



Theorembeweiserpraktikum

Aesop: A General Proof Search Tactic

Jakob von Raumer, Sebastian Ullrich | SS 2022



THEOREM PROVER

Recursive Simplification Revisited

We learned that `simp` can be quite powerful

```
example (h1 : y = 0 → x = 0) (h2 : p → 0 = y) (h3 : p) : x = 0 := by simp [h1, h2, h3]
```

Recursive Simplification Revisited

We learned that `simp` can be quite powerful

```
example (h1 : y = 0 → x = 0) (h2 : p → 0 = y) (h3 : p) : x = 0 := by simp [h1, h2, h3]
```

Limitations:

- depth-first search with very low default max depth (2)
- supports simple *backward reasoning* only

```
example (h1 : ∀ y, p y → y = 0) (h2 : p x) : 2 * x = 0 := sorry -- by simp [*]
example : ∃ x, x = 0 := sorry -- by simp [Exists.intro]
example : match n with | 0 => True | n + 1 => True := sorry -- by simp
```

- usability: must inspect trace to find out what went wrong!

Aesop (<https://github.com/JLimperg/aesop>) is “a work-in-progress proof search tactic for Lean 4”

```
import Aesop -- see also lakefile.lean for setup

example (h1 : ∀ x, p x → x = 0) (h2 : p x) : 2 * x = 0 := by aesop (add safe forward h1)
example : ∃ x, x = 0 := by aesop
example : match n with | 0 => True | n + 1 => True := by aesop
```

- *best-first* search explores different branches in turn
- applies *safe* rules exhaustively followed by *unsafe* rules, shows state after safe rules on error

```
example (h : p ∧ q) : r ∨ s := by aesop
-- After applying safe rules, Aesop tried to solve these goals:
-- (unprovable)
-- : p
-- : q
-- ⊢ r ∨ s
```

Aesop (<https://github.com/JLimperg/aesop>) is “a work-in-progress proof search tactic for Lean 4”

```
import Aesop -- see also lakefile.lean for setup

example (h1 : ∀ x, p x → x = 0) (h2 : p x) : 2 * x = 0 := by aesop (add safe forward h1)
example : ∃ x, x = 0 := by aesop
example : match n with | 0 => True | n + 1 => True := by aesop
```

- *best-first* search explores different branches in turn
 - applies *safe* rules exhaustively followed by *unsafe* rules, shows state after safe rules on error
- Builtin safe/unsafe examples:

```
safe Eq.refl
unsafe 30% constructors Exists

[safe cases, unsafe 50% constructors] 0r
```

Adding Rules

Add globally:

```
@[aesop safe constructors] inductive Foo ...  
attribute [aesop safe cases] Foo -- post hoc
```

or locally:

```
by aesop (add safe constructors Foo, ...)
```

Adding Rules

Add globally:

```
@[aesop safe constructors] inductive Foo ...  
attribute [aesop safe cases] Foo -- post hoc
```

or locally:

```
by aesop (add safe constructors Foo, ...)
```

More rule examples; see <https://github.com/JLimperg/aesop#rule-builders> for everything

```
safe (cases (patterns := [<.skip, _> ⇓ _ : _])) Bigstep -- apply rule inversion on any `skip` hypothesis  
unsafe apply f -- default success probability is 50%  
safe elim g -- like `forward`, but remove used hypotheses  
norm simp h  
norm unfold f
```

Hints For Effectively Using Aesop

- Don't be misled by (unknown) goals in the output, focus on (unprovable) s
- Start with safe to observe a rule's effects on the goal(s), transition to unsafe only when necessary
- Use norm unfold f (even if f is already [simp]) to expose a function's hidden match block and split its cases

Demo

<https://github.com/IPDSnelting/tba-2022/blob/master/TBA/AesopSort.lean>