Efficient and Predictable tools

with

Orthologic-Based Reasoningc

Simon Guilloud

EPFL

Lausanne, Switzerland

{firstname.lastname}@epfl.ch

Verification is hard

Verification often faces undecidable problems, or NP-Hard.

- Heuristics methods are immensely useful in practice, but offer few guarantees.

Verification is hard

Verification often faces undecidable problems, or NP-Hard.

- Heuristics methods are immensely useful in practice, but offer few guarantees.
  - Instability between versions
  - Non-determinism
  - Bugs hard to reproduce
  - Trial and Error for the user

Example: Type Checking

Sometimes, reliability is clearly more important than completeness (or expressivity)

Example: Type Checking

Sometimes, reliability is clearly more important than completeness (or expressivity)

Type checking:
Typeclass resolution, subtyping with union/disjunction types, liquid/refinement types, proving disjointness, etc.

Example: Type Checking

Sometimes, reliability is clearly more important than completeness (or expressivity)

Type checking:
Typeclass resolution, subtyping with union/disjunction types, liquid/refinement types, proving disjointness, etc.

- Not logically complete
- As expressive as possible
- Same behaviour in all contexts and machines
- Reasonably fast
- Compatible with other elements of the compiler/type system

Efficient and Predictable building blocks

Efficient and Predicatable building blocks
for verification tools

- Incomplete, but...
- Clear completeness guarantees
- Efficient ($\approx$ polynomial)
- Combines with other approaches
- Reliable, Reusable, Modular

Orthologic

One particularly important domain: classical **propositional logic**

- Validity and Satisfiability are (co)NP-complete

"Is a given formula ϕ true?"

Orthologic

One particularly important domain: classical **propositional logic**

- Validity and Satisfiability are (co)NP-complete

  "Is a given formula ϕ true?"

- Most interesting problems are computationally hard (interpolation, unification modulo, ...)

Orthologic

- Can we obtain efficient, predictable algorithms for well-characterized weakening of classical propositional logic?
- What about intuitionistic logic? Not better: deciding validity is PSPACE-complete.

Orthologic

- Can we obtain efficient, predictable algorithms for well-characterized weakening of classical propositional logic?
- What about intuitionistic logic? Not better: deciding validity is PSPACE-complete.
- Other Possibility: Orthologic

Ortholattices

- The propositional logic whose structure is that of Ortholattices
- $\wedge, \vee, \neg$

Classical Logic	Boolean Algebras
Intuitionistic Logic	Heyting Algebras
Orthologic	**Ortholattices**

Orthologic

Commutativity	$x \vee y = y \vee x$	$x \wedge y = y \wedge x$
Associativity	$x \vee (y \vee z) = (x \vee y) \vee z$	$x \wedge (y \wedge z) = (x \wedge y) \wedge z$
Idempotence	$x \vee x = x$	$x \wedge x = x$
Constants laws	$x \vee 1 = 1$	$x \wedge 0 = 0$
Double negation	$\neg\neg x = x$	
Excluded middle	$x \vee \neg x = 1$	$x \wedge \neg x = 0$
De Morgan's law	$\neg(x \vee y) = \neg x \wedge \neg y$	$\neg(x \wedge y) = \neg x \vee \neg y$
Absorption	$x \vee (x \wedge y) = x$	$x \wedge (x \vee y) = x$

Orthologic

Commutativity	$x \vee y = y \vee x$	$x \wedge y = y \wedge x$
Associativity	$x \vee (y \vee z) = (x \vee y) \vee z$	$x \wedge (y \wedge z) = (x \wedge y) \wedge z$
Idempotence	$x \vee x = x$	$x \wedge x = x$
Constants laws	$x \vee 1 = 1$	$x \wedge 0 = 0$
Double negation	$\neg\neg x = x$	
Excluded middle	$x \vee \neg x = 1$	$x \wedge \neg x = 0$
De Morgan's law	$\neg(x \vee y) = \neg x \wedge \neg y$	$\neg(x \wedge y) = \neg x \vee \neg y$
Absorption	$x \vee (x \wedge y) = x$	$x \wedge (x \vee y) = x$

- Boolean Algebra = Ortholattice + distributivity

  Distributivity: $x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$

Example

In orthologic, given

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

Does

$$(\neg c \vee b) \vee (\neg b \wedge (c \vee \neg a))$$

hold?

Example

In orthologic, given

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

Does

$$(\neg c \vee b) \vee (\neg b \wedge (c \vee \neg a))$$

hold?
Yes (and hence so does it in classical logic)

Why is it interesting?

Orthologic has good properties:

- $\mathcal{O}(n^2)$ normalization algorithm[1]

[1] Guilloud, Bucev, Milovančević, Kunčak. Formula Normalizations in Verification. CAV 2023.

Orthologic Normal Form

Definition

Let $\mathcal{T}$ be the set of terms over $(\wedge, \vee, \neg, 0, 1)$.
$f : \mathcal{T} \to \mathcal{T}$ is a normal form function if

$$\forall w_1, w_2 \in \mathcal{T}, w_1 \sim w_2 \iff f(w_1) = f(w_2)$$

Theorem

There exists a normal form function for OL computable in $\mathcal{O}(n^2)$.
Moreover, it computes a term of smallest size.

Orthologic Normal Form

Example:

$$[\neg(a \wedge \neg b) \wedge (\neg a \vee c)] \vee b \rightsquigarrow \neg a \vee b$$

- Fully compatible with structure sharing
- Never increases size
- Normal form is equivalent, not just equisatisfiable

Orthologic Normal Form

Example:

$$[\neg(a \wedge \neg b) \wedge (\neg a \vee c)] \vee b \rightsquigarrow \neg a \vee b$$

- Fully compatible with structure sharing
- Never increases size
- Normal form is equivalent, not just equisatisfiable

Efficient, predictable, modular building block

Why is it interesting?

Orthologic has good properties:

- $\mathcal{O}(n^2)$ normalization algorithm[3]
- Proof system with $\mathcal{O}(n^3)$ proof search with non-logical axioms ($\mathcal{O}(n^2)$ without axioms)[4]

[3] Guilloud, Bucev, Milovančević, Kunčak. Formula Normalizations in Verification. CAV 2023.
[4] Guilloud, Kunčak. Orthologic with Axioms. POPL 2024.

Proof System for Orthologic

Sequent-Calculus-like:

$$\phi^L, \psi^R \text{ provable} \iff \phi \leq \psi \text{ valid in all ortholattices}$$

Classical Logic	Sequent Calculus LK
Intuitionistic Logic	max. one formula on the right
Orthologic	**max. two formulas total**

Proof System for Orthologic

Sequent Calculus style proof system:

$$\frac{}{\phi^L, \phi^R}\ \text{Hyp}$$

$$\frac{\Gamma, \psi^R \qquad \psi^L, \Delta}{\Gamma, \Delta}\ \text{Cut} \qquad \frac{\Gamma}{\Gamma, \Delta}\ \text{Weaken}$$

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

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

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

$$\frac{}{\Gamma, \Delta}\ \text{Ax}(\Gamma,\Delta) \qquad \text{If } \Gamma, \Delta \text{ is an axiom}$$

Γ and Δ are arbitrary annotated formula, or no formula.

Proof System for Orthologic

Adding axioms makes Orthologic more expressive

- Reasoning within a body of knowledge
- Unlike classical logic, we can't put axiom directly in the formula
- Allowing axioms allows stating and prove more things

Proof System for Orthologic

Adding axioms makes Orthologic more expressive

- Reasoning within a body of knowledge
- Unlike classical logic, we can't put axiom directly in the formula
- Allowing axioms allows stating and prove more things

Example: set of known facts of classical logic, asserted by a solver

Cut elimination

Let A be a set of axioms:

Theorem

If a sequent S is provable, it has a proof where the only cut formulas are among the axioms in A, i.e. $\psi \in A$.

$$\frac{\Gamma, \psi^R \qquad \psi^L, \Delta}{\Gamma, \Delta}\ \text{Cut}$$

Cut elimination

Let A be a set of axioms:

Theorem

If a sequent S is provable, it has a proof where the only cut formulas are among the axioms in A, i.e. $\psi \in A$.

$$\frac{\Gamma, \psi^R \qquad \psi^L, \Delta}{\Gamma, \Delta}\ \text{Cut}$$

Corollary

The proof system enjoy the *Subformula Property*: If a sequent S is provable, it has a proof where only subformulas of S and axioms in A appear.

Efficient Proof Search

The Subformula property lets us devise an efficient proof search algorithm:

Algorithm: Proof Search for OL with Axioms

```
1 def prove(Γ, Δ)
2     Find all rules that can conclude with Γ, Δ
3     Recursively solve the m smaller formulas
4     Memoize intermediate results
```

Efficient Proof Search

The Subformula property lets us devise an efficient proof search algorithm:

Algorithm: Proof Search for OL with Axioms

```
1 def prove(Γ, Δ)
2     Find all rules that can conclude with Γ, Δ
3     Recursively solve the m smaller formulas
4     Memoize intermediate results
```

Let n be the size of the input (axioms + goal):

- at most $\mathcal{O}(n^2)$ different inputs

Efficient Proof Search

The Subformula property lets us devise an efficient proof search algorithm:

Algorithm: Proof Search for OL with Axioms

```
1 def prove(Γ, Δ)
2     Find all rules that can conclude with Γ, Δ
3     Recursively solve the m smaller formulas
4     Memoize intermediate results
```

Let n be the size of the input (axioms + goal):

- at most $\mathcal{O}(n^2)$ different inputs
- $m = \mathcal{O}(1 + |A|)$

Efficient Proof Search

The Subformula property lets us devise an efficient proof search algorithm:

Algorithm: Proof Search for OL with Axioms

```
1 def prove(Γ, Δ)
2     Find all rules that can conclude with Γ, Δ
3     Recursively solve the m smaller formulas
4     Memoize intermediate results
```

Let n be the size of the input (axioms + goal):

- at most $\mathcal{O}(n^2)$ different inputs
- $m = \mathcal{O}(1 + |A|)$
- Running time: $\mathcal{O}(n^2 \cdot (1 + |A|))$

Why is it interesting?

Orthologic has good properties:

- $\mathcal{O}(n^2)$ normalization algorithm[3]
- Proof system with $\mathcal{O}(n^3)$ proof search with non-logical axioms ($\mathcal{O}(n^2)$ without axioms)[2]
- Classicaly complete for important classes of formulas

[3] Guilloud, Bucev, Milovančević, Kunčak. Formula Normalizations in Verification. CAV 2023.
[2] Guilloud, Kunčak. Orthologic with Axioms. POPL 2024.

Classical completeness

OL with axioms is complete for Horn clauses and extensions of Horn clauses

$$\text{Horn clause } \{\neg a_1, ..., \neg a_n, b\} \text{ becomes } (a_1 \wedge ... \wedge a_n)^L, b^R$$

Theorem

A set of Horn clauses is satisfiable in OL if and only if it is satisfiable in CL.

Also true for renamed Horn, extended Horn and 2SAT

Predicate Orthologic

- Propositional Orthologic can be extended to Predicate Orthologic (with axioms)
- Of interest: Effectively Propositional Orthologic (i.e. predicates, constants and variables but no functions nor quantifiers)

Predicate Orthologic

- Propositional Orthologic can be extended to Predicate Orthologic (with axioms)
- Of interest: Effectively Propositional Orthologic (i.e. predicates, constants and variables but no functions nor quantifiers)
- Complete for Horn Clauses $\implies$ Extension of Datalog

Why is it interesting?

Orthologic has good properties:

- $\mathcal{O}(n^2)$ normalization algorithm[3]
- Proof system with $\mathcal{O}(n^3)$ proof search with non-logical axioms ($\mathcal{O}(n^2)$ without axioms)[3]
- Classicaly complete for important classes of formulas
- Has interpolation property[4]
- Other useful and interesting logical properties.

[3] Guilloud, Bucev, Milovančević, Kunčak. Formula Normalizations in Verification. CAV 2023.
[3] Guilloud, Kunčak. Orthologic with Axioms. POPL 2024.
[4] Guilloud, Gambhir, Kunčak. Interpolation and Quantifiers in Ortholattices. VMCAI 2024

Interpolation

Theorem

Let A and B be propositional formulas. If $A \vdash B$ then there exists a formula I such that:

$$A \leq I \leq B$$

and $FV(I) \subseteq FV(A) \cap FV(B)$.

Proof.

Show it for sequents. By induction on the proof of A^L, B^R □

Additional Properties

- OL admits a Tseitin-like normal form
- OL can be simulated by width 5 resolution
- More properties of Effectively Propositional OL

Coq formalization

Formalized OL proof system (without axioms) in Coq[5].

- Mechanized Cut Elimination
- Soundness of orthologic proof search, with memoization and reference equality
- Tactic (using reflection) for OL equivalence and normalization, including for the bool type.

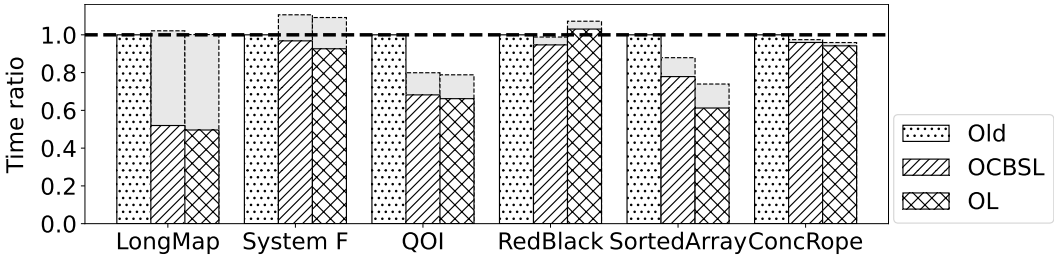
[5] Guilloud, Pit-Claudel. Verified and Optimized Implementation of Orthologic Proof Search. Preprint.

Stainless: Program Verification using OL

Stainless is a tool for verification of Scala programs.

- Generates Verification Conditions (VC) that are then submitted to SMT solvers
- VCs are simplified and cached with respect to orthologic.

Stainless: Program Verification using OL



- The grey-filled boxes represent the time saved thanks to extra caching.
- Simplification occasionally made the solvers' life harder (hand tuned assertions).
- OCBSL = Orthocomplemented Bisemilattices[6]

[6] Guilloud, Kunčak. Equivalence Checking for Orthocomplemented Bisemilattices in Log-Linear Time. TACAS 2022.

Lisa's Equivalence Checker

LISA's Kernel contains an algorithm to decide:

Given two formulas ϕ and ψ, does $\phi \sim_{OL} \psi$ hold?

Lisa's Equivalence Checker

LISA's Kernel contains an algorithm to decide:

Given two formulas ϕ and ψ, does $\phi \sim_{OL} \psi$ hold?

- Worst case $\mathcal{O}(n^2)$ time
- Also alpha-equivalence, symmetry and reflexivity of equality...

Lisa's Equivalence Checker

LISA's Kernel contains an algorithm to decide:

Given two formulas ϕ and ψ, does $\phi \sim_{OL} \psi$ hold?

- Worst case $\mathcal{O}(n^2)$ time
- Also alpha-equivalence, symmetry and reflexivity of equality...
- Proof Checker uses it instead of syntactic equality.

Other example:

```
1    assume( (a \/ b) /\ ( a \/ c) \/ b )
2    have( a \/ b ) by Restate
```

application: DPLL-like Propositional Solver

For a formula f:

- Simplify f
- If it is $\top$ returns **true**. If it is $\bot$, returns **false**
- Pick a literal a in f
- Solve recursively $f[a := \top]$ and $f[a := \bot]$

Idead: simplify with Orthologic.

Propositional Solver tactic

```
1  def dpll( f: Formula) =
2      val f = reducedForm(_f)      //computes OL—normal form
3      if f == ⊤ then have(f) by Hypothesis
4      else if f == ⊥ then fail("Not-a-tautology")
5      else
6          val a = findBestAtom(f)
7          val step1 = subproof :    //solve recursively
8              have(dpll(f(a := ⊤)))
9              thenHave(a |— f) by Substitution(⊤ ⟺ a)
10         val step2 = subproof :    //solve recursively
11             have(dpll(f(a := ⊥)))
12             thenHave(!a |— f) by Substitution(⊥ ⟺ a)
13         have(f) by Cut(step1, step2)
```

Orthologic Type System

- Type system with subtyping ($<:$), union ($|$), intersection types ($\&$) : **lattice**
- Intuitively, $t : T_1 | T_2$ iff $t : T_1$ or $t : T_2$

Orthologic Type System

- Type system with subtyping ($<:$), union ($|$), intersection types ($\&$) : **lattice**
- Intuitively, $t : T_1 | T_2$ iff $t : T_1$ or $t : T_2$
- Examples: Scala, Flow, TypeScript

Orthologic Type System

- Type system with subtyping ($<:$), union ($|$), intersection types ($\&$) : **lattice**
- Intuitively, $t : T_1 | T_2$ iff $t : T_1$ or $t : T_2$
- Examples: Scala, Flow, TypeScript
- If we add negation types: Ortholattice
- Intuitively, $t : \neg T$ iff not $t : T_1$

Orthologic Type System

- Type system with subtyping ($<:$), union ($|$), intersection types ($\&$) : **lattice**
- Intuitively, $t : T_1 | T_2$ iff $t : T_1$ or $t : T_2$
- Examples: Scala, Flow, TypeScript
- If we add negation types: Ortholattice
- Intuitively, $t : \neg T$ iff not $t : T_1$
- Idea: decide $A <: B$ by checking $A \vdash B$ in orthologic!

Orthologic Type System

- We also want to support **type constructors**, such as List$[T]$ or arrow types, $A \Rightarrow B$

Orthologic Type System

- We also want to support **type constructors**, such as List$[T]$ or arrow types, $A \Rightarrow B$
- Some are covariant or contravariant:

$$A <: B \longrightarrow \text{List}[A] <: \text{List}[B]$$

Orthologic Type System

- We also want to support **type constructors**, such as List$[T]$ or arrow types, $A \Rightarrow B$
- Some are covariant or contravariant:

$$A <: B \longrightarrow \text{List}[A] <: \text{List}[B]$$

Luckily, OL normalization and proof search with axioms can be extended to support (anti)monotonic functions and still work ($\mathcal{O}(n^2) \cdot |A|$)

Orthologic Type System

Many common and less common constructs can be encoded in such system:

- Inheritence relations become classes

Orthologic Type System

Many common and less common constructs can be encoded in such system:

- Inheritence relations become classes
- Bounded polymorphism

```
def foo[T<: Int](x: T): T = ...
```

Orthologic Type System

Many common and less common constructs can be encoded in such system:

- Inheritence relations become classes
- Bounded polymorphism

$$\textbf{def}\ \text{foo}[\text{T} <: \text{Int}](\text{x}: \text{T}): \text{T} = \ldots$$

- Types of things that are not **null**, things that are not functions, ...

Orthologic Type System

Many common and less common constructs can be encoded in such system:

- Inheritence relations become classes
- Bounded polymorphism

$$\textbf{def}\ \text{foo}[\text{T} <: \text{Int}](\text{x}: \text{T}): \text{T} = \ldots$$

- Types of things that are not **null**, things that are not functions, ...
- Record types with depth, width and permutation subtyping

Orthologic Type System

Many common and less common constructs can be encoded in such system:

- Inheritence relations become classes
- Bounded polymorphism

$$\textbf{def}\ \text{foo}[\text{T} <: \text{Int}](\text{x}: \text{T}): \text{T} = \ldots$$

- Types of things that are not **null**, things that are not functions, ...
- Record types with depth, width and permutation subtyping
- Equirecursive types

Orthologic Type System

Many common and less common constructs can be encoded in such system:

- Inheritence relations become classes
- Bounded polymorphism

$$\textbf{def}\ \text{foo}[\text{T} <: \text{Int}](\text{x}: \text{T}): \text{T} = \ldots$$

- Types of things that are not **null**, things that are not functions, ...
- Record types with depth, width and permutation subtyping
- Equirecursive types
- and more

Acknowledgement

- Viktor Kunčak
- Sankalp Gambhir
- Clément Pit-Claudel, Mario Bucev, Dragana Milovančević

Conclusion

- Orthologic
  - Efficient and Predictable Building Block
  - Normalization algorithm, proof system
  - Good logical properties (interpolation, ...)
  - Tons of applications, many more to explore!

Example

Assume $\neg(\neg a \vee (a \wedge b))$. Deduce:

$$
\begin{array}{ll}
a \wedge (\neg a \vee \neg b) & \text{NNF} \\
a, \quad (\neg a \vee \neg b) & \\
(\neg 1 \vee \neg b) & \text{substituting } a = 1 \\
\neg b &
\end{array}
$$

Then,

$$
\begin{array}{ll}
(\neg c \vee b) \vee (\neg b \wedge (c \vee \neg a)) & \\
\sim (\neg c \vee 0) \vee (1 \wedge (c \vee 0)) & \text{substituting } a = 1, b = 0 \\
\sim \neg c \vee c & \\
\sim 1 &
\end{array}
$$