Embedding specialized proof languages into Lean
A Case Study

Simon Hudon

Department of Philosophy and
Department of Mathematical Sciences
Carnegie Mellon University

work done in the context of my PhD at York University

January 2019

Dependent Type Theory

Dependent type theory is a general purpose reasoning framework
It is useful for reasoning about:

- algebra
- analysis
- functional programs and data structures
- theory of computation
- etc
- etc

Dependent Type Theory

Dependent type theory is a general purpose reasoning framework
It is useful for reasoning about:

- algebra
- analysis
- functional programs and data structures
- theory of computation
- etc
- etc

Dependent Type Theory

Dependent type theory is a general purpose reasoning framework
It is useful for reasoning about:

- algebra
- analysis
- functional programs and data structures
- theory of computation
- etc
- etc

Dependent Type Theory

Dependent type theory is a general purpose reasoning framework
It is useful for reasoning about:

- algebra
- analysis
- functional programs and data structures
- theory of computation
- etc
- etc

Dependent Type Theory

Dependent type theory is a general purpose reasoning framework
It is useful for reasoning about:

- algebra
- analysis
- functional programs and data structures
- theory of computation
- etc
- etc

Dependent Type Theory

Dependent type theory is a general purpose reasoning framework
It is useful for reasoning about:

- algebra
- analysis
- functional programs and data structures
- theory of computation
- etc
- etc

Dependent Type Theory

Dependent type theory is a general purpose reasoning framework
It is useful for reasoning about:

- algebra
- analysis
- functional programs and data structures
- theory of computation
- etc
- etc

Dependent Type Theory

Dependent type theory is a general purpose reasoning framework
It is useful for reasoning about:

- algebra
- analysis
- functional programs and data structures
- theory of computation
- etc
- etc

Specialized Logics

Specialized languages and logics are useful tools for emphasizing specific aspects in models

Examples:

- hoare logic — input / output relation for imperative programs
- separation logic — resource consumption in programs
- temporal logic — evolution of the state of a computation over time
- communicating sequential processes (CSP) — interactions between a set of processes

Specialized Logics

Specialized languages and logics are useful tools for emphasizing specific aspects in models
Examples:

- hoare logic — input / output relation for imperative programs
- separation logic — resource consumption in programs
- temporal logic — evolution of the state of a computation over time
- communicating sequential processes (CSP) — interactions between a set of processes

Specialized Logics

Specialized languages and logics are useful tools for emphasizing specific aspects in models
Examples:

- hoare logic — input / output relation for imperative programs
- separation logic — resource consumption in programs
- temporal logic — evolution of the state of a computation over time
- communicating sequential processes (CSP) — interactions between a set of processes

Specialized Logics

Specialized languages and logics are useful tools for emphasizing specific aspects in models
Examples:

- hoare logic — input / output relation for imperative programs
- separation logic — resource consumption in programs
- temporal logic — evolution of the state of a computation over time
- communicating sequential processes (CSP) — interactions between a set of processes

Specialized Logics

Specialized languages and logics are useful tools for emphasizing specific aspects in models
Examples:

- hoare logic — input / output relation for imperative programs
- separation logic — resource consumption in programs
- temporal logic — evolution of the state of a computation over time
- communicating sequential processes (CSP) — interactions between a set of processes

How to ...

The many possible approaches to building tools for a logic:

- build specialized provers / tools

  pro: lots of freedom in deciding how the provers will work

  cons: demands a lot of expertise other than domain expertise
- write a *deep* embedding in a proof assistant

  pro: can reuse some of the facilities of the prover

  cons: many features of the prover must be modelled in its own logic (hard)
- write a *shallow* embedding in a proof assistant

  pro: can reuse more support from the prover

  cons: proofs become specific to chosen encoding

How to ...

The many possible approaches to building tools for a logic:

- build specialized provers / tools

  pro: lots of freedom in deciding how the provers will work
  cons: demands a lot of expertise other than domain expertise

- write a *deep* embedding in a proof assistant

  pro: can reuse some of the facilities of the prover
  cons: many features of the prover must be modelled in its own logic (hard)

- write a *shallow* embedding in a proof assistant

  pro: can reuse more support from the prover
  cons: proofs become specific to chosen encoding

How to ...

The many possible approaches to building tools for a logic:

- build specialized provers / tools

  pro: lots of freedom in deciding how the provers will work
  cons: demands a lot of expertise other than domain expertise

- write a *deep* embedding in a proof assistant

  pro: can reuse some of the facilities of the prover
  cons: many features of the prover must be modelled in its own logic (hard)

- write a *shallow* embedding in a proof assistant

  pro: can reuse more support from the prover
  cons: proofs become specific to chosen encoding

How to ...

The many possible approaches to building tools for a logic:

- build specialized provers / tools

  pro: lots of freedom in deciding how the provers will work
  cons: demands a lot of expertise other than domain expertise

- write a *deep* embedding in a proof assistant

  pro: can reuse some of the facilities of the prover
  cons: many features of the prover must be modelled in its own logic (hard)

- write a *shallow* embedding in a proof assistant

  pro: can reuse more support from the prover
  cons: proofs become specific to chosen encoding

How to ...

The many possible approaches to building tools for a logic:

- build specialized provers / tools

  pro: lots of freedom in deciding how the provers will work

  cons: demands a lot of expertise other than domain expertise

- write a *deep* embedding in a proof assistant

  pro: can reuse some of the facilities of the prover

  cons: many features of the prover must be modelled in its own logic (hard)

- write a *shallow* embedding in a proof assistant

  pro: can reuse more support from the prover

  cons: proofs become specific to chosen encoding

How to ...

The many possible approaches to building tools for a logic:

- build specialized provers / tools

  pro: lots of freedom in deciding how the provers will work

  cons: demands a lot of expertise other than domain expertise

- write a *deep* embedding in a proof assistant

  pro: can reuse some of the facilities of the prover

  cons: many features of the prover must be modelled in its own logic (hard)

- write a *shallow* embedding in a proof assistant

  pro: can reuse more support from the prover

  cons: proofs become specific to chosen encoding

How to ...

The many possible approaches to building tools for a logic:

- build specialized provers / tools

  pro: lots of freedom in deciding how the provers will work

  cons: demands a lot of expertise other than domain expertise

- write a *deep* embedding in a proof assistant

  pro: can reuse some of the facilities of the prover

  cons: many features of the prover must be modelled in its own logic (hard)

- write a *shallow* embedding in a proof assistant

  pro: can reuse more support from the prover

  cons: proofs become specific to chosen encoding

How to ...

The many possible approaches to building tools for a logic:

- build specialized provers / tools

  pro: lots of freedom in deciding how the provers will work

  cons: demands a lot of expertise other than domain expertise

- write a *deep* embedding in a proof assistant

  pro: can reuse some of the facilities of the prover

  cons: many features of the prover must be modelled in its own logic (hard)

- write a *shallow* embedding in a proof assistant

  pro: can reuse more support from the prover

  cons: proofs become specific to chosen encoding

How to ...

The many possible approaches to building tools for a logic:

- build specialized provers / tools

  pro: lots of freedom in deciding how the provers will work

  cons: demands a lot of expertise other than domain expertise

- write a *deep* embedding in a proof assistant

  pro: can reuse some of the facilities of the prover

  cons: many features of the prover must be modelled in its own logic (hard)

- write a *shallow* embedding in a proof assistant

  pro: can reuse more support from the prover

  cons: proofs become specific to chosen encoding

How to ...

The many possible approaches to building tools for a logic:

- build specialized provers / tools

  pro: lots of freedom in deciding how the provers will work

  cons: demands a lot of expertise other than domain expertise

- write a *deep* embedding in a proof assistant

  pro: can reuse some of the facilities of the prover

  cons: many features of the prover must be modelled in its own logic (hard)

- write a *shallow* embedding in a proof assistant

  pro: can reuse more support from the prover

  cons: proofs become specific to chosen encoding

How to ... (continued)

The many possible approaches to building tools for a logic (continued):

- declare special notation on top of a *deep* or *shallow* embedding

  pro: can immitate the look of desired logic

  con: parser often limits how close we can get to desired logic

- override tactic notation (thanks to Lean!)

  pro: can allow the user to reason in terms of the logic rather than in terms of its encoding without implementing a new prover from scratch

  con: —

- define top-level syntax (thanks to Lean!)

  pro: can embed a complete language inside a prover

  con: too awesome

How to ... (continued)

The many possible approaches to building tools for a logic (continued):

- declare special notation on top of a *deep* or *shallow* embedding

  pro: can immitate the look of desired logic

  con: parser often limits how close we can get to desired logic

- override tactic notation (thanks to Lean!)

  pro: can allow the user to reason in terms of the logic rather than in terms of its encoding without implementing a new prover from scratch

  con: —

- define top-level syntax (thanks to Lean!)

  pro: can embed a complete language inside a prover

  con: too awesome

How to ... (continued)

The many possible approaches to building tools for a logic (continued):

- declare special notation on top of a *deep* or *shallow* embedding

  pro: can immitate the look of desired logic
  con: parser often limits how close we can get to desired logic

- override tactic notation (thanks to Lean!)

  pro: can allow the user to reason in terms of the logic rather than in terms of its encoding without implementing a new prover from scratch
  con: —

- define top-level syntax (thanks to Lean!)

  pro: can embed a complete language inside a prover
  con: too awesome

How to ... (continued)

The many possible approaches to building tools for a logic (continued):

- declare special notation on top of a *deep* or *shallow* embedding

  pro: can immitate the look of desired logic
  con: parser often limits how close we can get to desired logic

- override tactic notation (thanks to Lean!)

  pro: can allow the user to reason in terms of the logic rather than in terms of its encoding without implementing a new prover from scratch
  con: —

- define top-level syntax (thanks to Lean!)

  pro: can embed a complete language inside a prover
  con: too awesome

How to ... (continued)

The many possible approaches to building tools for a logic (continued):

- declare special notation on top of a *deep* or *shallow* embedding

  pro: can immitate the look of desired logic

  con: parser often limits how close we can get to desired logic

- override tactic notation (thanks to Lean!)

  pro: can allow the user to reason in terms of the logic rather than in terms of its encoding without implementing a new prover from scratch

  con: —

- define top-level syntax (thanks to Lean!)

  pro: can embed a complete language inside a prover

  con: too awesome

How to ... (continued)

The many possible approaches to building tools for a logic (continued):

- declare special notation on top of a *deep* or *shallow* embedding

  pro: can immitate the look of desired logic

  con: parser often limits how close we can get to desired logic

- override tactic notation (thanks to Lean!)

  pro: can allow the user to reason in terms of the logic rather than in terms of its encoding without implementing a new prover from scratch

  con: —

- define top-level syntax (thanks to Lean!)

  pro: can embed a complete language inside a prover

  con: too awesome

How to ... (continued)

The many possible approaches to building tools for a logic (continued):

- declare special notation on top of a *deep* or *shallow* embedding

  pro: can immitate the look of desired logic

  con: parser often limits how close we can get to desired logic
- override tactic notation (thanks to Lean!)

  pro: can allow the user to reason in terms of the logic rather than in terms of its encoding without implementing a new prover from scratch

  con: —
- define top-level syntax (thanks to Lean!)

  pro: can embed a complete language inside a prover

  con: too awesome

How to ... (continued)

The many possible approaches to building tools for a logic (continued):

- declare special notation on top of a *deep* or *shallow* embedding

  pro: can immitate the look of desired logic

  con: parser often limits how close we can get to desired logic

- override tactic notation (thanks to Lean!)

  pro: can allow the user to reason in terms of the logic rather than in terms of its encoding without implementing a new prover from scratch

  con: —

- define top-level syntax (thanks to Lean!)

  pro: can embed a complete language inside a prover

  con: too awesome

How to ... (continued)

The many possible approaches to building tools for a logic (continued):

- declare special notation on top of a *deep* or *shallow* embedding

  pro: can immitate the look of desired logic

  con: parser often limits how close we can get to desired logic

- override tactic notation (thanks to Lean!)

  pro: can allow the user to reason in terms of the logic rather than in terms of its encoding without implementing a new prover from scratch

  con: —

- define top-level syntax (thanks to Lean!)

  pro: can embed a complete language inside a prover

  con: too awesome

How to ... (continued)

The many possible approaches to building tools for a logic (continued):

- declare special notation on top of a *deep* or *shallow* embedding

  pro: can immitate the look of desired logic

  con: parser often limits how close we can get to desired logic

- override tactic notation (thanks to Lean!)

  pro: can allow the user to reason in terms of the logic rather than in terms of its encoding without implementing a new prover from scratch

  con: —

- define top-level syntax (thanks to Lean!)

  pro: can embed a complete language inside a prover

  con: too awesome

Chosen approach

- write a *shallow* embedding in a proof assistant
- declare special notation
- override tactic notation

Chosen approach

- write a *shallow* embedding in a proof assistant
- declare special notation
- override tactic notation

Chosen approach

- write a *shallow* embedding in a proof assistant
- declare special notation
- override tactic notation

What is temporal logic?

Temporal logic makes time ubiquitous and implicit

- superset of first order logic
- add two modalities: $\Diamond$ and $\Box$
- $\Diamond P$, with P, a proposition means "at some point in the future, P will hold"
- $\Box P$, with P, a proposition means "from now on, P holds"

What is temporal logic?

Temporal logic makes time ubiquitous and implicit

- superset of first order logic
- add two modalities: $\Diamond$ and $\Box$
- $\Diamond P$, with P, a proposition means "at some point in the future, P will hold"
- $\Box P$, with P, a proposition means "from now on, P holds"

What is temporal logic?

Temporal logic makes time ubiquitous and implicit

- superset of first order logic
- add two modalities: $\Diamond$ and $\Box$
- $\Diamond P$, with P, a proposition means "at some point in the future, P will hold"
- $\Box P$, with P, a proposition means "from now on, P holds"

What is temporal logic?

Temporal logic makes time ubiquitous and implicit

- superset of first order logic
- add two modalities: $\Diamond$ and $\Box$
- $\Diamond P$, with P, a proposition means "at some point in the future, P will hold"
- $\Box P$, with P, a proposition means "from now on, P holds"

What is temporal logic?

Temporal logic makes time ubiquitous and implicit

- superset of first order logic
- add two modalities: $\Diamond$ and $\Box$
- $\Diamond P$, with P, a proposition means "at some point in the future, P will hold"
- $\Box P$, with P, a proposition means "from now on, P holds"

What is temporal logic good for?

Use: Specifying the desired behavior of concurrent / distributed programs under development.

$$Init \triangleq x = 0 \land y = 0$$

$$Next \triangleq \begin{array}{ll} & x' = x + 1 \land y' = y - 1 \\ \lor & x' = x - 2 \land y' = y + 2 \end{array}$$

$$Spec \triangleq Init \land \Box Next$$

$$Theorem : Spec \Rightarrow \Box(x + y = 0)$$

What is temporal logic good for?

Use: Specifying the desired behavior of concurrent / distributed programs under development.

$$Init \triangleq x = 0 \land y = 0$$

$$Next \triangleq \quad \begin{array}{l} \ x' = x + 1 \land y' = y - 1 \\ \lor\ x' = x - 2 \land y' = y + 2 \end{array}$$

$$Spec \triangleq Init \land \Box Next$$

$$Theorem : Spec \Rightarrow \Box(x + y = 0)$$

What is temporal logic good for?

Use: Specifying the desired behavior of concurrent / distributed programs under development.

$$Init \triangleq x = 0 \land y = 0$$

$$Next \triangleq \begin{array}{ll} & x' = x + 1 \land y' = y - 1 \\ \lor & x' = x - 2 \land y' = y + 2 \end{array}$$

$$Spec \triangleq Init \land \Box Next$$

$$Theorem : Spec \Rightarrow \Box(x + y = 0)$$

What is temporal logic good for?

Use: Specifying the desired behavior of concurrent / distributed programs under development.

$$Init \triangleq x = 0 \land y = 0$$

$$Next \triangleq \begin{array}{ll} & x' = x + 1 \land y' = y - 1 \\ \lor & x' = x - 2 \land y' = y + 2 \end{array}$$

$$Spec \triangleq Init \land \Box Next$$

$$Theorem : Spec \Rightarrow \Box(x + y = 0)$$

What is temporal logic good for?

Use: Specifying the desired behavior of concurrent / distributed programs under development.

$$
\begin{aligned}
Init &\triangleq x = 0 \land y = 0 \\
Next &\triangleq \begin{array}{ll} & x' = x + 1 \land y' = y - 1 \\ \lor & x' = x - 2 \land y' = y + 2 \end{array} \\
Spec &\triangleq Init \land \Box Next \\
Theorem : Spec &\Rightarrow \Box(x + y = 0)
\end{aligned}
$$

Shallow embedding

```
def tprop := ℕ → Prop -- ℕ is a discrete time

def entails (p q : tprop) : Prop :=
∀ i : ℕ, p i → q i
infix ` ⊢ `:53 := judgement -- \|-

def eventually (p : tprop) : tprop :=
λ i : ℕ, ∃ j : ℕ, p (i+j)
prefix `◊`:95 := eventually -- \di

def henceforth (p : tprop) : tprop :=
λ i : ℕ, ∀ j : ℕ, p (i+j)
prefix `□`:95 := henceforth -- \sqw
```

More notation

```
def t_and (p q : tprop) : tprop :=
λ i : ℕ, p i ∧ q i
prefix ` ⋀ `:95 := t_and -- \And, not \and

def t_or (p q : tprop) : tprop :=
λ i : ℕ, p i ∨ q i
prefix ` ⋁ `:95 := t_or -- \Or, not \or

def t_all {α} (P : α → tprop) : tprop :=
λ t : ℕ, ∀ x : α, P x t
notation `∀∀` binders `, ` r:(scoped P, t_all P) := r
```

Example: Proposition

```
p : α → tprop,
q : α → tprop
⊢ (∀∀ x : α, p x ⋀ q x) : tprop
```

Example: Proof

Available to step through at:
https://github.com/unitb/temporal-logic/blob/amsterdam-talk/src/temporal_logic/lemmas.lean#L469-L490

```
protected lemma leads_to_cancellation'
    {p q b r : tpred} {t : ℕ}
    (P₀ : t ⊨ p ~> q ⋁ b)
    (P₁ : t ⊨ q ~> r)
    : t ⊨ p ~> r ⋁ b :=
begin
  intros Δ h,
  have := P₀ _ h, clear h,
  cases this with Δ' h,
  cases h with h h,
  { rw add_assoc at h,
    specialize P₁ _ h,
    cases P₁ with Δ'' h, rw ← add_assoc at h,
    existsi (Δ' + Δ''), rw ← add_assoc,
    left, apply h },
  { existsi Δ', right, assumption },
end
```

Criticism

Let's step through a small part of the proof:

```
-- ...
{ rw add_assoc at h,
  specialize P₁ _ h,
  cases P₁ with Δ'' h,
  rw ← add_assoc at h,
  existsi (Δ' + Δ''),
  rw ← add_assoc,
  left, apply h },
-- ...
```

Proof Goal

Criticism

Let's step through a small part of the proof:

```
-- ...
{ rw add_assoc at h,
  specialize P₁ _ h,
  cases P₁ with Δ'' h,
  rw ← add_assoc at h,
  existsi (Δ' + Δ''),     ◁
  rw ← add_assoc,
  left, apply h },
-- ...
```

Proof Goal

$$
\begin{aligned}
& \mathtt{t}\ \Delta\ \Delta' : \mathbb{N}, \\
& \mathtt{h} : \mathtt{t} + (\Delta + \Delta') \models \mathtt{q}, \\
& \Delta'' : \mathbb{N}, \\
& \mathtt{h} : \mathtt{t} + \Delta + \Delta' + \Delta'' \models \mathtt{r} \\
\vdash\ & \mathtt{t} + \Delta \models \Diamond(\mathtt{r} \bigvee \mathtt{b})
\end{aligned}
$$

Criticism

Let's step through a small part of the proof:

```
-- ...
{ rw add_assoc at h,
  specialize P₁ _ h,
  cases P₁ with Δ'' h,
  rw ← add_assoc at h,
  existsi (Δ' + Δ''),
  rw ← add_assoc,          ◁
  left, apply h },
-- ...
```

Proof Goal

$t\ \Delta\ \Delta' : \mathbb{N},$
$h : t + (\Delta + \Delta') \models q,$
$\Delta'' : \mathbb{N},$
$h : t + \Delta + \Delta' + \Delta'' \models r$
$\vdash t + \Delta + (\Delta' + \Delta'') \models r \bigvee b$

Criticism

Let's step through a small part of the proof:

```
-- ...
{ rw add_assoc at h,
  specialize P₁ _ h,
  cases P₁ with Δ'' h,
  rw ← add_assoc at h,
  existsi (Δ' + Δ''),
  rw ← add_assoc,
  left, apply h },          ◁
-- ...
```

Proof Goal

$\mathtt{t}\ \Delta\ \Delta' : \mathbb{N},$
$\mathtt{h} : \mathtt{t} + (\Delta + \Delta') \models \mathtt{q},$
$\Delta'' : \mathbb{N},$
$\mathtt{h} : \mathtt{t} + \Delta + \Delta' + \Delta'' \models \mathtt{r}$
$\vdash \mathtt{t} + \Delta + \Delta' + \Delta'' \models \mathtt{r} \bigvee \mathtt{b}$

Improvement

https://github.com/unitb/temporal-logic/blob/amsterdam-talk/src/temporal_logic/lemmas.lean#L454-L467

```
protected lemma leads_to_cancellation {p q b r : tpred}
  (P₀ : Γ ⊢ p ~> q ⋁ b)
  (P₁ : Γ ⊢ q ~> r) :
  Γ ⊢ p ~> r ⋁ b :=
begin [temporal]
 unfold tl_leads_to in *,
 henceforth,
 intros h,
 have := P₀ h, clear h,
 eventually this,
 rw [eventually_or],
 cases this with h h,
 { left, apply P₁ h },
 { right, assumption },
end
```

Improvement

```
begin [temporal]    -- ← we're using a special tactic language
 unfold tl_leads_to in *,
 henceforth,
 intros h,
 have := P₀ h, clear h,
 eventually this,
 rw [eventually_or],
 -- ...
```

Improvement

```
begin [temporal]
  unfold tl_leads_to in *,
  henceforth,                ◁
  intros h,
  have := P₀ h, clear h,
  eventually this,
  rw [eventually_or],
  -- ...
```

Proof Goal

p q b r : tprop,
P_0 : □(p ⟶ ◇(q ⋁ b)),
P_1 : □(q ⟶ ◇r)
⊢ □(p ⟶ ◇(r ⋁ b))

Improvement

```
begin [temporal]
  unfold tl_leads_to in *,
  henceforth,
  intros h,                ◁
  have := P₀ h, clear h,
  eventually this,
  rw [eventually_or],
  -- ...
```

Proof Goal

$p\ q\ b\ r : \mathtt{tprop}$,
$\_\mathtt{inst\_1} : \mathtt{persistent}\ \Gamma$,
$P_0 : \Box(p \longrightarrow \Diamond(q \bigvee b))$,
$P_1 : \Box(q \longrightarrow \Diamond r)$
$\vdash p \longrightarrow \Diamond(r \bigvee b)$

Improvement

```
begin [temporal]
  unfold tl_leads_to in *,
  henceforth,
  intros h,
  have := P₀ h, clear h,   ◁
  eventually this,
  rw [eventually_or],
  -- ...
```

Proof Goal

$p\ q\ b\ r : \mathrm{tprop},$
$P_0 : \Box(p \longrightarrow \Diamond(q \bigvee b)),$
$P_1 : \Box(q \longrightarrow \Diamond r),$
$h : p$
$\vdash \Diamond(r \bigvee b)$

Improvement

```
begin [temporal]
  unfold tl_leads_to in *,
  henceforth,
  intros h,
  have := P₀ h, clear h,
  eventually this,          ◁
  rw [eventually_or],
  -- ...
```

Proof Goal

$\texttt{p q b r : tprop},$
$P_0 : \Box(p \longrightarrow \Diamond(q \bigvee b)),$
$P_1 : \Box(q \longrightarrow \Diamond r),$
$\texttt{this} : \Diamond(q \bigvee b)$
$\vdash \Diamond(r \bigvee b)$

Improvement

```
begin [temporal]
  unfold tl_leads_to in *,
  henceforth,
  intros h,
  have := P₀ h, clear h,
  eventually this,
  rw [eventually_or],        ◁
  -- ...
```

Proof Goal

p q b r : tprop,
P_0 : $\Box(p \longrightarrow \Diamond(q \bigvee b))$,
P_1 : $\Box(q \longrightarrow \Diamond r)$,
this : $q \bigvee b$
$\vdash \Diamond(r \bigvee b)$

Observe:

- Neither $t \models$ nor $\Gamma \vdash$ appear in the proof goal
- Temporal reasoning is limited to the tactics `henceforth` and `eventually`
- Time and time intervals are completely anonymous
- The goal (e.g. $\Diamond(r \bigvee b)$) is not a type; it is `tprop`

What's the trick?

Displayed proof state:

$$\begin{array}{l} \quad \mathtt{p\ q\ b\ r} : \mathtt{tprop}, \\ \quad \mathtt{P_0} : \Box(\mathtt{p} \longrightarrow \Diamond(\mathtt{q} \bigvee \mathtt{b})), \\ \quad \mathtt{P_1} : \Box(\mathtt{q} \longrightarrow \Diamond\mathtt{r}), \\ \quad \mathtt{this} : \mathtt{q} \bigvee \mathtt{b} \\ \vdash \Diamond(\mathtt{r} \bigvee \mathtt{b}) \end{array}$$

Internal proof state:

$$\begin{array}{l} \quad \Gamma\ \mathtt{p\ q\ b\ r} : \mathtt{tprop}, \\ \quad \mathtt{P_0} : \Gamma \vdash \Box(\mathtt{p} \longrightarrow \Diamond(\mathtt{q} \bigvee \mathtt{b})), \\ \quad \mathtt{P_1} : \Gamma \vdash \Box(\mathtt{q} \longrightarrow \Diamond\mathtt{r}), \\ \quad \mathtt{this} : \Gamma \vdash \mathtt{q} \bigvee \mathtt{b} \\ \vdash \Gamma \vdash \Diamond(\mathtt{r} \bigvee \mathtt{b}) \end{array}$$

What's the trick? (cont.)

Reasoning:

- In most lemmas, use a single Γ;
- use specialized lemmas for substituting Γ for Γ' and have the tactics apply them transparently;
- use function coercion so that $(\forall\forall\_, \_)$, $\_ \longrightarrow \_$ and $\Box(\_ \longrightarrow \_)$ will behave like the normal type theory $\rightarrow$ and Π

Highlights

- write a *shallow* embedding in a proof assistant;
- declare special notation;
- override tactic notation

Highlights (cont.)

Benefits: we can use a specialized logic in a context where

- others have proved advanced and not so advanced mathematical theorems;
- powerful automation is available;
- the prover subscribes to the small trusted kernel model