Towards General-Purpose Proof Automation for Lean

Jannis Limperg

Vrije Universiteit Amsterdam

5 January 2021

“Towards”

Architecture

Customisation

Debugging

Proof Search

1. Simplify everywhere.
2. Apply safe resolution rules.
3. Apply unsafe resolution rules.

```
A B : set α
⊢ A ∩ B ⊆ A ∪ B
```

```
A B : set α
⊢ A ∩ B ⊆ A ∪ B

...
⊢ (λ x, A x ∧ B x) ⊆ (λ x, A x ∨ B x)
```

```
A B : set α
⊢ A ∩ B ⊆ A ∪ B

...
⊢ (λ x, A x ∧ B x) ⊆ (λ x, A x ∨ B x)

...
⊢ ∀ x, A x ∧ B x → A x ∨ B x
```

```
...
⊢ ∀ x, A x ∧ B x → A x ∨ B x
```

```
...
⊢ ∀ x, A x ∧ B x → A x ∨ B x

...
x : α
h : A x ∧ B x
⊢ A x ∨ B x
```

```
...
⊢ ∀ x, A x ∧ B x → A x ∨ B x

...
x : α
h : A x ∧ B x
⊢ A x ∨ B x

...
h₁ : A x
h₂ : B x
⊢ A x ∨ B x
```

```
...
⊢ ∀ x, A x ∧ B x → A x ∨ B x
```

```
...
x : α
h : A x ∧ B x
⊢ A x ∨ B x
```

```
...
h₁ : A x
h₂ : B x
⊢ A x ∨ B x
```

```
...      |    ...
⊢ A x    |    ⊢ B x
```

```
@[backward unsafe]
lemma or_intro_left  {α β} : α → α ∨ β

@[backward unsafe]
lemma or_intro_right {α β} : β → α ∨ β
```

Customisation

Customisation

```
@[backward safe]
meta def continuous_comp_rule : tactic unit :=
/- Apply continuous.comp unless the function
   is constant or the identity. -/
```

Customisation

```
@[backward safe]
meta def continuous_comp_rule : tactic unit :=
/- Apply continuous.comp unless the function
   is constant or the identity. -/

@[backward finisher]
meta def arithmetic_rule : tactic unit :=
/- If target is n < m or n > m or ...,
   try linarith and nlinarith. -/
```

Debugging

Debugging

```
α β γ : Prop
⊢ α → (β ∨ γ) ∨ α
```

Debugging

```
α β γ : Prop
⊢ α → (β ∨ γ) ∨ α

- intros
- apply or.intro_left
- apply or.intro_right
  - assumption
```

```
- intros
- apply or.intro_left
  - apply or.intro_left
    - failure: no rules applicable
  - apply or.intro_right
    - failure: no rules applicable
- apply or.intro_right
  - assumption
```

```
- intros
- apply or.intro_left
- apply or.intro_right
  α β γ : Prop
  a : α
  ⊢ α
  - assumption
```

```
- intros (0 ms)
- 42 rules not applicable (10ms)
- apply or.intro_left (0ms + 5ms)
- apply or.intro_right (0 ms)
  - assumption (0 ms)
```

Evaluation

Power	☹
Performance	☹
Customisability	☺
Transparency	☺

Please Discuss!

- Proof search based on simplification and resolution rules.
- Custom rules for heuristics, decision procedures, ...
- Interactive search tree for debugging.

Jannis Limperg
`https://limperg.de`
`j.b.limperg@vu.nl` (or on Zulip)

Tagging

- Speculative tagging

Tagging

- Speculative tagging
- Linting (eg `simp_nf`)

Tagging

- Speculative tagging
- Linting (eg `simp_nf`)
- Automated refactoring