A Novice-Friendly Induction Tactic for Lean

Jannis Limperg

Vrije Universiteit Amsterdam

CPP 2021

```
case semantics.big_step.skip
t h : state
⊢ false
```

```
case semantics.big_step.assign
t : state,
h_x : string,
h_a : state → ℕ,
h_s : state
⊢ false
```

```
case semantics.big_step.seq
t : state,
h_S h_T : program,
h_s h_t h_u : state,
h_hS : (h_S, h_s) ⟹ h_t,
h_hT : (h_T, h_t) ⟹ h_u,
h_ih_hS h_ih_hT : false
⊢ false
```

```
case semantics.big_step.ite_true
t : state,
h_b : state → Prop,
h_S h_T : program,
h_s h_t : state,
h_hcond : h_b h_s,
h_hbody : (h_S, h_s) ⟹ h_t,
h_ih : false
⊢ false
```

```
case semantics.big_step.ite_false
t : state,
h_b : state → Prop,
h_S h_T : program,
h_s h_t : state,
h_hcond : ¬h_b h_s,
h_hbody : (h_T, h_s) ⟹ h_t,
h_ih : false
⊢ false
```

```
case semantics.big_step.while_true
t : state,
h_b : state → Prop,
h_S : program,
h_s h_t h_u : state,
h_hcond : h_b h_s,
h_hbody : (h_S, h_s) ⟹ h_t,
h_hrest : (while h_b h_S, h_t) ⟹ h_u,
h_ih_hbody h_ih_hrest : false
⊢ false
```

```
case semantics.big_step.while_false
t : state,
h_b : state → Prop,
h_S : program,
h_s : state,
h_hcond : ¬h_b h_s
⊢ false
```

Complex Indices

Induction Hypothesis Generalisation

Naming

Complex Indices

Induction Hypothesis Generalisation

Naming

Complex Indices

```
∀ S s t,
 (while (λ _, true) S, s) ⟹ t → false
```

Complex Indices

```
∀ S s t,
 (while (λ _, true) S, s) ⟹ t → false

case skip
t t′ : state
⊢ false
```

McBride's Method

```
...
h : (while (λ _, true) S, s) ⟹ t
⊢ false
```

[Conor McBride, Elimination with a Motive, TYPES 2000]

McBride's Method

```
…
h : (while (λ _, true) S, s) ⟹ t
⊢ false
```

```
…
i : program × state,
h : i ⟹ t
⊢ (while (λ _, true) S, s) = i → false
```

[Conor McBride, Elimination with a Motive, TYPES 2000]

McBride's Method

```
…
h : (while (λ _, true) S, s) ⟹ t
⊢ false
```

```
…
i : program × state,
h : i ⟹ t
⊢ (while (λ _, true) S, s) = i → false
```

```
case skip
…
⊢ (while (λ _, true) S, s) = (skip, u) → false
```

[Conor McBride, Elimination with a Motive, TYPES 2000]

Induction Hypothesis Simplification

```
∀ S′ s′,
  (while (λ _, true) S′, s′) =
  (while (λ _, true) S, t) →
  false
```

Induction Hypothesis Simplification

```
∀ S′ s′,
  (while (λ _, true) S′, s′) =
  (while (λ _, true) S, t) →
  false
```

```
(while (λ _, true) ?S′, ?s′) =
(while (λ _, true) S,  t) →
false
```

Induction Hypothesis Simplification

```
∀ S′ s′,
  (while (λ _, true) S′, s′) =
  (while (λ _, true) S, t) →
  false
```

```
(while (λ _, true) ?S′, ?s′) =
(while (λ _, true) S,  t) →
false
```

```
(while (λ _, true) S, t) =
(while (λ _, true) S, t) →
false
```

Induction Hypothesis Simplification

```
∀ S′ s′,
  (while (λ _, true) S′, s′) =
  (while (λ _, true) S, t) →
  false

(while (λ _, true) ?S′, ?s′) =
(while (λ _, true) S,  t) →
false

(while (λ _, true) S, t) =
(while (λ _, true) S, t) →
false

false
```

Complex Indices

Induction Hypothesis Generalisation

Naming

Induction Hypothesis Generalisation

[Benjamin C. Pierce et al., Software Foundations]

Induction Hypothesis Generalisation

```
∀ n m, n + n = m + m → n = m
```

[Benjamin C. Pierce et al., Software Foundations]

Induction Hypothesis Generalisation

```
∀ n m, n + n = m + m → n = m
```

```
case succ
m n : ℕ
ih : n + n = m + m → n = m
h : n + n + 2 = m + m
⊢ n + 1 = m
```

[Benjamin C. Pierce et al., Software Foundations]

Induction Hypothesis Generalisation

```
∀ n m, n + n = m + m → n = m
```

```
case succ
m n : ℕ
ih : n + n = m + m → n = m
h : n + n + 2 = m + m
⊢ n + 1 = m
```

```
…
ih : ∀ m, n + n = m + m → n = m
h : n + n + 2 = m + m
⊢ n + 1 = m
```

[Benjamin C. Pierce et al., Software Foundations]

```
x : X
n m : ℕ
⊢ n + m = m + n
```

```
x : X
n m : ℕ
⊢ n + m = m + n

ih : X → ∀ m, n + m = m + n
```

Criteria for Auto-Generalisation

Connection with the target

```
n m : ℕ
⊢ n + m = m + n
```

Criteria for Auto-Generalisation

Connection with the target

```
n m : ℕ
⊢ n + m = m + n
```

Connection with the eliminated hypothesis

```
n : ℕ
h₁ : n ≥ 0
h₂ : n ≠ 0
⊢ n > 0
```

Complex Indices

Induction Hypothesis Generalisation

Naming

Naming

Naming

```
∀ α (r : α → a → Type) (a b c : α)
  (h₁ : tc r a b) (h₂ : tc r b c), tc r a c
```

Naming

```
∀ α (r : α → a → Type) (a b c : α)
  (h₁ : tc r a b) (h₂ : tc r b c), tc r a c
```

```
α : Type
r : α → α → Type
c a b h₁_x h₁_y h₁_z : α
h₁_hr : r h₁_x h₁_y
h₁_ht : tc r h₁_y h₁_z
h₁_ih : tc r h₁_z c → tc r h₁_y c
h₂ : tc r h₁_z c
⊢ tc r h₁_x c
```

Naming

```
∀ α (r : α → a → Type) (a b c : α)
  (h₁ : tc r a b) (h₂ : tc r b c), tc r a c
```

```
α : Type
r : α → α → Type
c x y z : α
hr : r x y
h₁ : tc r y z
IHh₁ : tc r z c → tc r y c
h₂ : tc r z c
⊢ tc r x c
```

Naming

```
∀ α (r : α → a → Type) (a b c : α)
  (h₁ : tc r a b) (h₂ : tc r b c), tc r a c
```

```
α : Type
r : α → α → Type
a y b c : α
hr : r a y
h₁ : tc r y b
ih : ∀ c, tc r b c → tc r y c
h₂ : tc r b c
⊢ tc r a c
```

Index Association

h_1 : tc r a b

…
hr : r x y
h_1 : tc r y z
…

Index Association

```
h1 : tc r a b

...
hr : r x y
h1 : tc r y z
...

inductive tc {α : Type} (r : α → α → Type) :
  α → α → Type
| base : ∀ x y (hr : r x y), tc x y
| step : ∀ x y z (hr : r x y) (ht : tc y z),
    tc x z
```

Type-Based Naming

```
class variable_names (α : Type) : Type :=
(names : list name)
```

Type-Based Naming

```
class variable_names (α : Type) : Type :=
(names : list name)

instance : variable_names ℕ :=
⟨[`n, `m, `o]⟩

instance : variable_names Prop :=
⟨[`P, `Q, `R]⟩

instance {α} [variable_names α] :
  variable_names (list α) :=
⟨(names α).map (λ n, n.append_suffix "s")⟩
```

Summary

- Simplification of induction hypotheses
- Auto-generalisation
- Index- and type-based naming
- (Not in this talk:) Evaluation of Lean 3's Metaprogramming Framework

Jannis Limperg
j.b.limperg@vu.nl
`https://limperg.de/contact`