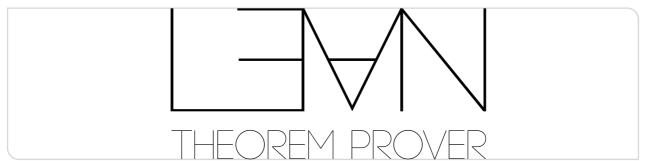




### Theorembeweiserpraktikum

Aesop: A General Proof Search Tactic

Jakob von Raumer, Sebastian Ullrich | SS 2022



## **Recursive Simplification Revisited**



We learned that simp can be quite powerful

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





We learned that simp can be quite powerful

```
example (h1: y = 0 \rightarrow x = 0) (h2: p \rightarrow 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: \forall y, p y \rightarrow y = 0) (h2: p x): 2 * x = 0:= sorry -- by simp [*]
example : \exists 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



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
import TBA.Util.AesopExts -- temporary
example (h1: \forall x, p x \rightarrow x = 0) (h2: p x): 2 * x = 0 := by aesop (add safe forward h1)
example : \exists x \cdot 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 \land q) : r \lor s := by aesop
-- After applying safe rules, Aesop tried to solve these goals:
-- (unprovable)
-- : p
-- : q
-- + r V s
```

## Aesop



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
import TBA.Util.AesopExts -- temporary
example (h1: \forall x, p x \rightarrow x = 0) (h2: p x): 2 * x = 0 := by aesop (add safe forward h1)
example : \exists x \cdot 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] Or
```

## **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



## **Custom Tactic Rules from Our Repo**

- enabled by default as safe: substVars, split, splitAt
- disabled by default: simpAll (probably safe), elimAny (might be safe)

## 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://gist.github.com/Kha/96d67c8b947b48f8786aea90857fbb5c