributivity holds.

$x \vee y = y \vee x$	$x \wedge y = y \wedge x$
$x \vee (y \vee z) = (x \vee y) \vee z$	$x \wedge (y \wedge z) = (x \wedge y) \wedge z$
$x \vee x = x$	$x \wedge x = x$
$x \vee \top = \top$	$x \wedge \bot = \bot$
$x \vee \bot = x$	$x \wedge \top = x$
$\neg\neg x = x$	
$x \vee \neg x = \top$	$x \wedge \neg x = \bot$
$\neg(x \vee y) = \neg x \wedge \neg y$	$\neg(x \wedge y) = \neg x \vee \neg y$
$x \vee (x \wedge y) = x$	$x \wedge (x \vee y) = x$

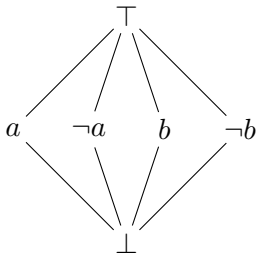
Table 1: Axiomatization of ortholattices

Furthermore, note that this set of axiom is not minimal, but it highlights the duality between $\wedge$ and $\vee$, as well as the duality between $\bot$ and $\top$. For the minimal set of axioms of ortholattices, refer to [McC98]. We shall write $a = b$ if and only if $a \leq b$ and $b \leq a$. Remark also that ortholattices are weaker than orthomodular lattices since they do not admit the following law as an axiom:

$$a \leq b \Rightarrow a \vee (\neg a \wedge b) = b$$

In the following of the report, we will denote orthologic by the acronym OL. Let us illustrate ortholattices through the following example.

Example 5. *The following is an ortholattice, while it is not a boolean algebra.*



This lattice is known as the M_4 lattice. Let us consider the relation $x \wedge (\neg x \vee y) \leq y$. Although it is true in boolean algebras, it does not hold in M_4. Take, as a counterexample, the mapping x to a and y to b.

1.2 Normalization

Given two elements a and b, one would like to know whether $a \leq b$, $a = b$ or $b \leq a$ hold. This is commonly known as the *word problem*. Whitman proposes a procedure to solve it on free lattices [Whi41] and that relies on the following relations ;

$$\bigwedge a_i \leq x \Leftrightarrow \exists i, a_i \leq x \tag{1}$$

$$x \leq \bigvee b_i \Leftrightarrow \exists i, x \leq b_i \tag{2}$$

$$\bigvee a_i \leq x \Leftrightarrow \forall i, a_i \leq x \tag{3}$$

$$x \leq \bigwedge b_i \Leftrightarrow \forall i, x \leq b_i \tag{4}$$

Those relations are consequences of the axiomatization of free lattices. The important relation stressed by Whitman, and that holds only in free lattices, is the following ;

$$\bigwedge a_i \leq \bigvee b_j \Leftrightarrow \exists j, \bigwedge a_i \leq b_j \text{ or } \exists i, a_i \leq \bigvee b_j$$

Several decades later, the word problem for *free ortholattices* has been shown to be solvable in quadratic time [Bru76] and, more recently, an algorithm has been proposed to efficiently compute such normal forms [GBMK23]. The important property of the normal form for orthologic we are about to expose, is that the computed transformation is not larger than the original formula. The following result exhibits the normal form for disjunction, it works dually for conjunction.

Theorem 6 (Normal form [RF95])**.** *A term that is a litteral is in normal form. A term $t = t_1 \vee \cdots \vee t_m$, with $m > 1$ is in normal form if and only if ;*

1. *if $t_i = \bigwedge t_{ij}$, then for all j, $t_{ij} \not\leq t$*
2. *the family $(t_1, ..., t_n)$ forms an antichain meaning that, if $i \neq j$ then $t_i \not\leq t_j$*

Inductive functions to ensure those properties are explicited in [GBMK23] and are actively used in the proposed algorithm.

Theorem 7. NF_{OL} *is a computable normal form for ortholattices.*

Theorem 8 (Transformation's complexity [GBMK23])**.** *The normal form for OL can be computed by an algorithm with a complexity in time and in space belonging to* $\mathcal{O}(n^2)$*. Moreover, the resulted form is guaranteed to be the smallest in the equivalence class of the input terms.*

1.3 Proof system

In order to use orthologic as a strong approximation of classical logic, we need to handle non-logical axioms. To that end, one has to go beyond normal form and introduce a proof system. There has been many proposals of proof systems and, among them, we may highlight [Lau17] and [GK24]. The second one precisely has the advantage to handle non-logical axioms and to have a better complexity, therefore, we will focus on it for our purposes.

Definition 9 (Orthologic's sequent)**.** *If* ϕ *is a formula, we say that* ϕ^L *and* ϕ^R *are annotated formulas. A sequent is a set of at most two annotated formulas.*

Therefore, to express the sequent, we will use the notation $\phi^\diamond, \psi^\circ$, where ϕ and ψ are formulas and $\diamond, \circ \in \{L, R\}$. By upper case greek letters Γ and Δ, we denote a set of annotated formula being either empty, or a singleton. Note that, in the sequent calculus Gentzen [Gen35] exhibits for intuitionnistic logic, the right side of the sequent can contain at most one formula. More generally, the proof system presented can be thought of a sequent calculus of classical logic with the restriction that the sequents contain at most two formulas, whatever is the side they belong to. A sequent can be interpreted with ortholattice in the following way. Suppose the sequent ϕ^L, ψ^R, this is exactly $\phi \leq \psi$ in ortholattices. Now, suppose the sequent ϕ^L, ψ^L, this is interpreted by $\phi \leq \neg\psi$. Dually, we have that ϕ^R, ψ^R is interpreted by $\neg\phi \leq \psi$. Finally, ϕ^L is interpreted by $\phi \leq \bot$ and ϕ^R is interpreted by $\top \leq \phi$.

Theorem 10 (Soundness and completeness [GK24])**.** *The orthologic proof system is sound and complete.*

There is two remaining results that are of great importance for what follows that are the cut rule elimination in orthologic and the subformula property.

Definition 11. *An instance of the cut rule has rank 1 if either of its premises is an axioms. It has rank 2 if either of its premises is the conclusion of a rank 1 cut rule.*

Theorem 12 (Cut elimination [GK24])**.** *If a sequent is provable in the proof system 1 with axioms* $(a_i^\circ, b_i^\diamond) \in A$*, where* $\circ, \diamond \in \{L, R\}$*, then there is a proof of that sequent from the same axioms such that,*

$$\frac{\Gamma, \phi^R \qquad \phi^L, \Delta}{\Gamma, \Delta}\ (\textsc{Cut})$$

$$\frac{\Gamma}{\Gamma, \Delta}\ (\textsc{Weaken})$$

$$\frac{}{\phi^L, \phi^R}\ (\textsc{Hyp}) \qquad \frac{\Gamma \cup \Delta \in A}{\Gamma, \Delta}\ (\textsc{Ax})$$

$$\frac{\phi^L, \Delta}{(\phi \wedge \psi)^L, \Delta}\ (\wedge\text{-}L) \qquad \frac{\Gamma, \phi^R \qquad \Gamma, \psi^R}{\Gamma, (\phi \wedge \psi)^R}\ (\wedge\text{-}R)$$

$$\frac{\phi^L, \Delta \qquad \psi^L, \Delta}{(\phi \vee \psi)^L, \Delta}\ (\vee\text{-}L) \qquad \frac{\Gamma, \phi^R}{\Gamma, (\phi \vee \psi)^R}\ (\vee\text{-}R)$$

$$\frac{\Gamma, \phi^R}{\Gamma, (\neg\phi)^L}\ (\neg\text{-}L) \qquad \frac{\phi^L, \Delta}{(\neg\phi)^R, \Delta}\ (\neg\text{-}R)$$

Figure 1: Proof system for propositional orthologic with axioms

1. *all instances of the cut rule use only formulas among $a_1, ..., a_n, b_1, ..., b_n$ as cut formulas*

2. *all instance of the cut rule are rank 1 or 2*

Corollary 13 (Subformula property for orthologic [GK24])**.** *If a sequent S has a proof in the proof system 1 with axioms, then it has such a proof where each formula in each sequent occurring in the proof is a subformula of S or a subformula of an axiom.*

First, let us remark that the law of *excluded middle* is provable in OL, the corresponding sequent being $\phi^R, (\neg\phi)^R$. Furthermore, notice the presence of axioms in this proof system. Indeed, starting with a base knowledge allows to prove more formulas within the OL framework. To see how, let us sketch the following example.

Example 14. *Suppose again $x \wedge (\neg x \vee y) \leq y$. We have seen, in the example 5 that this relation is not valid in ortholattices and, hence, the sequent $(x \wedge (\neg x \vee y))^L, y^R$ is not provable in orthologic without the axiom rule. However, considering the axiom rule, it is possible, by using the axiom $(x \wedge (\neg x \vee y))^R$, to prove the sequent y^R. The following*

is the proof. First, let us prove $(\neg x \vee y)^R$.

$$\dfrac{\dfrac{}{(x \wedge (\neg x \vee y))^R}\ Ax \quad \dfrac{\dfrac{}{(\neg x \vee y)^L, (\neg x \vee y)^R}\ Hyp}{(x \wedge (\neg x \vee y))^L, (\neg x \vee y)^R}\ (\wedge\text{-}L)}{(\neg x \vee y)^R}\ Cut$$

We shall call π *that former proof. Now, let us prove* y^R *;*

$$\dfrac{\dfrac{\pi}{(\neg x \vee y)^R} \qquad \dfrac{\dfrac{}{y^L, y^R}\ Hyp \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{(x \wedge (\neg x \vee y))^R}\ Ax \quad \dfrac{\dfrac{}{x^L, x^R}\ Hyp}{(x \wedge (\neg x \vee y))^L, x^R}\ (\wedge\text{-}L)}{x^R}\ Cut}{(\neg x)^L}\ (\neg\text{-}L)}{(\neg x)^L, y^R}\ Weaken}{(\neg x \vee y)^L, y^R}\ (\vee\text{-}L)}{y^R}\ Cut$$

2 Preprocessing

Since the procedure presented in the paper of Davis and Putnam [DP60], the practical methods to decide whether a given formula is satisfiable or not has remained more or less the same. The trick is to normalize the original formula into conjunctive normal form and then, learn knowledge on litterals and propagate it. The Tseitin's transformation [Tse83] allows one to compute the conjunctive normal form of a formula in linear time. However, the resulting formula is usually larger than the original one and carries a lot of redundant information. Nowaday, SAT solvers do some preprocessing on the given CNF formula to speed up the solving. As previously explained, the normal form in OL can be computed in worst-case quadratic time and has the good property of not being larger than the original formula. Therefore, the immediate question to ask is whether a SAT solver answers faster to a formula that has been first normalized into OL normal form and then, into CNF. The main problem to that question is the lack of benchmarks that would contain formulas not in conjunctive normal form that still remain hard for SAT solvers to decide. To test our approach, we mainly used two kind of benchmarks.

2.1 Formula generation

The first way is to generate ourselves such formulas according to a procedure proposed by Navarro and Voronkov [NV05]. The idea is to craft formulas by alternating the disjunction and conjunction operations, given a family of integers, called the *shape*, that specifies the arity of the operators at each level of depth. Formally, suppose a

family of integers $(k_1, \cdots, k_n)$ such that $k_i \geq 2$, the sets of formulas $[\![k_1, \cdots, k_n]\!]$ and $\langle k_1, \cdots, k_n \rangle$ are inductively defined such as the following.

1. if $n = 0$ then $[\![]\!]$ and $\langle\rangle$ contain literals only.
2. if $n \geq 1$ then $[\![k_1, \cdots, k_n]\!]$ is the set of conjunctions of arity k_1 of formulas in $\langle k_2, \cdots, k_n \rangle$. Dually, $\langle k_1, \cdots, k_n \rangle$ is the set of disjunctions of arity k_1 of formulas in $[\![k_2, \cdots, k_n]\!]$.

To the purpose of our experimentations, we isolated the specific shape $\langle 6, 3 \rangle$ that generates short – in term of nodes – and hard enough formulas to do appropriate measures.

2.2 Circuits

The second way to do the benchmarks is to take real-life circuits written in the And-Inverted Graph (AIG) or the International Symposium of Circuits and Systems (ISCAS) formats. Those have the advantage to be raw and, hence, not in conjunctive normal form. For the AIG format, we used the arithmetic circuits offered by the EPFL combinatorial benchmark suite [AGDM15]. Older, but not easier for the solvers, is the ISCAS benchmark, that contains 21 circuits, all unsatisfiable and generated in the formal verification of correct superscalar microprocessors [Vel01].

2.3 Evaluation

We evaluated our implementation on both of the benchmarks, the generated one and the AIG/ISCAS one, sticking to the following method. Given a formula F, on one hand we generate its CNF and, on the other hand, we generate its OL normal form that we then transform into a CNF. The two resulting formulas are sent to the same SAT solver and we compare the time it takes to decide. For the generated formula benchmark, we did the evaluation on a set of forty formulas of the shape $\langle 6, 3 \rangle$. Because we lack space, we will, in each benchmarks, show five of them randomly chosen, the remaining of the results can be found in the appendix.

Problem	Without NF_{OL}	With NF_{OL}	Speed-up
S1	187.8s	132.9s	0.41
S2	102s	76.4s	0.34
S3	118.8s	88.4s	0.34
S4	172.3s	180s	-0.04
S5	83.5s	98.3s	-0.15

Table 2: Time (in seconds) to decide for $\langle 6, 3 \rangle$-shaped formulas

Problem	Without NF_{OL}	With NF_{OL}	Speed-up
4pipe3	14.96s	10.41s	0.44
4pipe4	16.4s	11.13s	0.47
5pipe	25.88s	31.56s	-0.18
5pipe1	59.22s	32.77s	0.8
6pipe	354s	200s	0.77

Table 3: Time (in seconds) for CaDiCaL to decide

3 The equivalence problem

To take advantage of the proof search algorithm we will present later, one needs to find a class of problems that is hard for SAT solver to solve and that is easy to encode in the framework of orthologic. Let us go back to the circuits we used to benchmark the preprocessing method. An interesting question, that is hard for most of the solvers to answer to, is the question of equivalence between two circuits modulo variable renaming. First, let us provide the encoding of the problem into axioms. In the circuit, each intermediate variable x is equal to a formula ϕ. Let $\mathcal{A}$ be the set of axioms. For each of those intermediate variable, we add to $\mathcal{A}$ the axioms x^L, ϕ^R and x^R, ϕ^L. The example 16 illustrates the procedure. The remaining work is now to prove that this specific problem can effectively be encoded within the OL proof system with axioms.

Theorem 15. *Let F and G be two equivalent circuits modulo intermediate variable renaming. Furthermore, let z be the output of F and z' be the output of G and $\mathcal{A}$ the set of axioms. Then the sequent z^L, z'^R is valid in OL with axioms.*

Proof. We shall do a syntactic proof by induction on the structure of z.

- if z is a litteral, then it is straightforward.
- if $z = z_1 \wedge z_2$, we need the axioms $z^L, (z_1 \wedge z_2)^R$ and $(z'_1 \wedge z'_2)^L, z'^R$. Then, we first apply the cut rule,

$$\frac{z^L, (z_1 \wedge z_2)^R \qquad (z_1 \wedge z_2)^L, z'^R}{z^L, z'^R}\ \textsc{Cut}$$

  The first sequent is an axiom. To prove the second one, we apply again the cut rule, leading us to,

$$\frac{(z_1 \wedge z_2)^L, (z'_1 \wedge z'_2)^R \qquad (z'_1 \wedge z'_2)^L, z'^R}{(z_1 \wedge z_2)^L, z'^R}\ \textsc{Cut}$$

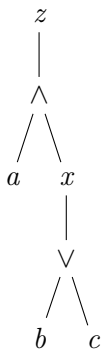
  Here, the second sequent is an axiom by hypothesis whereas the first one can be proven by applying first ($\wedge$-R), then successively ($\wedge$-L), giving us the two sequents $z_1^L, z_1'^R$ and $z_2^L, z_2'^R$ that admit a proof by induction hypothesis.

- if $z = z_1 \vee z_2$, we need the axioms $z^L, (z_1 \vee z_2)^R$ and $(z'_1 \vee z'_2)^L, z'^R$. Then, the reasonning is exactly the same until the second application of the cut rule, where it remains to prove the sequent $(z_1 \vee z_2)^L, (z'_1 \vee z'_2)^R$. Here, we apply first ($\vee$-$L$) and then successively ($\vee$-R), giving us the two sequents $z_1^L, z_1'^R$ and $z_2^L, z_2'^R$, both admitting a proof by induction hypothesis.

- if $z = \neg z_1$, we need the axioms $z^L, (\neg z_1)^R$ and $(\neg z'_1)^L, z'^R$. Then, we first apply the cut rule to introduce $\neg z_1$, producing the two sequents $z^L, (\neg z_1)^R$ and $(\neg z_1)^L, z'^R$. The first one is an axiom. To prove the second one, we apply, once again, the cut rule to introduce $\neg z'_1$, giving us the two sequents $(\neg z_1)^L, (\neg z'_1)^R$ and $(\neg z'_1)^L, z'^R$. The second one is an axiom. To prove the remaining one, we successively apply ($\neg$-L) and ($\neg$-R), leading to the sequent $z_1'^L, z_1^R$, that admits a proof by induction hypothesis.

Every axiom we use is in $\mathcal{A}$, therefore, we get that z^L, z'^R is valid in OL. □

Example 16. *Let us now illustrate the equivalence problem on a simple example. Suppose the following circuit.*



The formula represented is, in fact, $a \wedge (b \vee c)$. Therefore, the variables we rename to build the new circuit are z and x. One wants to prove either the sequent z^L, z'^R or z'^L, z^R in OL. Since we have the equalities ; $x = b \vee c$ (resp. $x' = b \vee c$) and $z = a \wedge x$ (resp. $z' = a \wedge x'$), we give the following axioms to our system ; $x^L, (b \vee c)^R$ and $(b \vee c)^L, x^R$ (resp. for x') and $z^L, (a \wedge x)^R$ and $(a \wedge x)^L, z^R$ (resp. for z'). According to theorem 15, the OL proof system with axioms is able to find a proof for the equivalence statement.

4 Proof search procedure for OL with axioms

In relation with the proof system 1, researchers have proposed a proof search algorithm for orthologic with axioms that has a cubical complexity in time [GK24]. The idea is to start from the goal sequent and to compute a backward search to eventually reach the

hypothesis and axioms by applying the rules of the proof system. However, the original formulation as a backward proof search was incorrect, as some edge cases related to *memoization* lead to incompleteness. To remedy this, the immediate idea is to divide the algorithm in two distinct parts. First, we build, backwardly, the structure of every sequent we might have to prove in order to prove the goal sequent, according to the rules of our proof system. Then, we explore the resulted structure forwardly, until we reach the goal sequent. Later in the section, we improve this algorithm by avoiding the first computation of every possible clause.

4.1 Backward-forward approach

As previously sketched, the idea of the algorithm we introduce can be decomposed in two parts such as what follows ;

1. First, it constructs backwardly the hypergraph of the proof space. The nodes of that hypergraph are the sequents and the hyperedges are the relations between the premises and the conclusion.

2. Second, it explores the resulted hypergraph forwardly *i.e.* starting from the axioms and targetting the goal. The way it does this is a variant of the classical breadth-first search algorithm.

To be clear on what data structure we are using, let us define the one that composes the core of the algorithm, that is the *directed hypergraph*.

Definition 17. *A directed hypergraph is a tuple $\langle V, E\rangle$ where V is a set of vertices and $E \subseteq \mathcal{P}(V) \times \mathcal{P}(V)$ is a set of subsets of pairs whose elements are subsets of V. The first component of such pair being the source nodes and the second component being the target nodes.*

In fact, in our specific case, the elements of E are essentially in $\mathcal{P}(V) \times V$ since there is only one sequent to conclude with. With that set, we propose the proof search algorithm 1 for orthologic. We shall now prove the correctness and study carefully the complexity of that algorithm. To that last, we will prove several intermediate lemmas that will help to prove the complexity theorem.

Lemma 18 ([GK24])**.** *Let A be a set of axioms and S be a sequent. The set of generated sequents from subformulas of sequents in A and S is of cardinal at most $4(||A|| + ||S||)^2$.*

Lemma 19 ([GK24])**.** *Let S be a sequent. In the OL sequent calculus, assuming that axioms are never hypothesis, there exists at most $7 + 4|A|$ instances of rules whose conclusion is S.*

Theorem 20. *The algorithm 1 is correct.*

Algorithm 1 Cubic time proof search algorithm for OL with axioms

(A, S) *the given problem*
$A^* \leftarrow \{a, b \mid (a^\circ, b^\diamond) \in A\}$
$V \leftarrow \{S\}$
$E \leftarrow \varnothing$
$W \leftarrow V$ *the set of vertices to visit*
while $W \neq \varnothing$ **do**
  $s = (\Gamma, \Delta) \leftarrow \text{choose}(W)$
  $W \leftarrow W \setminus \{s\}$
  if $(\Gamma, \Delta) = (\phi^L, \phi^R)$ || $(\Gamma, \Delta) \in A$ **then**
    $E \leftarrow E \cup \{\langle \varnothing, s\rangle\}$ *hyperedge for either an hypothesis or an axiom*
    continue
  end if
  $E \leftarrow E \cup \{\langle\{(\Gamma, \Gamma)\}, s\rangle\} \cup \{\langle\{(\Delta, \Delta)\}, s\rangle\}$ *hyperedges for the weakening rule*
  $W \leftarrow W \cup \{(\Gamma, \Gamma), (\Delta, \Delta)\}$
  if $\Delta = \neg\phi^\diamond$ **then**
    $E \leftarrow E \cup \{\langle\{(\Gamma, \phi^\diamond)\}, s\rangle\}$
    $W \leftarrow W \cup \{(\Gamma, \phi^\diamond)\}$
  else if $\Delta = (\phi \vee \psi)^L$ **then**
    $E \leftarrow E \cup \{\langle\{(\Gamma, \phi^L), (\Gamma, \psi^L)\}, s\rangle\}$
    $W \leftarrow W \cup \{(\Gamma, \phi^L), (\Gamma, \psi^L)\}$
  else if $\Delta = (\phi \wedge \psi)^R$ **then**
    $E \leftarrow E \cup \{\langle\{(\Gamma, \phi^R), (\Gamma, \psi^R)\}, s\rangle\}$
    $W \leftarrow W \cup \{(\Gamma, \phi^R), (\Gamma, \psi^R)\}$
  else if $\Delta = (\phi \vee \psi)^R$ **then**
    $E \leftarrow E \cup \{\langle\{(\Gamma, \phi^R)\}, s\rangle\} \cup \{\langle\{(\Gamma, \psi^R)\}, s\rangle\}$
    $W \leftarrow W \cup \{(\Gamma, \phi^R), (\Gamma, \psi^R)\}$
  else if $\Delta = (\phi \wedge \psi)^L$ **then**
    $E \leftarrow E \cup \{\langle\{(\Gamma, \phi^L)\}, s\rangle\} \cup \{\langle\{(\Gamma, \psi^L)\}, s\rangle\}$
    $W \leftarrow W \cup \{(\Gamma, \phi^L), (\Gamma, \psi^L)\}$
  end if
  the dual matching is applied to Γ
  $E \leftarrow E \cup \bigcup_{a \in A^*} \{\langle\{(\Gamma, a^R), (a^L, \Delta)\}, s\rangle, \langle\{(\Delta, a^R), (a^L, \Gamma)\}, s\rangle\}$
  $W \leftarrow W \cup \bigcup_{a \in A^*} \{(\Gamma, a^R), (a^L, \Delta), (\Delta, a^R), (a^L, \Gamma)\}$
  $V \leftarrow V \cup W$
end while
$H \leftarrow \langle V, E\rangle$ *the resulting hypergraph*
$P \leftarrow A$ *the set of proven sequents*
$R \leftarrow \varnothing$
while $P \neq \varnothing$ **do**
  $s \leftarrow \text{choose}(P)$
  $P \leftarrow P \setminus \{s\}$
  $R \leftarrow R \cup \{s\}$
  $E \leftarrow E[(K, d) \mapsto (K \setminus \{s\}, d)]$ *substitution within the set*
  $P \leftarrow P \cup \{t \in V \mid (\varnothing, t) \in E\}$
end while
return $S \in R$ *checks whether the goal* S *is proven or not*

Proof. In the beginning of the first loop, the sets W and V only contain the goal sequent and the edge set E is empty. At each iteration, we pop a sequent s from W and, for each rule, we add the hypothesis sequents to W and the hyperedges of those sequents to $s = (\Gamma, \Delta)$. For the hypothesis rule, we check whether s is equal to a sequent of the form ϕ^L, ϕ^R. If it is the case, then we add the hyperedge $\langle \varnothing, s \rangle$. For the remaining rules, one has to explicitely do the comparison with Γ and Δ. Let us focus on Δ with the ($\wedge$-R) rule ; the proof is the same for Γ and the other rules. If Γ is equal to $(\phi, \psi)^R$ then, according to the rule, one needs to prove Γ, ϕ^R and Γ, ψ^R, therefore, the algorithm adds the hyperedge $\langle \{(\Gamma, \phi^R), (\Gamma, \psi^R)\}, s \rangle$. Finally, to handle the cut rule, it enumerates every formula a in the set of formulas that belong to axioms A^* and adds the hyperedges going from Γ, a^R and a^L, Δ to s and reciprocally, from Δ, a^R and a^L, Γ to s. All those new sequents are added to W for the following iteration. According to lemma 18, we explore a bounded set of sequents and, since we visit them once, W is, at some point, decreasing until reaching emptiness. Let us now prove the correctness for the forward search. At the beginning of the loop, we have H the hypergraph structure and P the set of proven sequents to be explored, composed only of the axioms at the beginning. Furthermore, R is the set of all proven sequents so far. At each iteration, the algorithm pops a sequent s from P, adds it to R and alters the set of hyperedges E in the following way, for each hyperedge, it removes s from the source. If, after the operation, the source of a given hyperedge is empty, it means that the destination has been proven, therefore, it adds it to P. Since H is a finite structure, P is, at some point, decreasing until emptiness. By the end of the loop, R contains every provable sequents of H. Hence, the algorithm returns whether or not S, the goal sequent, belongs to R. □

Theorem 21. *The algorithm 1 runs in* $\mathcal{O}(n^3)$.

Proof. To begin, let us analyze the complexity of the first part of the algorithm. Let H be the hypergraph constructed by algorithm 1. By lemma 18, we assert that the cardinal of V is at most $4(||A|| + ||S||)^2$, that belongs to $\mathcal{O}(n^2)$. By lemma 19, we know that the number of hyperedges we need to create for each sequent is at most $7 + 4|A|$, that belongs to $\mathcal{O}(1 + |A|)$. Hence, the algorithm builds the hypergraph H in $\mathcal{O}((1 + |A|)n^2)$. Let us now analyze the complexity of the forward part. The set of proven sequents is of cardinal at most the cardinal of V. At each iteration, there is at most $\mathcal{O}(n)$ and, since the set of proven sequents is of cardinal at most the cardinal of V, we get a time complexity in $\mathcal{O}(n^3)$. Hence the final complexity. □

There is several optimizations to do and that are done in the OCaml implementation. We can split them in two distinct parts. The reduction of the search space and the acceleration of forward exploration of the hypergraph. Let us start with the former. When building the hypergraph backwardly, there is no need to add remaining hyperedges once the axiom or hypothese hyperedge has been added. Also, we can give the priority to revertible rules that is to say, once we found such, no need to add other

hyperedges. In the OL sequent calculus, invertible rules are ($\vee$-L) and ($\wedge$-R). This considerably reduces the size of the hypergraph but note that this may not be always the useful thing to do. Indeed, although it accelerates the backward constuction and the search, we could wonder whether there exists shorter proofs passing through those erased derivations. The second optimization is about the exploration of the hypergraph. To improve that, we mainly use memoization and the average constant time access provided by hash tables in average. Still, this is not enough to beat any SAT solver. The main problem lies in the use of the cut rule that generates a lot of irrelevant hyperedges. Thus, we spend $\mathcal{O}(|A|n^2)$ time in the first phase of the algorithm, even when a tiny fragment of those hyperedges are actually needed to complete the second phase.

4.2 Forward approach

Let us refine this strategy by using the hypergraph *implicitely* to find a path between the goal sequent and the axioms. To achieve that, we completely give up the backward approach to only proceed forwardly. The idea of the algorithm is the following, we shall emphasize its efficiency. Each time we add a sequent to the set of proven sequents, we want to deduce every new sequent according to the set of rules of the OL proof system. To do this efficiently, we are not allowed to do any kind of check while going through the set of proven sequents. Therefore, the data structure we were using until now is not sufficient and we shall introduce a new one. We can split the algorithm in three important parts, namely ; the cut rule, the rules with one premise and the rules with two premises.

Remark. *For the sake of legibility in what follows, we may write* (a, b) *for the sequent* a^L, b^R*. Furthermore, we will write* SF*, the set of subformulas of our problem and* AF*, the set of formulas that compose the axioms. By* P*, we denote the set of proven sequents.*

Suppose we just proved the sequent (a, b). For the cut rule, it is possible to efficiently deduce the new sequent quite easily since we only have to lookup among the already proven sequents. Therefore, we need two dual maps $\overrightarrow{P} = x \mapsto \{y \in SF \mid x^L, y^R\}$ and $\overleftarrow{P} = x \mapsto \{y \in SF \mid y^L, x^R\}$. Hence, to deduce the new sequent from (a, b), we can just add forall $x \in \overrightarrow{P}(b)$, (a, x) and, dually, forall $y \in \overleftarrow{P}(a)$, (y, b). Doing that, we nevertheless have an important issue that is that the algorithm is more likely to loop since it will, at some point, add a sequent that has already been proved. To remedy this, we put a check at the beginning of the function that will test whether the sequent we're trying to add hasn't been proved previously. Although, we can achieve that in constant time with a hash table, we still *check* the belonging for every sequent we encounter. Let us now have a look to the rules with only one premise, that is ($\wedge$-L) and ($\vee$-R). We analyze only the former, since the reasonning is dual for the other. We know (a, b) is proven, from that, we can deduce that $a^L, (b \vee c)^R$ and $(a \wedge c)^L, b^R$ for a given c and such that $b \vee c$ and $a \wedge c$ still belong to the set of subformulas SF of

our problem. To achieve that efficiently, we use two hash maps $A^\star$, with $\star \in \{\wedge, \vee\}$, that bind a formula x to the set of *superformulas* $\phi \in SF$ *i.e.* that are of the shape $\phi = x \star y$ or $\phi = y \star x$ for a given y. Those two dual maps are computable in linear time with respect to the cardinal of the set of subformulas of the given problem. Finally, it remains to handle the case of the rules admitting two premises, namely, ($\vee$-L) and ($\wedge$-R). Let us analyze only the left disjunction case, since the right conjunction case is entirely symmetric. Roughly speaking, the difficulty here is to maintain the intersection between the set of subformulas SF and the set of formulas appearing in the left part of the proven sequents. For instance, suppose we have proved (a, b), we would like to deduce every $(a \vee c)^L, b^R$ such that $a \vee c \in SF$. To achieve that, we obviously need $(c, b) \in P$ which, with the naive method, forces us to go through either SF or P and to do some useless testing. To do this efficiently, we need to ensure that every element we encounter in our data structure will allow to produce a new proven sequent. The data structure we introduce takes advantage of the element the three sequents of the rule (the two premises and the conclusion) have in common. In our case, it is the right part, that is to say b. First, let us consider $B^\vee$ (resp. $B^\wedge$) a hash map that binds to every formula x another hash map that itself binds to every formula y a non-empty set $\{x \vee y, y \vee x\} \subseteq SF$. It is clear that those sets have cardinal at most 2 depending on whether the formulas are in SF or not. We now need the data structure that will keep in memory that *intersection* between the proven sequents and the subformulas we sketched before. We shall call it D (resp. C) ; it is again a hash map that binds every formula x to another hash map that itself binds every formula y to a set of formulas that is of the shape $\{y \vee z, z \vee y \in SF \mid \forall z, (z, x) \in P\}$. To make things we just said clearer, let us sum up the types of each structure ;

$$\begin{aligned} P &: \mathcal{P}(SF \times SF) \\ A^\star, \overrightarrow{P}, \overleftarrow{P} &: SF \to \mathcal{P}(SF) \\ B^\star, C, D &: SF \to SF \to \mathcal{P}(SF) \end{aligned}$$

where $\star \in \{\wedge, \vee\}$. The algorithm 2 exhibits the pseudo-code of everything that has been said.

Theorem 22. *The algorithm 2 is correct.*

Proof. Before the first call to the function add, the maps $A^\star$ and $B^\star$ are built and are not meant to be changed afterwards. The map C (resp. D) is empty before the first call to add. Each time a call to add is performed with the sequent (a, b) as argument, if the sequent is the goal we seek, then the algorithm returns true, making itself stop. If the sequent already belongs to P, it means that the algorithm already deduced sequents from it, therefore it returns false. Otherwise, it first updates the structures. To P, it adds the sequent, therefore, at each iteration, every sequent belonging to P is valid. To

Algorithm 2 Cubical forward proof search algorithm for OL with axioms

Let SF be the set of the subformulas of a given problem $(\mathcal{A}, S)$ in NNF
Let $AF \subseteq SF$ be the set of subformulas of the axioms
$P, \overrightarrow{P}, \overleftarrow{P} \leftarrow \varnothing$
$A^\star \leftarrow x \mapsto \{\phi \in SF \mid \exists y, \phi = x \star y \text{ or } \phi = y \star x\}$
$B^\star \leftarrow x \mapsto y \mapsto \{x \star y, y \star x\} \in \mathcal{P}(SF) \setminus \varnothing$
$C \leftarrow \varnothing$
$D \leftarrow \varnothing$
procedure ADD(a, b)
  if $(a, b) \in P$ **then**
    return false
  end if
  if $(a, b) = S$ **then**
    return true
  end if
  $P \leftarrow P \cup \{(a, b)\}$
  $\overrightarrow{P}(a) \leftarrow \overrightarrow{P}(a) \cup \{b\}$
  $\overleftarrow{P}(b) \leftarrow \overleftarrow{P}(b) \cup \{a\}$
  for $k \in keys(B^\vee(a))$ **do**
    $D(b)(k) \leftarrow D(b)(k) \cup B^\vee(a)(k)$
  end for
  for $k \in keys(B^\wedge(b))$ **do**
    $C(a)(k) \leftarrow C(a)(k) \cup B^\wedge(b)(k)$
  end for
  $res \leftarrow$ **false**
  for $\phi \in A^\wedge(a) \cup D(b)(a) \cup \overleftarrow{P}(a)$ **do**
    $res \leftarrow res \mid\mid ADD(\phi, b)$
  end for
  for $\phi \in A^\vee(b) \cup C(a)(b) \cup \overrightarrow{P}(b)$ **do**
    $res \leftarrow res \mid\mid ADD(a, \phi)$
  end for
  return res
end procedure
for $(a_1, a_2) \in \mathcal{A}$ **do**
  $ADD(a_2, a_2)$
end for

$\overrightarrow{P}(a)$ it adds b, therefore, at each iteration, forall $x \in SF$ and $y \in \overrightarrow{P}(x)$ the sequent (x, y) is valid. Dually for $\overleftarrow{P}$. It updates C (resp. D) by doing the union of conjunctive (resp. disjunctive) formulas of the form $b \wedge k$ or $k \wedge b$ (resp. $a \vee k$ or $k \vee a$) that may be proven if, at some point, (a, k) (resp. (b, k)) is proven. Here, the k lie in the set of keys of $B^{\wedge}(b)$, therefore it covers all the subformulas. After having updated C (resp. D), the algorithm will deduce new sequents. There is two kind of sequents we can deduce, (ϕ, b) and (a, ϕ) where $\phi \in SF$. Let us handle both of the cases.

1. for the case (ϕ, b), the ϕ formulas belong to the union of $A^{\wedge}(a)$, $D(b)(a)$ and $\overleftarrow{P}(a)$. At this stage of the algorithm, $A^{\wedge}(a)$, as we said before, is not altered and therefore, it is still of the shape $\{\phi \in SF \mid \exists y, \phi = a \wedge y \text{ or } \phi = y \wedge a\}$. Hence, adding the sequent (ϕ, b) to the set of proven ones is correct. Furthermore, at that point of the algorithm, $D(b)(a)$ contains exactly the following intersection set, $\{a \vee x \in SF \mid (x, b) \in P\}$. Therefore, since (a, b) and (x, b) are proven, adding $(a \vee x, b)$ is correct. Finally, $\overleftarrow{P}(a)$ contains every ϕ such that (ϕ, a) is proven and, since we just added (a, b), adding (ϕ, b) is correct. However, (ϕ, b) might have already been proven ; the check, to avoid cycles, is done in the beginning of the function add. Thus, there is no cycles. Therefore, adding (ϕ, b) doesn't create any cycle and is correct.

2. for the case (a, ϕ), the reasonning is completely dual.

Now we have proven the correctness of the add procedure, it remains to prove the correctness of the main procedure. That last calls add passing it, as arguments, the sequents in $\mathcal{A}$, that is to say, the axioms of the given problem. Hence, the algorithm is correct with respect to the OL proof system. □

Theorem 23. *The algorithm 2 has a time complexity in* $\mathcal{O}(n^3)$.

Proof. Let us start by analyzing the complexity of the procedure add. At each of its call with a given sequent (a, b), there is two checks, whether the sequent belongs to the already proven sequents and whether it is the sequent we seek. The first check is done in logarithmic time or in constant time, depending on the structure we use. The second check is done is constant time. Then, the function updates the structures P, $\overrightarrow{P}$, $\overleftarrow{P}$, C and D. The update of the first three structures is done in constant time because P is a set and we access $\overleftarrow{P}(a)$ and $\overrightarrow{P}(b)$ in constant time. Let us now have a closer look to the update of D, since the update of C is dual. The algorithm enumerates the keys of the map $B^{\vee}(a)$, that we get in constant time, since $B^{\vee}$ is a hash map. The cardinal of that set of keys is at most $|SF|$ since $B^{\vee}(a)(k)$ is a subset of SF. Therefore, the update of D is done in $\mathcal{O}(n)$. Next, the function deduces the new sequents. Let us get a closer look to the sequents of the shape (ϕ, b) since, for the case (a, ϕ), it is the same reasonning. The algorithm enumerates the formulas in the union of $A^{\wedge}(a)$, $D(b)(a)$ and $\overleftarrow{P}(a)$. We have that $|A^{\wedge}(a)| \leq |SF|$ because it is a subset of SF. Furthermore, we

also have $|D(b)(a)| \leq |SF|$ because it is the union of sets being subsets of SF. Finally, $|\overleftarrow{P}(a)| \leq |SF|$ because a can be proven by at most all the subformulas of S. Thus, enumerating the formulas in the union of those three sets takes time in $\mathcal{O}(n)$. Because of the first guard, checking whether the sequent is proven or not, the function add is called at most the number of possible sequents of the shape a^L, b^R where $a, b \in SF$. Thus, there is at most $|SF|^2$ call to add. In overall, the algorithm has a worst case time complexity in $\mathcal{O}(n^3)$.

□

Remark. *Let us take time to write several remarks on our OCaml implementation. First, since the recursion call may be deep, to avoid any stack overflow, the algorithm has been written tail-recursively. Second, pay a special attention to the data structure B. Indeed, it supposes that the operators $\wedge$ and $\vee$ are of arity exactly two. Therefore, one cannot encode the formulas in an ADT that, for instance, would type the conjunction constructor such as $And : formula\ list \to formula$. Doing that, one would need to filter the formulas having arity exactly 2 to add them to the data structure B, which results in an unsound algorithm, according to the OL proof system.*

4.3 Evaluation

Now that we have a working efficient algorithm, we are ready to compare it to SAT solvers on the class of problems of circuit equivalence checking, modulo variable renaming, presented in section 3. Among the benchmarks we dispose, a circuit that appears to be particularly hard is the EPFL's circuit of multiplication operation (multiplier.aig). The testing procedure is the following. First, we parse the given AIGER file and we store the data in a convenient algebraic data type. Then, from this data, we build the circuit equivalence by renaming the intermediate variables (the ones that name the subformulas). From the resulting formula, we compute, on one hand, the conjunctive normal form of its negation, applying the usual Tseitin's transformation and, on the other hand, we encode it into OL with axioms. Finally, we can give that encoding to our algorithm, so it tries to prove it. On the other side, we run a SAT solver on the produced CNF. Note that, if it terminates, it must yield unsatisfiability, because the equivalence formula is valid. Otherwise, it means that the encoding is wrong. In our case, we expect the solver to yield unsat and our algorithm to find a proof. The table 4 shows the time taken by our algorithm (OLSC) *versus* the time taken by the CryptoMinisat [SNC09], the CaDiCaL [BFF+24a] and the Kissat [BFF+24b] solvers to decide the equivalence problem on several instances of the multiplier circuit. Depending on the instance of the multiplier circuit, OLSC performs 20 to 40 times faster than CryptoMinisat and 7 to 17 time faster than CaDiCaL. Concerning Kissat, OLSC is slower on the four first instances and has a speed up from 5 to 40 percent on the two last. Evaluation has been made on the same instances, randomly changing a variable renaming, making them wrong *i.e.* the SAT solvers find an assignation and OLSC is

	OLSC	CryptoMinisat	CaDiCaL	Kissat
mult 12 bits	0.8s	37.3s	5.7s	0.4s
mult 16 bits	1.6s	97.5s	18.2s	0.9s
mult 20 bits	3.1s	165.7s	30.9s	1.5s
mult 24 bits	4.4s	294.4s	56.5s	4s
mult 32 bits	10.6s	> 500s	117.3s	11.2s
mult 40 bits	19.7s	> 500s	327.2s	27.4s

Table 4: Time (in seconds) to solve the equivalence problem, comparing our algorithm with different SAT solvers

not able to find any proof.

5 Going further

The obtained results lead us to consider two main improvements on SAT solvers in order to speed-up the decision procedure for some class of problems. The first improvement concerns the preprocessing. Before converting any formula to conjunctive normal form, it might be useful to transform it first to the OL normal form. Depending on the shape of the formula, the decision procedure is faster. The question is now to identify the species on which the OL normalization will certainly lead to a net speed-up of the SAT solver. The second improvement naturally concerns the proof algorithm itself. For some class of hard problems, one can show that it can be encoded in the OL framework and, hence, provable by our proof search algorithm. Therefore, it would be interesting for SAT solvers to provide an interface that allows one to specify the encoding of a given class of problems into the OL framework. Our algorithm would then be called when the solver has to deal with such class of problems. There is also improvements to do on the algorithm 2 itself. The main problem is the data structure $B^\star$ because it enforces us not to use an ADT able to flatten the formula, leading to stack overflows of the data structure when the formula is too big. Therefore, we would rather need to type $B^\star$ with SF list $\to \mathcal{P}(SF)$. This maps an arbitrary list of subformulas to the set of every existing permutation that lies in SF, according to the operator $\star$.

6 Conclusion

Along this work, we investigated several ways to improve the performance of SAT solvers using some properties of *orthologic*. On one hand, we have shown that preprocessing the formulas using the OL normal form could, in average, speed up the solving. On the other hand, we provided an efficient proof search algorithm to prove statements within the OL proof system.

References

[AGDM15] Luca Amarù, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. The epfl combinational benchmark suite. 2015.

[BFF+24a] Armin Biere, Tobias Faller, Katalin Fazekas, Mathias Fleury, Nils Froleyks, and Florian Pollitt. CaDiCaL 2.0. In Arie Gurfinkel and Vijay Ganesh, editors, *Computer Aided Verification - 36th International Conference, CAV 2024, Montreal, QC, Canada, July 24-27, 2024, Proceedings, Part I*, volume 14681 of *Lecture Notes in Computer Science*, pages 133–152. Springer, 2024.

[BFF+24b] Armin Biere, Tobias Faller, Katalin Fazekas, Mathias Fleury, Nils Froleyks, and Florian Pollitt. CaDiCaL, Gimsatul, IsaSAT and Kissat entering the SAT Competition 2024. In Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors, *Proc. of SAT Competition 2024 – Solver, Benchmark and Proof Checker Descriptions*, volume B-2024-1 of *Department of Computer Science Report Series B*, pages 8–10. University of Helsinki, 2024.

[BN36] Garrett Birkhoff and John Von Neumann. The logic of quantum mechanics. *Annals of Mathematics*, 37(4):823–843, 1936.

[Bru76] Günter Bruns. Free ortholattices. *Canadian Journal of Mathematics*, 28(5):977–985, 1976.

[DP60] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *J. ACM*, 7(3):201–215, July 1960.

[GBMK23] Simon Guilloud, Mario Bucev, Dragana Milovančević, and Viktor Kunčak. Formula normalizations in verification. In Constantin Enea and Akash Lal, editors, *Computer Aided Verification*, pages 398–422, Cham, 2023. Springer Nature Switzerland.

[Gen35] Gerhard Gentzen. Untersuchungen über das logische schließen. i. *Mathematische Zeitschrift*, 39:176–210, 1935.

[GK24] Simon Guilloud and Viktor Kunčak. Orthologic with axioms. *Proc. ACM Program. Lang.*, 8(POPL), January 2024.

[Gol74] R. I. Goldblatt. Semantic analysis of orthologic. *Journal of Philosophical Logic*, 3(1/2):19–35, 1974.

[JC09] Himanshu Jain and Edmund M. Clarke. Efficient sat solving for non-clausal formulas using dpll, graphs, and watched cuts. In *2009 46th ACM/IEEE Design Automation Conference*, pages 563–568, 2009.

[Lau17] Olivier Laurent. Focusing in orthologic. *Logical Methods in Computer Science*, 13(3):6, July 2017.

[McC98] William McCune. Automatic proofs and counterexamples for some ortholattice identities. *Information Processing Letters*, 65(6):285–291, 1998.

[NV05] Juan A. Navarro and Andrei Voronkov. Generation of hard non-clausal random satisfiability problems. In *Proceedings of the National Conference on Artificial Intelligence/Proc Natl Conf Artif Intell*, volume 1, pages 436–442, United States, 2005. AAAI Press. 20th National Conference on Artificial Intelligence and the 17th Innovative Applications of Artificial Intelligence Conference, AAAI-05/IAAI-05 ; Conference date: 01-07-2005.

[RF95] J. B. Nation R. Freese, J. Jezek. *Free Lattices*. Mathematical Surveys and Monographs, 1995.

[SNC09] Mate Soos, Karsten Nohl, and Claude Castelluccia. Extending sat solvers to cryptographic problems. In Oliver Kullmann, editor, *Theory and Applications of Satisfiability Testing - SAT 2009*, pages 244–257, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[Tse83] G. S. Tseitin. *On the Complexity of Derivation in Propositional Calculus*, pages 466–483. Springer Berlin Heidelberg, Berlin, Heidelberg, 1983.

[Vel01] M. N. Velev. Fvp-unsat 2.0 benchmark suite, available from: http://www.ece.cmu.edu/ mvelev. 2001.

[vNB18] John von Neumann and ROBERT T. BEYER. *Mathematical Foundations of Quantum Mechanics: New Edition*. Princeton University Press, ned - new edition edition, 2018.

[Whi41] Philip M. Whitman. Free lattices. *Annals of Mathematics*, 42(1):325–330, 1941.