Aesop: White-Box Automation for Lean 4

Jannis Limperg
University of Munich (LMU)
jannis@limperg.de

03. July 2024
Search Algorithm

Registering Rules

Built-In Rules

Debugging

Miscellaneous Features

Applications, Shortcomings and Work In Progress
Search Algorithm
A *rule* is an arbitrary Lean tactic.

Aesop provides convenient syntax (*rule builders*) for creating rules from theorems.

Aesop always operates with a user-defined *rule set*. 
Basic Tree Search

\[ \vdash A \rightarrow C \rightarrow A \land (B \lor C) \]
Basic Tree Search

\[
\vdash A \to C \to A \land (B \lor C)
\]

\[
\begin{array}{c}
\text{intros} \\
A, C \vdash A \land (B \lor C)
\end{array}
\]

\[
A, C \vdash A \land (B \lor C)
\]
Basic Tree Search

⊢ \( A \rightarrow C \rightarrow A \land (B \lor C) \)

- intros

\[ A, C \vdash A \land (B \lor C) \]

- And.intro

\[ A, C \vdash A \]
\[ A, C \vdash B \lor C \]
Basic Tree Search

⊢ \( A \rightarrow C \rightarrow A \land (B \lor C) \)

intros

\( A, C \vdash A \land (B \lor C) \)

And.intro

\( A, C \vdash A \)
\( A, C \vdash B \lor C \)

A
Basic Tree Search

\[ \vdash A \rightarrow C \rightarrow A \land (B \lor C) \]

intros

\[ A, C \vdash A \land (B \lor C) \]

And.intro

\[ A, C \vdash A \]

\[ A \]

\[ A, C \vdash B \lor C \]

Or.left

\[ A, C \vdash B \]

\[ A \]

\[ A, C \vdash C \]

\[ C \]
Basic Tree Search

\[
\Gamma \vdash A \to C \to A \land (B \lor C)
\]

**intros**

\[
A, C \vdash A \land (B \lor C)
\]

**And.intro**

\[
A, C \vdash A
\]

**Or.left**

\[
A, C \vdash B
\]

**Or.right**

\[
A, C \vdash C
\]
Basic Tree Search

⊢ 𝐴 → 𝐶 → 𝐴 ∧ (𝐵 ∨ 𝐶)

intros

A, C ⊢ 𝐴 ∧ (𝐵 ∨ 𝐶)

And.intro

A, C ⊢ 𝐴

A

A, C ⊢ 𝐵 ∨ 𝐶

Or.left

A, C ⊢ 𝐵

A, C ⊢ 𝐶

Or.right

C
Best-First Tree Search

⊢ \( A \rightarrow C \rightarrow A \land (B \lor C) \)

intros 100%

\( A, C \vdash A \land (B \lor C) \)

And.intro 100%

\( A, C \vdash A \)

A 100%

\( A, C \vdash B \lor C \)

Or.left 50%

\( A, C \vdash B \)

A, C ⊢ B

A 100%

Or.right 50%

\( A, C \vdash C \)

C 100%
Best-First Tree Search

\[ \vdash A \rightarrow C \rightarrow A \land (B \lor C) \quad 100\% \]

**intros 100\%**

\[ A, C \vdash A \land (B \lor C) \quad 100\% \]

**And.intro 100\%**

\[ A, C \vdash A \quad 100\% \]

\[ A, C \vdash B \lor C \quad 100\% \]

**Or.left 50\%**

\[ A, C \vdash B \quad 50\% \]

**Or.right 50\%**

\[ A, C \vdash C \quad 50\% \]

\[ C \quad 100\% \]
Safe Rules

- Run before unsafe rules
Safe Rules

- Run before unsafe rules
- If a safe rule succeeds on a goal $G$, no other rules are tried for $G$
Safe Rules

- Run before unsafe rules
- If a safe rule succeeds on a goal $G$, no other rules are tried for $G$
- Integer penalty
Safe Rules

• Run before unsafe rules

• If a safe rule succeeds on a goal \( G \), no other rules are tried for \( G \)

• Integer penalty

• Treated as 100% success probability
Safe Rules

- Run before unsafe rules
- If a safe rule succeeds on a goal $G$, no other rules are tried for $G$
- Integer penalty
- Treated as 100% success probability
- 😞 Good for performance

Users need to make sure that the rule really is safe.
Safe Rules

- Run before unsafe rules
- If a safe rule succeeds on a goal $G$, no other rules are tried for $G$
- Integer penalty
- Treated as 100% success probability
- 😃 Good for performance
- 😞 Users need to make sure that the rule really is safe
Examples

Safe rule: $\land$-introduction

$\Gamma \vdash A \land B$

$\begin{cases}
\Gamma \vdash A \\
\Gamma \vdash B
\end{cases}$

Unsafe rule: left $\lor$-introduction

$\Gamma \vdash A \lor B$

$\begin{cases}
\Gamma \vdash A
\end{cases}$
A rule $R$ is *logically safe* if it preserves provability:

For each goal $G$, if $G$ is provable and $R$, applied to $G$, generates subgoals $G_1, \ldots, G_n$, then $G_1, \ldots, G_n$ must still be provable.
When Is A Rule Safe?

A rule $R$ is *logically safe* if it preserves provability:

For each goal $G$, if $G$ is provable and $R$, applied to $G$, generates subgoals $G_1, \ldots, G_n$, then $G_1, \ldots, G_n$ must still be provable.

A rule $R$ is *relatively safe* if it preserves provability relative to a rule set $S$:

If a goal $G$ is provable with rules from $S$ and $R$, applied to $G$, generates subgoals $G_1, \ldots, G_n$, then $G_1, \ldots, G_n$ must still be provable with rules from $S$. 
Normalisation Rules

- Run before safe rules
Normalisation Rules

- Run before safe rules
- Integer penalty

Can establish invariants for other rules
Typically run multiple times on every goal
Normalisation Rules

- Run before safe rules
- Integer penalty
- Treated as 100% success probability
Normalisation Rules

- Run before safe rules
- Integer penalty
- Treated as 100% success probability
- May produce only one subgoal
Normalisation Rules

- Run before safe rules
- Integer penalty
- Treated as 100% success probability
- May produce only one subgoal
- Run in a fixpoint loop, i.e. until no normalisation rule succeeds any more
Normalisation Rules

- Run before safe rules
- Integer penalty
- Treated as 100% success probability
- May produce only one subgoal
- Run in a fixpoint loop, i.e. until no normalisation rule succeeds any more
- 🧪 Can establish invariants for other rules
Normalisation Rules

- Run before safe rules
- Integer penalty
- Treated as 100% success probability
- May produce only one subgoal
- Run in a fixpoint loop, i.e. until no normalisation rule succeeds any more
- 🐉 Can establish invariants for other rules
- 😞 Typically run multiple times on every goal
Example

∧-elimination

Γ, h : A ∧ B ⊢ T

Γ, h₁ : A, h₂ : B ⊢ T
Summary: Aesop’s Search Algorithm

Apply normalisation rules → progress

no progress

Apply safe rules

Apply unsafe rules

no progress → add subgoals to tree

can’t prove this goal

progress

add subgoals to tree

reinsert goal into goal queue
Registering Rules
Registering Rules

Globally

@\[aesop \text{ unsafe } 100\%\]

\textbf{theorem} And\texttt{.intro} : A \rightarrow B \rightarrow A \land B
Registering Rules

Globally

@ [aesop unsafe 100%]
\textbf{theorem} \texttt{And.intro} : A \rightarrow B \rightarrow A \land B

Locally

aesop (add 100\% And.intro)
Registering Rules

Globally

```
@aesop unsafe 100%
theorem And.intro : A → B → A ∧ B
```

Locally

```
aesop (add 100% And.intro)
```

Safe rules

```
@aesop safe 10
theorem And.intro : A → B → A ∧ B
```
A rule builder turns a declaration into an Aesop rule.

In the examples so far, we have implicitly used a default builder.

Aesop currently provides 7 rule builders.
Apply Builder

@[aesop safe apply 10]

**Theorem** And.intro : \( A \rightarrow B \rightarrow A \land B \)
Apply Builder

@[aesop safe apply 10]

**Theorem** And.intro : \( A \rightarrow B \rightarrow A \land B \)

Builds a rule that runs apply And.intro.
Constructors Builder

@[aesop 50% constructors]

inductive Or (A B : Prop) where
  left : A → Or A B
  right : B → Or A B

Equivalent to one apply rule for each constructor.
Cases Builder

@[aesop safe cases]

inductive Or (A B : Prop) where
  | left  : A → Or A B
  | right : B → Or A B

Builds a rule that runs cases on any hypothesis of type Or A B.
\[@\text{aesop safe forward}\]

\textbf{theorem pos_of_min_pos : } \forall \{x \ y : \mathbb{N}\},
\begin{align*}
0 & < \min x \ y \rightarrow \\
0 & < x \land 0 < y
\end{align*}

\[\Gamma, \ x \ y : \mathbb{N}, \ h : 0 < \min x \ y \vdash T\]
\[\Gamma, \ x \ y : \mathbb{N}, \ h : 0 < \min x \ y, \ h_1 : 0 < x \land 0 < y \vdash T\]
Destruct Builder

```lean
@[aesop safe destruct]
theorem pos_of_min_pos : ∀ {x y : ℕ},
  0 < min x y →
  0 < x ∧ 0 < y

Γ, x y : ℕ, h : 0 < min x y ⊢ T

Γ, x y : ℕ, h : 0 < x ∧ 0 < y ⊢ T
```
Aesop runs simp_all as a built-in normalisation rule with penalty 0.

This simp_all call uses the default simp set plus an Aesop-specific simp set.

The simp builder adds an equation or proposition to this Aesop-specific set.
Tactic Builder

aesop (add safe (by norm_num; done))

Requirement
If the tactic does not change the goal, it should fail.
Tactic Builder

aesop (add safe (by norm_num; done))

add_aesop_rules safe (by norm_num; done)
Tactic Builder

aesop (add safe (by norm_num; done))

add_aesop_rules safe (by norm_num; done)

Requirement
If the tactic does not change the goal, it should fail.
Built-In Rules
Logic: \( \land \)

\( \land \)-introduction (safe, penalty 101)

\[
\Gamma \vdash A \land B \\
\Gamma \vdash A \\
\Gamma \vdash B
\]
Logic: \( \land \)

\( \land \)-introduction (safe, penalty 101)

\[
\Gamma \vdash A \land B
\]

\[
\Gamma \vdash A \quad \Gamma \vdash B
\]

\( \land \)-elimination (norm, penalty 0)

\[
\Gamma, h : A \land B \vdash T
\]

\[
\Gamma, h_1 : A, h_2 : B \vdash T
\]
Logic: $\land$

$\land$-introduction (safe, penalty 101)

$$\Gamma \vdash A \land B$$

$$\Gamma \vdash A \quad \Gamma \vdash B$$

$\land$-elimination (norm, penalty 0)

$$\Gamma, \; h : A \land B \vdash T$$

$$\Gamma, \; h_1 : A, \; h_2 : B \vdash T$$

Similar for Prod, PProd, MProd
Logic: \( \lor \)

\( \lor \)-introduction (unsafe, success probability 50%)

\[
\Gamma \vdash A \lor B \\
\Gamma \vdash A \\
\Gamma \vdash B
\]

Similar for Sum, PSum
Logic: \( \lor \)

\( \lor \)-introduction (unsafe, success probability 50%)

\[
\Gamma \vdash A \lor B \\
\Gamma \vdash A \\
\Gamma \vdash B
\]

\( \lor \)-elimination (safe, penalty 100)

\[
\Gamma, h : A \lor B \vdash T \\
\Gamma, h : A \vdash T \\
\Gamma, h : B \vdash T
\]
Logic: \( \lor \)

\( \lor \)-introduction (unsafe, success probability 50%)

\[ \Gamma, \quad A \lor B \quad \vdash \quad A \\ \Gamma, \quad A \lor B \quad \vdash \quad B \]

\( \lor \)-elimination (safe, penalty 100)

\[ \Gamma, \quad h : A \lor B \vdash T \\ \Gamma, \quad h : A \vdash T \\ \Gamma, \quad h : B \vdash T \]

Similar for Sum, PSum
Logic: \( \forall \) and \( \rightarrow \)

\( \forall \)-introduction (norm, penalty -100)
Run the intros tactic
Logic: $\forall$ and $\rightarrow$

$\forall$-introduction (norm, penalty -100)
Run the intros tactic

$\forall$-elimination (unsafe, success probability 75%)

$$\Gamma, h : \forall(x_1 : A_1) \ldots (x_n : A_n), B \vdash B$$

$$\Gamma, h \vdash A_1 \quad \ldots \quad \Gamma, h \vdash A_n$$
Logic: ∃

∃-introduction (unsafe, success probability 30%)

\[ \Gamma \vdash \exists x, P \ x \]

\[ \Gamma \vdash P \ ?x \]
Logic: ∃

∃-introduction (unsafe, success probability 30%)

\[
\Gamma \vdash \exists x, \ P \ x \\
\Gamma \vdash P \ ?x
\]

∃-elimination (norm, penalty 0)

\[
\Gamma, \ h : \exists x : A, \ P \ x \vdash T \\
\Gamma, \ x : A, \ h : P \ x \vdash T
\]
Logic: $\exists$

$\exists$-introduction (unsafe, success probability 30%)

$$\Gamma \vdash \exists x, P \ x$$
$$\Gamma \vdash P \ ?x$$

$\exists$-elimination (norm, penalty 0)

$$\Gamma, h : \exists x : A, P \ x \vdash T$$
$$\Gamma, x : A, h : P \ x \vdash T$$

Similar for Subtype, Sigma, PSigma
Logic: $\leftrightarrow$

$\leftrightarrow$-introduction (safe, penalty 100)

$\Gamma \vdash A \leftrightarrow B$

$\Gamma \vdash A \rightarrow B$

$\Gamma \vdash B \rightarrow A$
Logic: $\iff$

$\iff$-introduction (safe, penalty 100)

$\Gamma \vdash A \iff B$

$\Gamma \vdash A \rightarrow B$  $\Gamma \vdash B \rightarrow A$

$\iff$ hypotheses

A hypothesis of type $A \iff B$ is treated as the equation $A = B$ by the simplifier and the substitution rule.
Logic: $\top$

$\top$-introduction$^1$ (safe, penalty 0)

\[ \Gamma \vdash \top \]

$^1_T = \text{True}$
Logic: $\top$

$\top$-introduction$^1$ (safe, penalty 0)

$\Gamma \vdash \top$

Similar for Unit, PUnit.

$^1_{\top} = \text{True}$
Logic: \( \bot \)

The simplifier already solves goals with an assumption \( h : \bot \).\(^2\)

\(^2\bot = \text{False}\)
The simplifier already solves goals with an assumption $h : \bot$.\(^2\)

We add destruct rules to conclude $\bot$ from Empty and PEmpty.
Logic: \( \neg \)

\( \neg \)-introduction

\[ \Gamma \vdash \neg A \]
\[ | \]
\[ \Gamma, \ h : A \vdash \bot \]
Negated hypotheses
Given a hypothesis of type $\neg A$, the simplifier replaces $A$ with $\bot$ everywhere in the goal.
The simplifier performs limited logical reasoning. If $A$ and $B$ are propositions:

- With assumption $h : A$: rewrite $A = \top$
- With assumption $h : \neg A$: rewrite $A = \bot$
- $(\top \land \bot) = \bot$
- $(\top \land \top) = \top$
- $(\top \rightarrow A) = A$
- etc.
The simplifier performs limited logical reasoning. If $A$ and $B$ are propositions:

- With assumption $h : A$: rewrite $A = T$
Logic: Simplifier

The simplifier performs limited logical reasoning. If $A$ and $B$ are propositions:

- With assumption $h : A$: rewrite $A = \top$
- With assumption $h : \neg A$: rewrite $A = \bot$
Logic: Simplifier

The simplifier performs limited logical reasoning. If $A$ and $B$ are propositions:

- With assumption $h : A$: rewrite $A = \top$
- With assumption $h : \neg A$: rewrite $A = \bot$
- $(\top \land \bot) = \bot$
Logic: Simplifier

The simplifier performs limited logical reasoning. If $A$ and $B$ are propositions:

- With assumption $h : A$: rewrite $A = T$
- With assumption $h : \neg A$: rewrite $A = \bot$
- $(\top \land \bot) = \bot$
- $(\top \land \top) = T$
The simplifier performs limited logical reasoning. If $A$ and $B$ are propositions:

- With assumption $h : A$: rewrite $A = \top$
- With assumption $h : \neg A$: rewrite $A = \bot$
- $(\top \land \bot) = \bot$
- $(\top \land \top) = \top$
- $(\top \rightarrow A) = A$
The simplifier performs limited logical reasoning. If $A$ and $B$ are propositions:

- With assumption $h : A$: rewrite $A = \top$
- With assumption $h : \neg A$: rewrite $A = \bot$
- $(\top \land \bot) = \bot$
- $(\top \land \top) = \top$
- $(\top \rightarrow A) = A$
- etc.
Logic: Completeness

In practice, these rules solve most ‘purely logical’ goals.

However, they are incomplete for first-order and even propositional logic.
Equality

Simplifier (norm, penalty 0)
Run the simp_all tactic as described previously
Equality

**Simplifier (norm, penalty 0)**
Run the simp_all tactic as described previously

**Reflexivity (safe, penalty 0)**
Run the rfl tactic
Equality

Simplifier (norm, penalty 0)
Run the simp_all tactic as described previously

Reflexivity (safe, penalty 0)
Run the rfl tactic

Substitution (norm, penalty -50)
Run the subst tactic on any hypothesis of type \(x = t\) or \(t = x\) where \(x\) is a variable.
This substitutes \(t\) for \(x\) everywhere in the goal and removes the now-redundant hypothesis.
Case Splitting

Split target (safe, penalty 100)
Runs the split tactic. This tactic looks for **if-then-else** and **match** expressions in the target and performs case splits on their discriminees.
Case Splitting

Split target (safe, penalty 100)
Runs the split tactic.
This tactic looks for **if-then-else** and **match** expressions in the target and performs case splits on their discriminreees.

Split hypotheses (safe, penalty 1000)
Ditto, but we look for case splits in hypotheses.
Extensionality

Extensionality (unsafe, success probability 80%)
Run the ext tactic.
This exhaustively applies extensionality lemmas to an equational goal. E.g.:

\[ \Gamma, p \; q : A \times B \vdash p = q \]
\[ \Gamma, p \; q : A \times B \vdash p.1 = q.1 \land p.2 = q.2 \]
Debugging
Leftover Goals

When Aesop fails to solve a goal, it reports the goals that remain after safe rules have been exhaustively applied.

This helps to check whether the safe rules do what you think they should.
Proof Generation

```lean
theorem last_cons {a : α} {l : List α} (h : l ≠ nil) :
    last (a :: l) (cons_ne_nil a l) = last l h := by
aesop? (add 1% cases List)
```
Proof Generation

```lean
theorem last_cons {a : α} {l : List α} (h : l ≠ nil) :
  last (a :: l) (cons_ne_nil a l) = last l h := by
aesop? (add 1% cases List)
```

`aesop?` generates a proof script:

```lean
intro h
cases l with
  | nil =>
    simp_all only [last, ne_eq]
split
  | cons head tail => simp_all only [last]
```

One click replaces `aesop?` with the generated proof.
Tracing

```mermaid
set_option trace.aesop true in aesop
```
Tracing

**set_option** `trace.aesop` `true` **in** `aesop`

- Lists the rules that Aesop (tried to) run and the resulting goals
Tracing

```plaintext
set_option trace.aesop true in aesop
```

- Lists the rules that Aesop (tried to) run and the resulting goals
- For other trace options see autocompletion for trace.aesop.
Miscellaneous Features
Custom Rule Sets

```
-- RuleSet.lean

-- Must be in a different file for technical reasons
declare_aesop_rule_sets [Foo]
```
Custom Rule Sets

-- RuleSet.lean

-- Must be in a different file for technical reasons
declare_aesop_rule_sets [Foo]

import RuleSet

[@aesop 100% (rule_sets [Foo])]
theorem foo
Custom Rule Sets

-- RuleSet.lean

-- Must be in a different file for technical reasons
declare_aesop_rule_sets [Foo]

import RuleSet

@[aesop 100% (rule_sets [Foo])]

theorem foo

Used for domain-specific automation, e.g. continuity, measurability, ...
Metavariavbles

Aesop supports rules that generate metavariables:

```plaintext
example \{a \ b \ c \ d : \text{Nat}\} \ (h_1 : a < b)
    (h_2 : a < c) (h_3 : c < d) : a < d := by
    apply \text{Nat.ltt\_trans}
    -- ⊢ a < ?x
    -- ⊢ ?x < d

example \{a \ b \ c \ d : \text{Nat}\} \ (h_1 : a < b)
    (h_2 : a < c) (h_3 : c < d) : a < d := by
    aesop (add 1\% \text{Nat.ltt\_trans})
```

Aesop's search algorithm should be complete even in the presence of metavariables. This is very expensive.
Metavariabes

Aesop supports rules that generate metavariables:

```
example {a b c d : Nat} (h₁ : a < b)  
       (h₂ : a < c) (h₃ : c < d) : a < d := by 
       apply Nat.lt_trans 
       -- ⊢ a < ?x 
       -- ⊢ ?x < d
```

```
example {a b c d : Nat} (h₁ : a < b)  
       (h₂ : a < c) (h₃ : c < d) : a < d := by 
       aesop (add 1% Nat.lt_trans)
```

• 🧵 Aesop’s search algorithm should be complete even in the presence of metavariables
Metavariables

Aesop supports rules that generate metavariables:

```
example {a b c d : Nat} (h₁ : a < b)
    (h₂ : a < c) (h₃ : c < d) : a < d := by
    apply Nat.lt_trans
    -- ⊢ a < ?x
    -- ⊢ ?x < d

example {a b c d : Nat} (h₁ : a < b)
    (h₂ : a < c) (h₃ : c < d) : a < d := by
    aesop (add 1% Nat.lt_trans)
```

- 😄 Aesop’s search algorithm should be complete even in the presence of metavariables
- 😞 This is very expensive
Applications, Shortcomings and Work In Progress
Applications

- General-purpose automation
Applications

- General-purpose automation
- Domain-specific solvers: continuity, measurability, aesop_cat, ...

Applications

• General-purpose automation

• Domain-specific solvers: continuity, measurability, aesop_cat, ...

• Domain-specific ‘goal preprocessors’ with aesop?
Applications

- General-purpose automation

- Domain-specific solvers: continuity, measurability, aesop_cat, ...

- Domain-specific ‘goal preprocessors’ with aesop?

Shortcomings and Work In Progress

- The built-in logical rules do not deal well with all-quantified hypotheses and sometimes negation
Shortcomings and Work In Progress

• The built-in logical rules do not deal well with all-quantified hypotheses and sometimes negation

• The default rule set is currently missing many rules
Shortcomings and Work In Progress

• The built-in logical rules do not deal well with all-quantified hypotheses and sometimes negation
• The default rule set is currently missing many rules
• Repeated calls to simp during normalisation are bad for performance
Shortcomings and Work In Progress

• The built-in logical rules do not deal well with all-quantified hypotheses and sometimes negation
• The default rule set is currently missing many rules
• Repeated calls to simp during normalisation are bad for performance
• Nontrivial sets of forward rules are very slow (WIP)