Categorial Logic in Lean

Jacob Neumann
University of Nottingham
TYPES 2023, Valencia, Spain
13 June 2023

Check out the website!

lean-catlogic.github.io

0 Completeness Proofs via LT Categories

Sketch of the idea

Proof Sketch To prove the soundness/completeness of a deductive calculus

1. Construct the Lindenbaum-Tarski category
   - Objects are (equiv. classes of) formulas, morphisms are 'lifted' from deductions
   - will have additional categorical structure corresponding to the logical structure of the theory
2. Give a natural bijection between models and and (structure-preserving) functors from LT-category into presheaf categories
3. Instantiate the bijection in 2 with the Yoneda embedding to get a *canonical model* which is *logically generic*

The Positive Propositional Calculus

top $\top$

$q \wedge p$

$p \to p$

implication

$p \wedge q$

q

p

$\top \to p$

$q \wedge q$

$p \wedge \top$

conjunction

atomic formula

The Positive Propositional Calculus

top $\top$

$q \land p$

$p \to p$

implication

$p \land q$

q

p

$\top \to p$

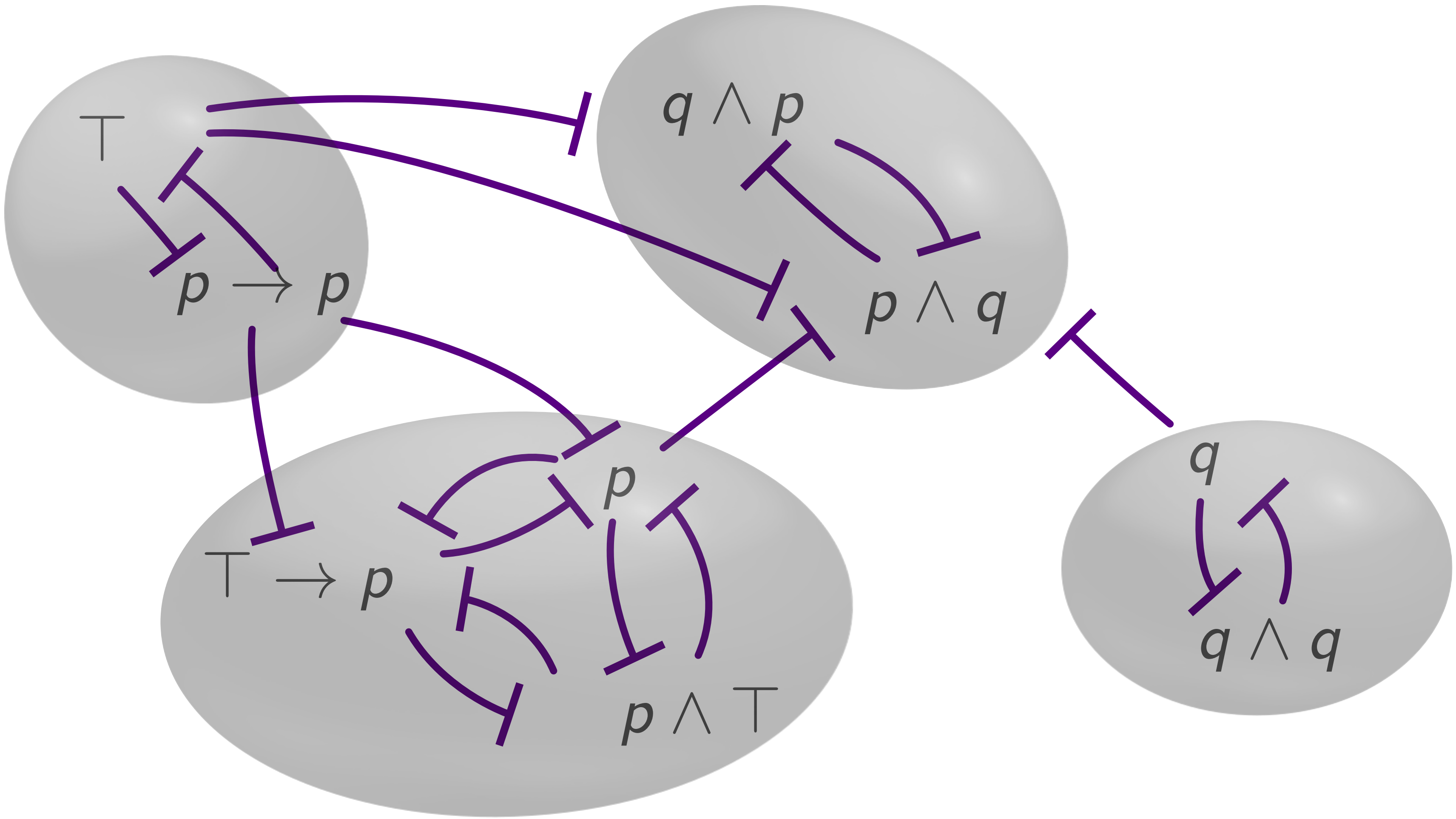
$p \land \top$

$q \land q$

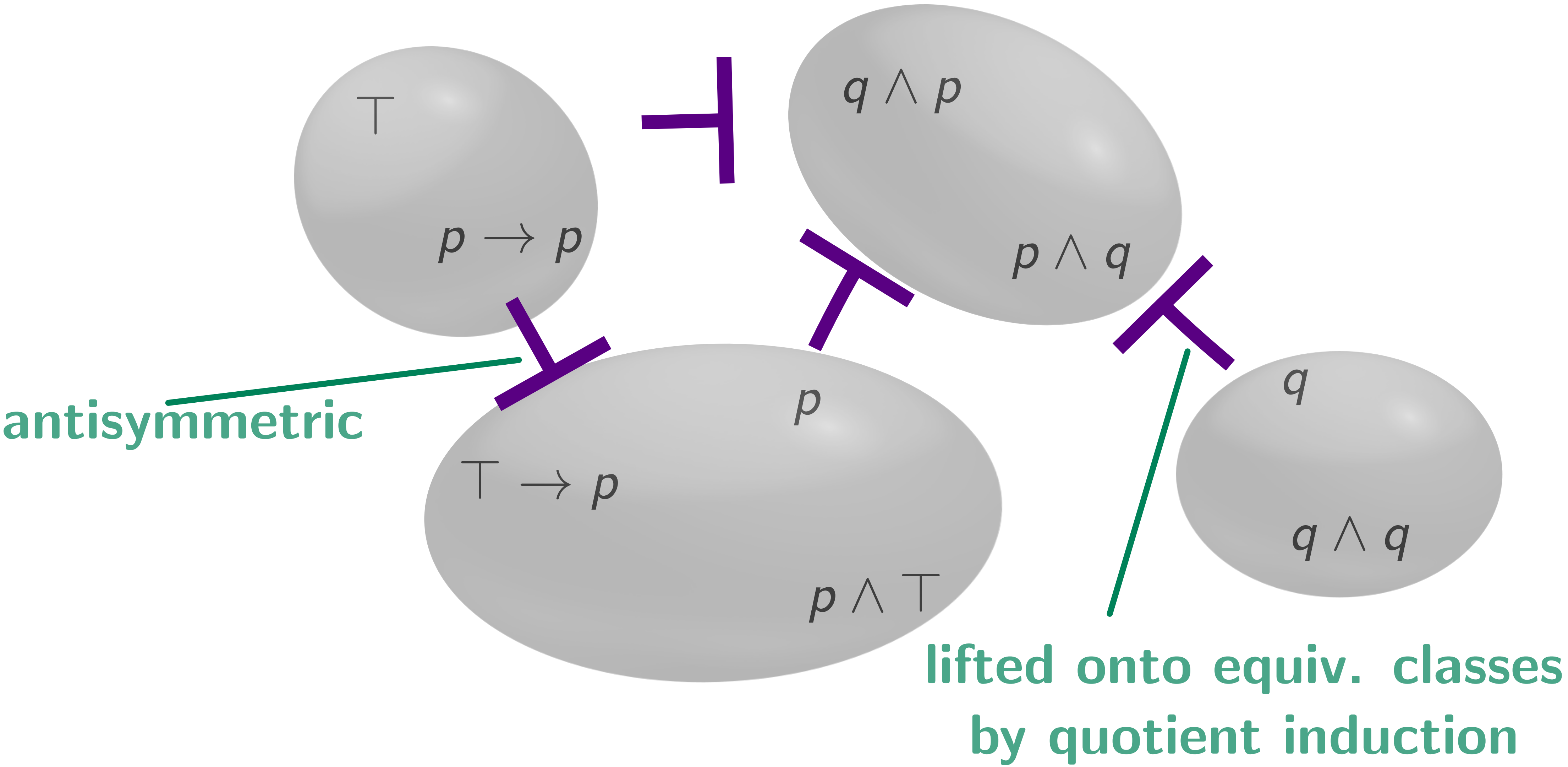
conjunction

atomic formula

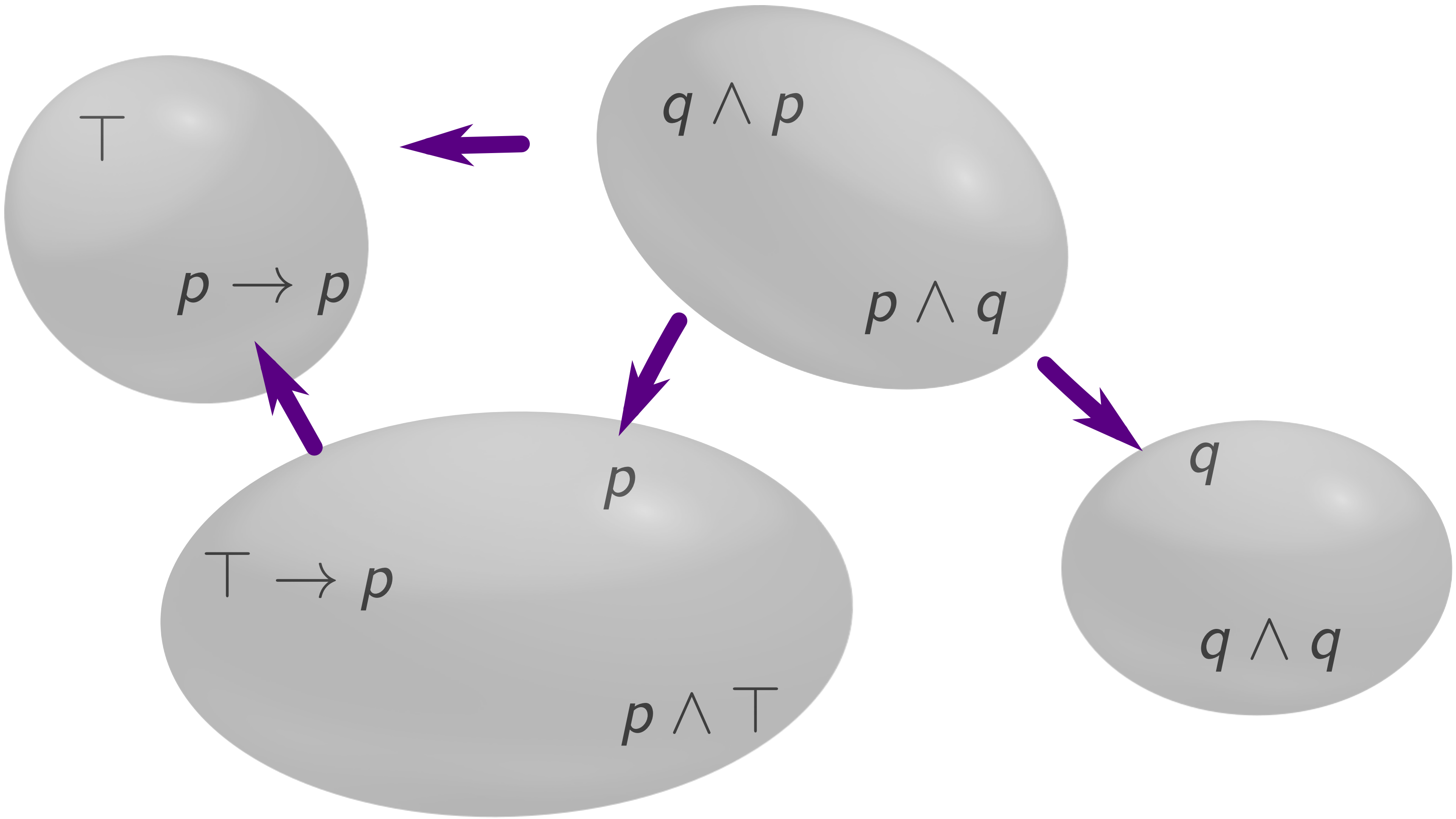
Quotient by inter-derivability and the syntactic category



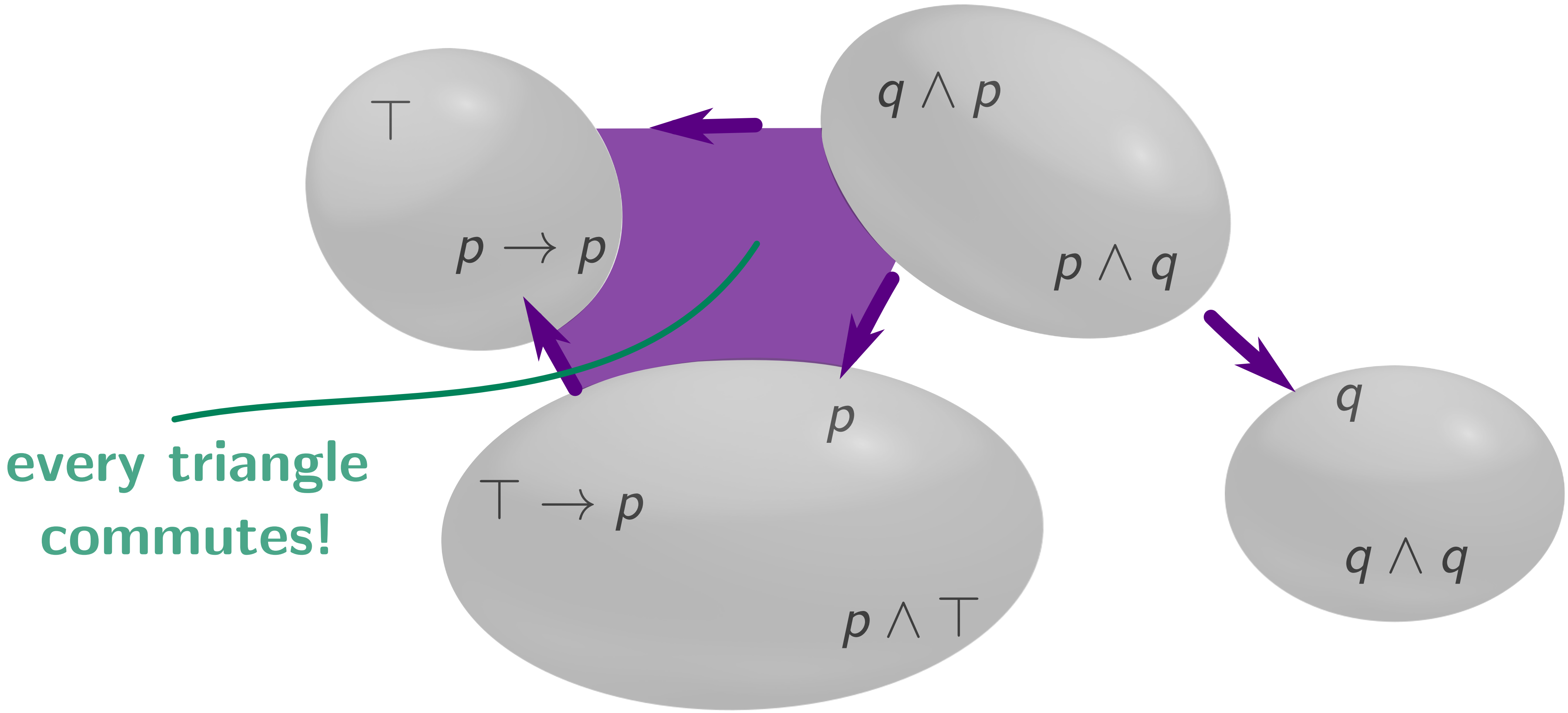
Quotient by inter-derivability and the syntactic category



Quotient by inter-derivability and the syntactic category



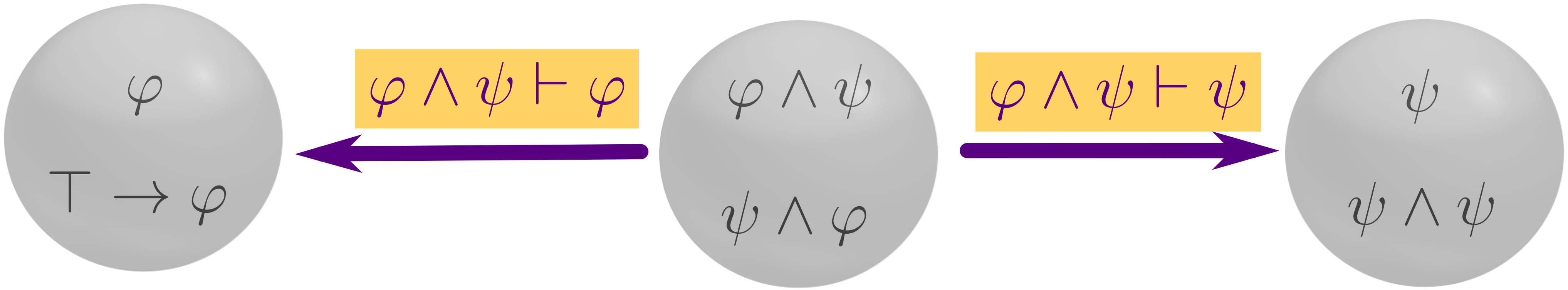
Quotient by inter-derivability and the syntactic category

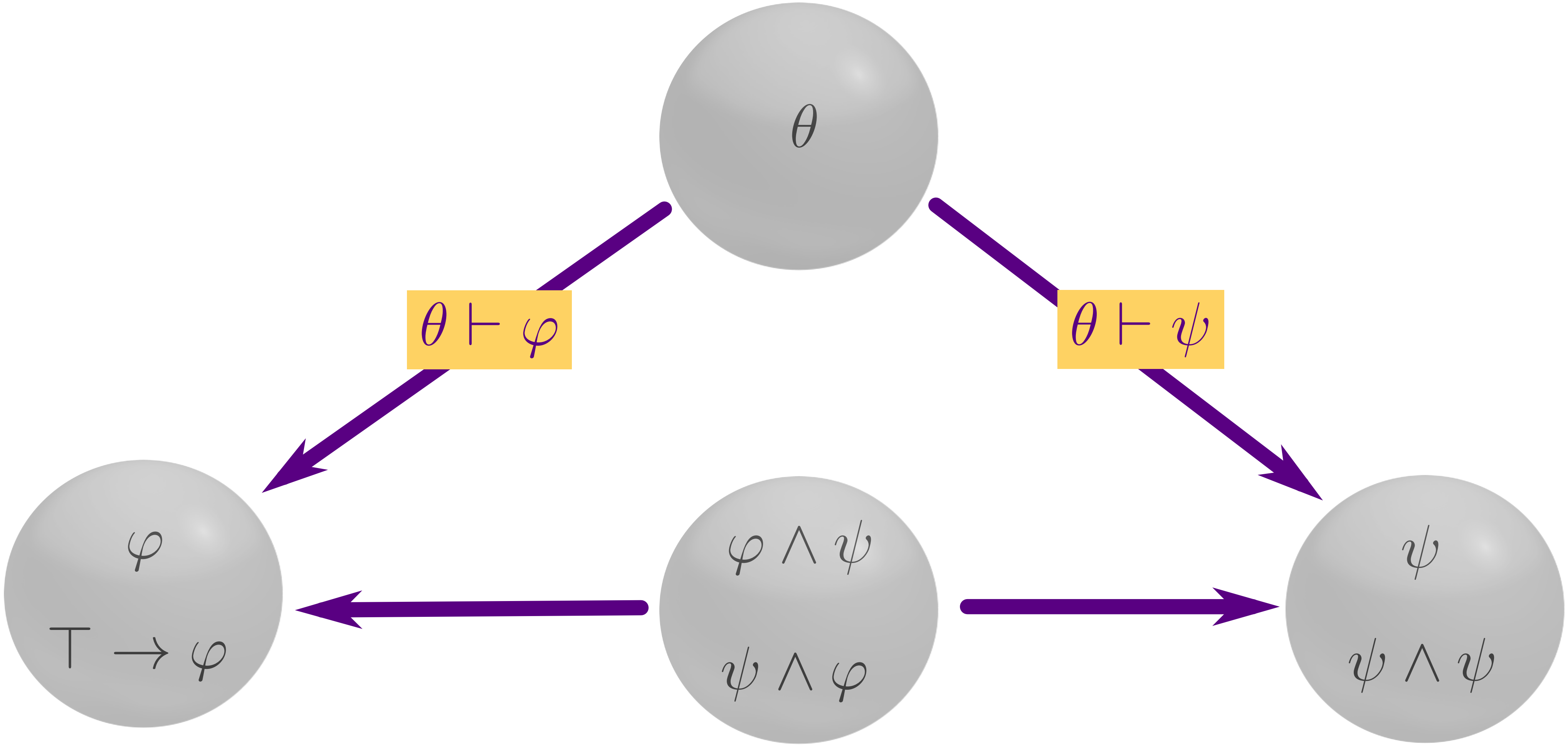


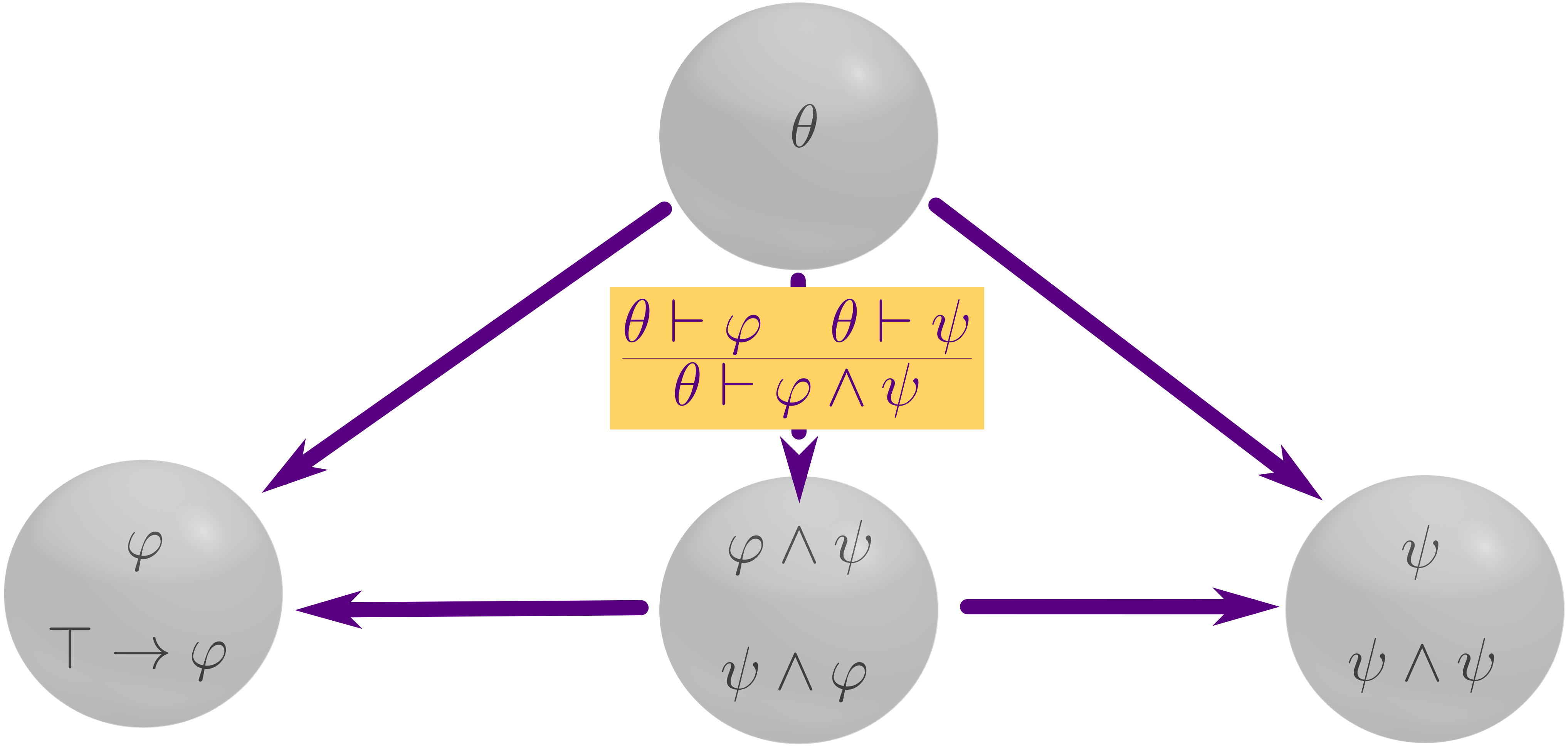
logical structure

lifts to

categorical structure


φ
$\top \to \varphi$
$\varphi \wedge \psi \vdash \varphi$
$\varphi \wedge \psi$
$\psi \wedge \varphi$
$\varphi \wedge \psi \vdash \psi$
ψ
$\psi \wedge \psi$


θ
$\theta \vdash \varphi$
$\theta \vdash \psi$
φ
$\top \to \varphi$
$\varphi \wedge \psi$
$\psi \wedge \varphi$
ψ
$\psi \wedge \psi$


θ
$\frac{\theta \vdash \varphi \quad \theta \vdash \psi}{\theta \vdash \varphi \wedge \psi}$
φ
$\top \to \varphi$
$\varphi \wedge \psi$
$\psi \wedge \varphi$
ψ
$\psi \wedge \psi$

Exercises:

- Verify that the equivalence class of $\top$ is the terminal object in the LT category
- Verify that these rules, when lifted onto the LT category, endow it with *exponentials*

$$\frac{\Phi \vdash \varphi \to \psi \quad \Phi \vdash \varphi}{\Phi \vdash \psi} \qquad \frac{\varphi \wedge \psi \vdash \theta}{\varphi \vdash \psi \to \theta}$$


Calculemus!

¡Formalicemos!

1 Formalization in Lean

The `lean-catLogic` project

- Content to formalize:
  - Awodey and Bauer's notes (awodey.github.io/catlog/notes), starting with Sect. 2.8
  - Awodey's Categorical Logic lectures, Proof and Computation Autumn School, Fischbachau, Germany, 2022 (awodey.github.io/fischbachau)
  - [MR95, HM92]
  - Eventually cover more of categorical logic...
  - Not aware of other formalizations of this material (?)
- Lean proof assistant [dMKA+15]
  - Lean 3 (current stable version), but may someday convert to Lean 4
  - Lean mathlib [mC20]
  - Tactic-based proofs, with utilities for custom tactics and metaprogramming in Lean itself

Agenda

- Original proof that LT category of the PPC is a CCC
  - Discuss shortcomings/limitations
- Optimizations (so far)


MONTH
DAY
YEAR
HOUR
MIN
JUN
13
2023
17
16
CURRENT TIME
1BEAF90BB0F8
DESTINATION COMMIT
MONTH
DAY
YEAR
HOUR
MIN
NOV
14
2022
16
06
DESTINATION TIME
git

Representing the proof calculus I

- Formulas are defined as an inductive type `PPC_form`

1beaf90 languages/PPC.lean [permalink]

```
inductive PPC_form : Type
  | top : PPC_form
  | var : ℕ → PPC_form
  | and : PPC_form → PPC_form → PPC_form
  | impl : PPC_form → PPC_form → PPC_form
```

- The `derives` relation is an inductively-defined relation between sets of formulas and formulas

1beaf90 proof/PPC_natDeduct.lean [permalink]

Representing the proof calculus II

```
  | and_intro {Φ} {φ ψ : PPC_form}
      : derives Φ φ → derives Φ ψ → derives Φ (φ & ψ)
  | and_eliml {Φ} {φ ψ : PPC_form}
      : derives Φ (φ & ψ) → derives Φ φ
  | and_elimr {Φ} {φ ψ : PPC_form}
      : derives Φ (φ & ψ) → derives Φ ψ
```

- Define ⊢ relation between formulas as singleton-`derives`

1beaf90 proof/PPC_natDeduct.lean [permalink]

```
infix `⊢`:80 := λ φ ψ, derives {φ} ψ
```

Quotienting and forming the LT category

- The inter-derivability equivalence relation
- `PPC_eq := PPC_form / ⊣⊢`

1beaf90 semantics/PPC_poset.lean [permalink]

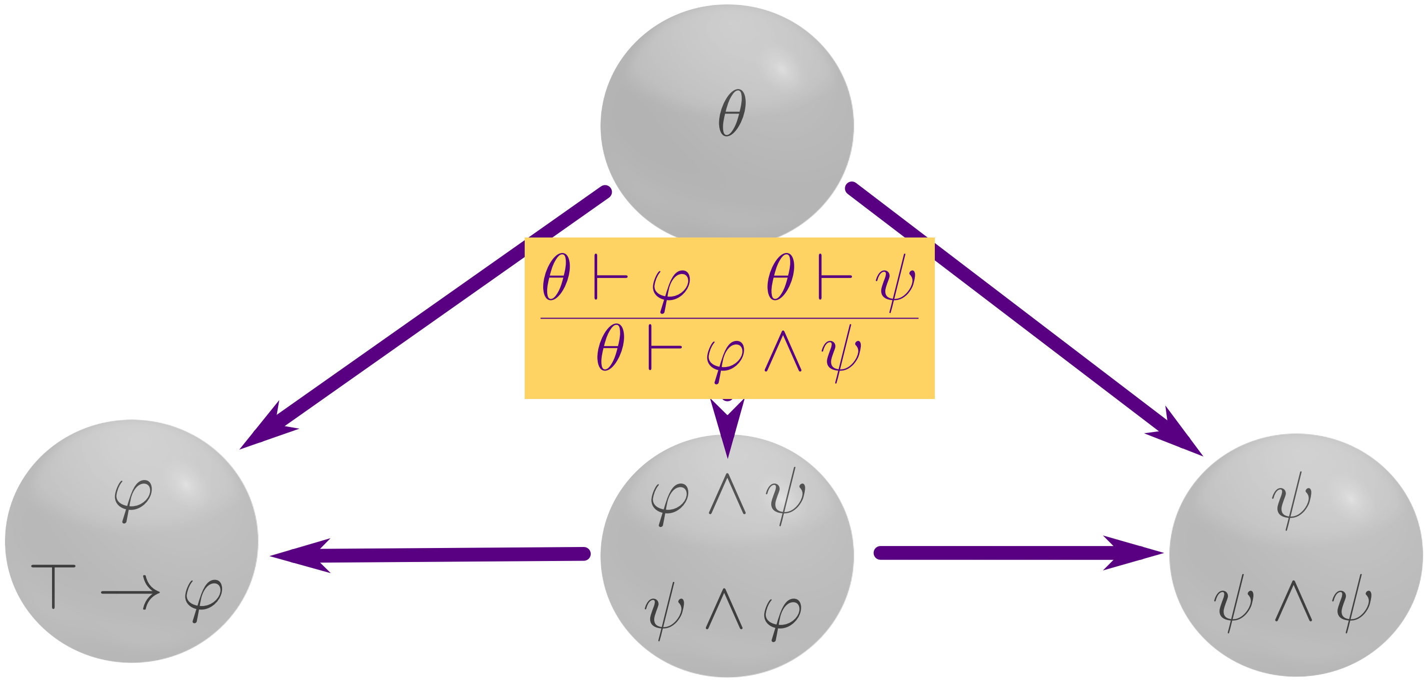
```
notation (name:=PPC_setoid.r) φ `⊣⊢` ψ :78 := PPC_setoid.r φ ψ

-- Quotienting PPC_form by inter-derivability
def PPC_eq : Type := quot (⊣⊢)
```

- Define the category:

1beaf90 semantics/PPC_syntacticCat.lean [permalink]

```
def ℂ_PPC : category (PPC_eq)
  := preorder.small_category PPC_eq
```

Proof it's a CCC



1beaf90 proof/PPC_natDeduct.lean [permalink]

```
  | and_intro {Φ} {φ ψ : PPC_form}
      : derives Φ φ → derives Φ ψ → derives Φ (φ & ψ)
```

Proof it's a CCC (cont.)

1beaf90 semantics/PPC_syntacticCat.lean [permalink]

```
  curry :=
    begin
      -- Actual construction
      assume X Y Z u,
      induction X with φ, induction Y with ψ, induction Z with θ,
      apply ℂ_PPC_hom,
      have h : (φ&ψ) ⊢ θ,
      exact (Exists.fst (ℂ_PPC_full u)),
      apply derives.impl_intro,
      apply and_Hyp_union,
      exact h,
      -- Proving this respects ⊣⊢
      apply funext, assume _, apply ℂ_PPC_thin,
      apply funext, assume _, apply ℂ_PPC_thin,
      apply funext, assume _, apply ℂ_PPC_thin,
```

Observation: Content is buried under all the boilerplate

Where's the actual content?

1beaf90 semantics/PPC_syntacticCat.lean [permalink]

```
pair :=
  begin
    -- Actual construction
    assume X Y Z f g,
    induction X with φ, induction Y with ψ, induction Z with χ,
    let h : χ ⊢ φ := le_of_hom f,
    let h' : χ ⊢ ψ := le_of_hom g,
    apply ℂ_PPC_hom,
    apply derives.and_intro,
    exact h, exact h',
    -- Proving this respects ⊣⊢
    apply funext, assume _, apply funext, assume _, apply ℂ_PPC_thin,
    apply funext, assume _, apply ℂ_PPC_thin,
    apply funext, assume _, apply ℂ_PPC_thin,
  end,
```

GENERIC

GENERIC

Proof it's a CCC (cont.)

1beaf90 semantics/PPC_syntacticCat.lean [permalink]

```
  curry :=
    begin
      -- Actual construction
      assume X Y Z u,
      induction X with φ, induction Y with ψ, induction Z with θ,
      apply ℂ_PPC_hom,
      have h : (φ&ψ) ⊢ θ,
      exact (Exists.fst (ℂ_PPC_full u)),
      apply derives.impl_intro,
      apply and_Hyp_union,
      exact h,
      -- Proving this respects ⊣⊢
      apply funext, assume _, apply ℂ_PPC_thin,
      apply funext, assume _, apply ℂ_PPC_thin,
      apply funext, assume _, apply ℂ_PPC_thin,
```

GENERIC

GENERIC

Let’s do better

2 The LiftT tactic

- Basic Lean proof: use tactics to solve goals, set new goals, etc.
- More advanced: use tactic combinators to automate repetitive aspects
- Even more advanced: define custom tactics using the `meta def` keyword
- Even more advanced: programming with the full power of the `tactic` monad

Recall: lifting boilerplate

1beaf90 semantics/PPC_syntacticCat.lean [permalink]

```
pair :=
  begin
    -- Actual construction
    assume X Y Z f g,
    induction X with φ, induction Y with ψ, induction Z with χ,
    let h : χ ⊢ φ := le_of_hom f,
    let h' : χ ⊢ ψ := le_of_hom g,
    apply ℂ_PPC_hom,
    apply derives.and_intro,
    exact h, exact h',
    -- Proving this respects ⊣⊢
    apply funext, assume _, apply funext, assume _, apply ℂ_PPC_thin,
    apply funext, assume _, apply ℂ_PPC_thin,
    apply funext, assume _, apply ℂ_PPC_thin,
  end,
```

GENERIC

GENERIC

LiftT

$$\texttt{LiftT}: \texttt{tactic unit} \to \texttt{tactic unit}$$

```
apply derives.and_intro,
```

tactic performing the relevant action on deductions

LiftT: $\texttt{tactic unit} \to \texttt{tactic unit}$

```
apply derives.and_intro,
```

```
-- Actual construction
assume X Y Z f g,
induction X with φ, induction Y with ψ, induction Z with χ,
let h : χ ⊢ φ := le_of_hom f,
let h' : χ ⊢ ψ := le_of_hom g,
apply ℂ_PPC_hom,
apply derives.and_intro,
exact h, exact h',
-- Proving this respects ⊣⊢
apply funext, assume _, apply funext, assume _, apply ℂ_PPC_thin,
apply funext, assume _, apply ℂ_PPC_thin,
apply funext, assume _, apply ℂ_PPC_thin,
```

tactic constructing an operation on the LT category

The finished proof

bf4eda9 semantics/syntacticCat_cartesian.lean [permalink]

```
instance syn_FP_cat {Form : Type} [And : has_and Form] : FP_cat (Form _eq) :=
{
  unit := syn_obj And.top,
  term := by LiftT `[ apply And.truth ],
  unit_η := λ X f, by apply thin_cat.K,
  prod := and_eq,
  pr1 := by LiftT `[ apply And.and_eliml ],
  pr2 := by LiftT `[ apply And.and_elimr ],
  pair := by LiftT `[ apply And.and_intro ],
  prod_β1 := λ X Y Z f g, by apply thin_cat.K,
  prod_β2 := λ X Y Z f g, by apply thin_cat.K,
  prod_η :=  λ X Y, by apply thin_cat.K
}
```

The finished proof (cont.)

bf4eda9 semantics/syntacticCat_cartesian.lean [permalink]

```
instance syn_CC_cat {Form : Type} [Impl : has_impl Form] : CC_cat (Form _eq) :=
{
  exp := impl_eq,
  eval := by LiftT `[ apply cart_x.modus_ponens ],
  curry := by LiftT `[ apply cart_x.impl_ε],
  curry_β := λ {X Y Z} u, by apply thin_cat.K,
  curry_η := λ {X Y Z} v, by apply thin_cat.K,
}
```

What is LiftT doing?

lean-catlogic.github.io/docs/semantics/synCat_tactics

LiftT's agenda:

1. Introduce necessary variables, apply quotient induction, turn into a deduction problem
2. Apply input tactic
3. Clean up any auxiliary goals created in the previous step
4. Use the fact that the LT category is a poset to prove the coherences required by quotient induction

```
meta def LiftT
  (debugMode : parse (optional $ tk "?"))
  (debugPerformTac : parse (optional $ tk "!"))
  (debugCleanup : parse (optional $ tk "!"))
  (T : tactic unit)
  : tactic unit :=
do
  let proceedLevel :=
    match (debugMode,debugPerformTac,debugCleanup) with
    | (none,_,_) := 4 -- LiftT    invoked: do everything
    | (_,none,_) := 1 -- LiftT?   invoked: just do the assume's and syn_hom
    | (_,_,none) := 2 -- LiftT?!  invoked: also apply the tactic
    | _ := 3          -- LiftT?!! invoked: also do the first stage of cleanup
    end,
```

```
    -- Count & print how many objects and morphisms to assume
    (numobjs,nummor) ← doCount none,
    /- Assume objects and morphisms,
      - use induction to get that every object is of the form {|φ|} for some φ
  :Form
      - use syn_hom_inv to turn every assumed morphism into a derivation -/
    repeat_assume_induct (gen_nameList `φ_ numobjs),
    repeat_assume_replace `synCat.syn_hom_inv (gen_nameList `f_ nummor),
    -- Turn the synCat hom goal to a derivation goal
    applyc `synCat.syn_hom,
    trace_goal "MAIN GOAL",
  when (proceedLevel > 1) $ do
    pre_goal_count ← count_goals,
    -- Apply the input tactic
    T,
    -- Difference in goals
    post_goal_count ← count_goals,
```

```
    let relGoals : nat :=
      if pre_goal_count > post_goal_count
      then 0
      else (post_goal_count - pre_goal_count) + 1,
    trace_goals relGoals "POST-TACTIC GOALS",
  when (proceedLevel > 2) $ do
    -- Eliminate other goals (first stage of cleanup)
    iterate (
      (applyc `deduction_basic.derive_refl)
      <|> assumption
    ),
    trace_all_goals "CLEANUP GOALS",
  when (proceedLevel > 3) $ do
    -- Prove the coherences
    thin_cat.by_thin
```

Future directions

- Formalize the rest of the Kripke Completeness proof
- Adjacent topics
  - Stone Duality
  - IPL, Beth models, and Heyting Algebras
  - Topological semantics
  - Lambda Calculus
  - Propositional modal logic
- Main topics of categorical logic
  - Lawvere Algebraic Theories
  - Predicate Logic
- Improve LiftT's operation
- Use mathlib in more places

[dMKA+15] Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris Van Doorn, and Jakob von Raumer.
The lean theorem prover (system description).
In *Automated Deduction-CADE-25: 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings 25*, pages 378–388. Springer, 2015.

[HM92] Victor Harnik and Michael Makkai.
Lambek's categorical proof theory and läuchli's abstract realizability.
The Journal of symbolic logic, 57(1):200–230, 1992.

[mC20] The mathlib Community.
The lean mathematical library.
In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2020, page 367–381, New York, NY, USA, 2020. Association for Computing Machinery.

[MR95] Michael Makkai and Gonzalo E. Reyes.
Completeness results for intuitionistic and modal logic in a categorical setting.
Annals of Pure and Applied Logic, 72(1):25–101, 1995.

Thank You!!

lean-catlogic.github.io

Diamond modality

bf4eda9 deduction/deduction_monadic.lean [permalink]

```
class has_diamond (Form : Type) extends has_struct_derives Form :=
  (diamond : Form → Form)
  (dmap  : ∀ {Φ : Hyp}{φ ψ : Form},
    derives (insert φ Φ) ψ → derives (insert (diamond φ) Φ) (diamond ψ))
  (dpure : ∀ {Φ : Hyp} {φ : Form},
    derives Φ φ → derives Φ (diamond φ))
  (djoin : ∀ {Φ : Hyp} {φ : Form},
    derives Φ (diamond (diamond φ)) → derives Φ (diamond φ))

notation (name:= has_diamond.diamond) `◇`:81 φ := has_diamond.diamond φ
```

Gives rise to a monad!

bf4eda9 semantics/syntacticCat_monadic.lean [permalink]

```
def diamond_monad {Form : Type} [Diam : has_diamond Form] :
  category_theory.monad (Form _eq) :=
{
  obj := diamond_eq,
  map := by LiftT `[ apply Diam.dmap ],
  η'  := ⟨
          by LiftT `[ apply Diam.dpure ],
          λ X Y f, by apply thin_cat.K,
         ⟩,
  μ' :=  ⟨
          by LiftT `[ apply Diam.djoin ],
          λ X Y f, by apply thin_cat.K,
         ⟩,
}
```