

# Logical Relations for a Logical Framework

sketching [RS13] by Florian Rabe and Kristina Sojakova

Navid Roux<sup>1</sup>

KWARC Seminar  
Computer Science, FAU Erlangen-Nürnberg

2021-01-27



This work is licensed under a “CC BY-SA 4.0” license.

---

<sup>1</sup><https://orcid.org/0000-0002-8348-2441>

1. Worked with logrels last 4 months
2. in master's project, thesis, seminar
3. Complicated topic; talk can only scratch surface
4. Still: let us learn something together

# Overview of this Talk

## Logical Relations

class of proof methods applicable on many type theories

## Logical Framework

a uniform presentation language to formalize logics/type theories

## Logical Relations for a Logical Framework

a uniform notion of “proof by logical relations” in the setting of LF

1. In many type theories, meta theorems can be proved by a “proof by logical relation”
2. e.g. prove SN in STLC, i.e., for well-typed terms beta reduction terminates
3. **informal class**; different type theories, different flavors, tweaks
4. Do **not confuse “logical”** adjectives.
5. Question: can we have a uniform notion of logrels just in the setting of LF that is type theory-agnostic?
6. Answer is yes, this talk gives a very rough sketch.

## Highlevel Motivation

To formalize a type theory, we have to formalize

- syntax
- typing rules
- operational semantics
- *and* meta theorems

these three already tedious enough

Meta theorems often employ a “proof by logical relation”.  
Formalization entails

w/o this work

- formalizing specific theory of logrels
- limited representation & tool support

w/ this work +  $\varepsilon^{-1}$

bold statement, ignorant of literature  
but helps beginners get the idea

1. general knowledge representation and processing is what kwarc does after all

## Highlevel Motivation

To formalize a type theory, we have to formalize

- syntax
- typing rules
- operational semantics
- *and* meta theorems

these three already tedious enough

Meta theorems often employ a “proof by logical relation”.  
Formalization entails

w/o this work

- formalizing specific theory of logrels
- limited representation & tool support

w/ this work +  $\varepsilon^{-1}$

- **instantiation of uniform notion** of logrels
- general representation & tool support

bold statement, ignorant of literature  
but helps beginners get the idea

1. general knowledge representation and processing is what kwarc does after all

## Required background knowledge:

- simply typed lambda calculus
- how to formalize things in a LF

e.g. as in  $\text{MMT}$

## Goals:

- 1 Introduce logical relations
- 2 Recap logical frameworks, esp.  $\text{MMT/LF}$
- 3 Represent toy logical relations in  $\text{MMT/LF}$

## Next Steps

### Logical Relations

class of proof methods applicable on many type theories

### Logical Framework

a uniform presentation language to formalize logics/type theories

### Logical Relations for a Logical Framework

a uniform notion of “proof by logical relations” in the setting of LF

# What are Logical Relations?

An (informal) **class of proof methods** used to prove

- strong normalization
- type safety
- program equivalence
  - correctness
  - theorems-for-free
  - security-typed languages

no infinite evaluation path  $t_1 \rightsquigarrow t_2 \rightsquigarrow \dots$

$$t : T \text{ and } t \rightsquigarrow^* t' \Rightarrow t' : T$$

e.g. of optimizations

terms of  $\forall \alpha. \alpha \rightarrow \alpha$  are the identity  
variables of “sensitive” type don’t leak

1. Next, to get a rough idea of what logrels are, let us sketch SN of STLC
2. This will be technical, but necessary, in my opinion, to understand why we care for logrels anyway.
3. Later, when representing in a LF, we will use much simpler examples.

# Simply-Typed Lambda Calculus (STLC)

## Definition

### Syntax:

$t$	$::=$	$x \mid \lambda x. t \mid s t$	terms
$T$	$::=$	$B \mid T_1 \rightarrow T_2$	types
$\Gamma$	$::=$	$\emptyset \mid \Gamma, x : T$	contexts

### Operational Semantics:

$$\frac{t \rightsquigarrow t'}{\lambda x. t \rightsquigarrow \lambda x. t'} \quad \frac{}{(\lambda x. s) t \rightsquigarrow s[x \mapsto t]} \quad \frac{s \rightsquigarrow s'}{s t \rightsquigarrow s' t} \quad \frac{t \rightsquigarrow t'}{s t \rightsquigarrow s t'}$$

$\mathcal{SN} = \{t \mid \nexists \text{ infinite eval. path } t = t_1 \rightsquigarrow t_2 \rightsquigarrow \dots\}$  set of strongly normalizing terms

## Theorem (Strong Normalization)

$$\Gamma \vdash t : T \implies t \in \mathcal{SN}$$



### Theorem (Strong Normalization)

$$\Gamma \vdash t : T \implies t \in \mathcal{SN}$$

### Proof Attempt: by induction on $t$

- case  $x$ : trivial
- case  $\lambda x. s$ : invert typing:



1. What if  $s$  is lambda abstraction?
2. A-priori unclear if beta-reduction with  $t$  doesn't destroy SN-ness of lambda abstraction.

## Theorem (Strong Normalization)

$$\Gamma \vdash t : T \implies t \in \mathcal{SN}$$

### Proof Attempt: by induction on $t$

- case  $x$ : trivial
- case  $\lambda x. s$ : invert typing:

$$\frac{\Gamma, x : T_1 \vdash s : T_2}{\Gamma \vdash \lambda x. s : T_1 \rightarrow T_2}$$

Together with IH yields  $s \in \mathcal{SN}$ , hence  $\lambda x. s \in \mathcal{SN}$ .

1. What if  $s$  is lambda abstraction?
2. A-priori unclear if beta-reduction with  $t$  doesn't destroy SN-ness of lambda abstraction.

## Theorem (Strong Normalization)

$$\Gamma \vdash t : T \implies t \in \mathcal{SN}$$

## Proof Attempt: by induction on $t$

- case  $x$ : trivial
- case  $\lambda x. s$ : invert typing: [...]
- case  $s \ t$ :



1. What if  $s$  is lambda abstraction?
2. A-priori unclear if beta-reduction with  $t$  doesn't destroy SN-ness of lambda abstraction.

## Theorem (Strong Normalization)

$$\Gamma \vdash t : T \implies t \in \mathcal{SN}$$

### Proof Attempt: by induction on $t$

- case  $x$ : trivial
- case  $\lambda x. s$ : invert typing: [...]
- case  $s t$ : invert typing:



$$\frac{\Gamma \vdash s : T_1 \rightarrow T_2 \quad \Gamma \vdash t : T_1}{\Gamma \vdash s t : T_2}$$

Together with IH yields  $s, t \in \mathcal{SN}$ ; but goal is  $s t \in \mathcal{SN}$ !

what if  $s? \lambda x. \dots$ ?

1. What if  $s$  is lambda abstraction?
2. A-priori unclear if beta-reduction with  $t$  doesn't destroy SN-ness of lambda abstraction.

## Theorem (Strong Normalization)

$$\Gamma \vdash t : T \implies t \in \mathcal{SN}$$

### Proof Attempt: by induction on $t$

- case  $x$ : trivial ✓
- case  $\lambda x. s$ : invert typing: [...] ✓
- case  $s t$ : invert typing: ✗

$$\frac{\Gamma \vdash s : T_1 \rightarrow T_2 \quad \Gamma \vdash t : T_1}{\Gamma \vdash s t : T_2}$$

Together with IH yields  $s, t \in \mathcal{SN}$ ; but goal is  $s t \in \mathcal{SN}$ !

#### Ideas:

- strengthen IH-condition on  $s$ :

$$s \in \mathcal{SN} \text{ and } s t \in \mathcal{SN} \text{ for arbitrary terms } t \text{ with } t \in \mathcal{SN}$$

- but only on those  $s$  of function type

1. What if  $s$  is lambda abstraction?
2. A-priori unclear if beta-reduction with  $t$  doesn't destroy SN-ness of lambda abstraction.

## Strengthening the Induction by a Logical Relation

This variant was too weak:

Failed Attempt (SN, naive formulation)

$$\Gamma \vdash t : T \implies t \in \mathcal{SN}$$

Let's try:

Attempt (SN, with naive logrel)

$$\Gamma \vdash t : T \implies P_T(t) \quad \text{where} \quad \left\{ \begin{array}{l} P_B(b) := b \in \mathcal{SN} \\ P_{T_1 \rightarrow T_2}(s) := s \in \mathcal{SN} \wedge \forall t : T_1. P_{T_1}(t) \Rightarrow P_{T_2}(s \ t) \end{array} \right\}$$

- For every type  $T$ , specify a relation  $P_T(-)$  on terms of type  $T$ .
- If that worked, strong normalization would be a corollary.

### Attempt (SN, with naive logrel)

$$\Gamma \vdash t : T \implies P_T(t) \quad \text{where} \quad \left\{ \begin{array}{l} P_B(b) := b \in \mathcal{SN} \\ P_{T_1 \rightarrow T_2}(s) := s \in \mathcal{SN} \wedge \forall t : T_1. P_{T_1}(t) \Rightarrow P_{T_2}(s \ t) \end{array} \right\}$$

### Proof Attempt: by induction on $t$

- case  $x$ : subtle, but doable



1. logrels usually closed under alpha and beta equivalence

### Attempt (SN, with naive logrel)

$$\Gamma \vdash t : T \implies P_T(t) \quad \text{where} \quad \left\{ \begin{array}{l} P_B(b) := b \in \mathcal{SN} \\ P_{T_1 \rightarrow T_2}(s) := s \in \mathcal{SN} \wedge \forall t : T_1. P_{T_1}(t) \Rightarrow P_{T_2}(s \ t) \end{array} \right\}$$

### Proof Attempt: by induction on $t$

- case  $x$ : subtle, but doable ✓

- case  $s \ t$ : invert typing and use IHs: 
$$\frac{\Gamma \vdash s : T_1 \rightarrow T_2 \quad \Gamma \vdash t : T_1}{\Gamma \vdash s \ t : T_2}$$
 ✓

1. logrels usually closed under alpha and beta equivalence



### Attempt (SN, with naive logrel)

$$\Gamma \vdash t : T \implies P_T(t) \quad \text{where} \quad \left\{ \begin{array}{l} P_B(b) := b \in \mathcal{SN} \\ P_{T_1 \rightarrow T_2}(s) := s \in \mathcal{SN} \wedge \forall t : T_1. P_{T_1}(t) \Rightarrow P_{T_2}(s \ t) \end{array} \right\}$$

### Proof Attempt: by induction on $t$

- case  $x$ : subtle, but doable ✓

- case  $s \ t$ : invert typing and use IHs: 
$$\frac{\Gamma \vdash s : T_1 \rightarrow T_2 \quad \Gamma \vdash t : T_1}{\Gamma \vdash s \ t : T_2} \quad \checkmark$$

- case  $\lambda x. s$ :

1. logrels usually closed under alpha and beta equivalence

### Attempt (SN, with naive logrel)

$$\Gamma \vdash t : T \implies P_T(t) \quad \text{where} \quad \left\{ \begin{array}{l} P_B(b) := b \in \mathcal{SN} \\ P_{T_1 \rightarrow T_2}(s) := s \in \mathcal{SN} \wedge \forall t : T_1. P_{T_1}(t) \Rightarrow P_{T_2}(s \ t) \end{array} \right\}$$

### Proof Attempt: by induction on $t$

- case  $x$ : subtle, but doable ✓

- case  $s \ t$ : invert typing and use IHs: 
$$\frac{\Gamma \vdash s : T_1 \rightarrow T_2 \quad \Gamma \vdash t : T_1}{\Gamma \vdash s \ t : T_2}$$
 ✓

- case  $\lambda x. s$ : invert typing:

$$\frac{\Gamma, x : T_1 \vdash s : T_2}{\Gamma \vdash \lambda x. s : T_1 \rightarrow T_2}$$

- $(\lambda x. s) \in \mathcal{SN}$ : trivial

1. logrels usually closed under alpha and beta equivalence

## Attempt (SN, with naive logrel)

$$\Gamma \vdash t : T \implies P_T(t) \quad \text{where} \quad \left\{ \begin{array}{l} P_B(b) := b \in \mathcal{SN} \\ P_{T_1 \rightarrow T_2}(s) := s \in \mathcal{SN} \wedge \forall t : T_1. P_{T_1}(t) \Rightarrow P_{T_2}(s \ t) \end{array} \right\}$$

## Proof Attempt: by induction on $t$

- case  $x$ : subtle, but doable ✓

- case  $s \ t$ : invert typing and use IHs: 
$$\frac{\Gamma \vdash s : T_1 \rightarrow T_2 \quad \Gamma \vdash t : T_1}{\Gamma \vdash s \ t : T_2}$$
 ✓

- case  $\lambda x. s$ : invert typing: ✗

$$\frac{\Gamma, x : T_1 \vdash s : T_2}{\Gamma \vdash \lambda x. s : T_1 \rightarrow T_2}$$

- $(\lambda x. s) \in \mathcal{SN}$ : trivial
- $\forall t : T_1. P_{T_1}(t) \Rightarrow P_{T_2}((\lambda x. s) \ t)$ : get stuck at  $P_{T_2}((\lambda x. s) \ t) \stackrel{?}{=} P_{T_2}(s[x \mapsto t])$

**We deviate from the IH on  $s$  *just* by a substitution!**

1. logrels usually closed under alpha and beta equivalence

## Strengthening the Induction by a Logical Relation II

1. Recover SN by identity substitution.

### Failed Attempt (SN, naive formulation)

$$\Gamma \vdash t : T \implies t \in \mathcal{SN}$$

### Failed Attempt (SN, with naive logrel)

$$\Gamma \vdash t : T \implies P_T(t) \quad \text{where} \quad \left\{ \begin{array}{l} P_B(b) := b \in \mathcal{SN} \\ P_{T_1 \rightarrow T_2}(s) := s \in \mathcal{SN} \wedge \forall t : T_1. P_{T_1}(t) \Rightarrow P_{T_2}(s \ t) \end{array} \right\}$$

This goes through:

### Theorem (SN, with logrel)

$$\Gamma \vdash t : T \wedge P_\Gamma(\gamma) \implies P_T(t\gamma) \quad \left\{ \begin{array}{l} P_B(b) := b \in \mathcal{SN} \\ P_{T_1 \rightarrow T_2}(s) := s \in \mathcal{SN} \wedge \forall t : T_1. P_{T_1}(t) \Rightarrow P_{T_2}(s \ t) \\ P_\Gamma(\gamma) := \forall (x : T) \in \Gamma. P_T(x\gamma) \end{array} \right\}$$

# Logical Relations: Summary

## Motivation:

### Theorem (Strong Normalization)

$$\Gamma \vdash t : T \implies t \in \mathcal{SN}$$

- Induction on  $t$  failed: IH too weak
- Needed to **strengthen IH based on type of  $t$**

## Solution: a “proof by logical relation”

applicable in many type theories

- 1 Define a logical relation  $P_{-}(-)$ :

for every type  $T$ ,  $P_T(-)$  is a relation on terms of type  $T$

a typed-indexed family of relations

- 2 Prove the “Basic Lemma”

$$\Gamma \vdash t : T \implies P_T(t)$$

- 3 Recover desired theorem as corollary

1. Leave the technical details behind
2. Central slide. This we want to later implement in a LF
3. Basic Lemma: aka abstraction theorem, parametricity, fundamental lemma
4. Note the difference between “proof by logical relation” and the logical relation itself. Do not confuse them.

## Next Steps

### Logical Relations

class of proof methods applicable on many type theories

### Logical Framework

a uniform presentation language to formalize logics/type theories

### Logical Relations for a Logical Framework

a uniform notion of “proof by logical relations” in the setting of LF

# Logical Framework: Motivation

- There are many logics

FOL, SFOL, HOL, Modal Logic, Dynamic Logics, Discourse Representation Theory, Temporal Logic, Relevant Logic, Set Theories, Extensional Type Theories, Intensional Type Theories, Dependent Type Theories, ...

- Many logics have

- abstract syntax
- binding and substitution
- proof calculi with (schematic) rules & side conditions

“[I]t is important to define a **[uniform] presentation language** for defining logical systems that is a suitable basis for a logic-independent proof development environment.” ([HHP93])

⇒ “Edinburgh Logical Framework”, aka LF

1. Elaborate two points
2. I) Implementing development environments for all logics a daunting task
3. Implementing binding a major hassle
  - easy on pen & paper
  - e.g. in Coq non-trivial even for STLC
  - How to formalize lambda binder? String references? Doesn't scale, fresh variable names, beta reduction
  - Alternative: de Bruijn indices, but even those daunting to get right when done manually
4. II) This entails means of proof construction and checker – for every logic
  - LFs use “judgements-as-types” and “propositions-as-types” idioms
  - Encode proofs as terms, and proof checking as type checking
  - allows reusing dev env for syntax!
  - might seem standard procedure – esp to kwarcies – but is big selling point of LF

# Logical Framework: Motivation

- There are many logics

FOL, SFOL, HOL, Modal Logic, Dynamic Logics, Discourse Representation Theory, Temporal Logic, Relevant Logic, Set Theories, Extensional Type Theories, Intensional Type Theories, Dependent Type Theories, ...

- Many logics have

- abstract syntax
- binding and substitution
- proof calculi with (schematic) rules & side conditions
- a notion of logical relations

“[I]t is important to define a **[uniform] presentation language** for defining logical systems that is a suitable basis for a logic-independent proof development environment.” ([HHP93])

⇒ “Edinburgh Logical Framework”, aka LF

1. Elaborate two points
2. I) Implementing development environments for all logics a daunting task
3. Implementing binding a major hassle
  - easy on pen & paper
  - e.g. in Coq non-trivial even for STLC
  - How to formalize lambda binder? String references? Doesn't scale, fresh variable names, beta reduction
  - Alternative: de Bruijn indices, but even those daunting to get right when done manually
4. II) This entails means of proof construction and checker – for every logic
  - LFs use “judgements-as-types” and “propositions-as-types” idioms
  - Encode proofs as terms, and proof checking as type checking
  - allows reusing dev env for syntax!
  - might seem standard procedure – esp to kwarcies – but is big selling point of LF
5. III) Some logics have a notion of logrels, in fact many type theories admit one
6. Uniform notion of logrel in LF = natural extension of LF idea



# MMT/LF: A Module System over LF

## Definition (MMT/LF Grammar)

Formalize knowledge (set/type theories, logics, ...) into *theories of declarations*:

$Thy$	$::=$	<b>theory</b> $T = \{Decl^*\}$	theory definition
$Decl$	$::=$	<b>include</b> $T \mid c : A [= A]$	declarations in a theory
$A$	$::=$	<b>type</b> $\mid c \mid x \mid A A \mid$ $\lambda x:A. A \mid \Pi x:A. A \mid A \rightarrow A$	terms

## Example (Propositional Logic)

$A ::= \langle \text{unspecified} \rangle$	atoms	$\text{theory PL} = \left\{ \begin{array}{l} \text{prop: type} \\ \neg: \text{prop} \rightarrow \text{prop} \\ \wedge: \text{prop} \rightarrow \text{prop} \rightarrow \text{prop} \\ \supset: \text{prop} \rightarrow \text{prop} \rightarrow \text{prop} \\ \quad = \lambda p. \lambda q. \neg(p \wedge \neg q) \end{array} \right\}$
$P ::= A \mid \neg P \mid P \wedge P \mid$	propositions	
$P \supset P$		

1. omit typings in lambdas/Pis if reasonably clear from context
2. function type arguably special case of the former
3. left: syntax as BNF grammar, right: MMT/LF
4. optional definiens e.g. for implication
5. **Atoms implicit:** Encode atom  $a$  as  $a : \text{prop}$ .

# MMT/LF: A Module System over LF

## Definition (MMT/LF Grammar)

Formalize knowledge (set/type theories, logics, ...) into *theories of declarations*:

$Thy$	$::=$	<b>theory</b> $T = \{Decl^*\}$	theory definition
$Decl$	$::=$	<b>include</b> $T \mid c : A [= A]$	declarations in a theory
$A$	$::=$	<b>type</b> $\mid c \mid x \mid A A \mid$ $\lambda x:A. A \mid \Pi x:A. A \mid A \rightarrow A$	terms

1. omit typings in lambdas/Pis if reasonably clear from context
2. function type arguably special case of the former
3. left: syntax as BNF grammar, right: MMT/LF
4. optional definiens e.g. for implication
5. **Atoms implicit:** Encode atom  $a$  as  $a : \text{prop}$ .

## Example (Propositional Logic)

$A ::= \langle \text{unspecified} \rangle$	atoms	$\text{theory PL} = \left\{ \begin{array}{l} \text{prop: type} \\ \neg: \text{prop} \rightarrow \text{prop} \\ \wedge: \text{prop} \rightarrow \text{prop} \rightarrow \text{prop} \\ \supset: \text{prop} \rightarrow \text{prop} \rightarrow \text{prop} \\ \quad = \lambda p. \lambda q. \neg(p \wedge \neg q) \\ a: \text{prop} \end{array} \right\}$
$P ::= A \mid \neg P \mid P \wedge P \mid$	propositions	
$P \supset P$		

## Prop. Logic in MMT/LF

Syntax (as before):

$$\mathbf{theory\ PL} = \left\{ \begin{array}{l} \mathbf{prop: type} \\ \neg: \mathbf{prop} \rightarrow \mathbf{prop} \\ \wedge: \mathbf{prop} \rightarrow \mathbf{prop} \rightarrow \mathbf{prop} \end{array} \right\}$$

Proof Calculus:

- use **judgement-as-types** paradigm:  $\mathbf{prop. } p \text{ provable} \iff \Vdash p \text{ inhabited}$

$$\mathbf{theory\ PLND} = \left\{ \begin{array}{l} \mathbf{include\ PL} \\ \Vdash: \mathbf{prop} \rightarrow \mathbf{type} \end{array} \right\}$$

Syntax (as before):

$$\mathbf{theory\ PL} = \left\{ \begin{array}{l} \mathbf{prop: type} \\ \neg: \mathbf{prop} \rightarrow \mathbf{prop} \\ \wedge: \mathbf{prop} \rightarrow \mathbf{prop} \rightarrow \mathbf{prop} \end{array} \right\}$$

Proof Calculus:

- use **judgement-as-types** paradigm:  $\mathbf{prop. } p \text{ provable} \iff \Vdash p \text{ inhabited}$
- encode schematic rules by dependent function types

$$\mathbf{theory\ PLND} = \left\{ \begin{array}{l} \mathbf{include\ PL} \\ \Vdash: \mathbf{prop} \rightarrow \mathbf{type} \\ \Rightarrow_I: \prod p: \mathbf{prop. } \Vdash p \rightarrow \Vdash \neg \neg p \\ \wedge_I: \prod p q: \mathbf{prop. } \Vdash p \rightarrow \Vdash q \rightarrow \Vdash p \wedge q \\ \wedge_{EL}: \prod p q: \mathbf{prop. } \Vdash p \wedge q \rightarrow \Vdash p \\ \dots \end{array} \right\}$$

## Next Steps

### Logical Relations

class of proof methods applicable on many type theories

### Logical Framework

a uniform presentation language to formalize logics/type theories

### Logical Relations for a Logical Framework

a uniform notion of “proof by logical relations” in the setting of LF

## Recall: Proof by Logical Relation

- 1 Define a logical relation  $P_{-}(-)$ :  
on every type  $T$ , a relation  $P_T(-)$
- 2 Prove the “Basic Lemma”:  $\Gamma \vdash t : T \implies P_T(t)$
- 3 Recover desired theorem as corollary

1. see  $\text{PL}_r$  as an interface
2. This is one of the central slides of this talk. If there are any questions, please ask them now.

## Recall: Proof by Logical Relation

- 1 Define a logical relation  $P_{-}(-)$ :  
on every type  $T$ , a relation  $P_T(-)$
- 2 Prove the “Basic Lemma”:  $\Gamma \vdash t : T \implies P_T(t)$
- 3 Recover desired theorem as corollary

$$\text{theory PL} = \left\{ \begin{array}{l} \text{prop: type} \\ \neg: \text{prop} \rightarrow \text{prop} \\ \wedge: \text{prop} \rightarrow \text{prop} \rightarrow \text{prop} \end{array} \right\}$$

1. see  $\text{PL}_r$  as an interface
2. This is one of the central slides of this talk. If there are any questions, please ask them now.

## Recall: Proof by Logical Relation

- 1 Define a logical relation  $P_{-}(-)$ :  
on every type  $T$ , a relation  $P_T(-)$
- 2 Prove the “Basic Lemma”:  $\Gamma \vdash t : T \implies P_T(t)$
- 3 Recover desired theorem as corollary

$$\text{theory PL} = \left\{ \begin{array}{l} \text{prop: type} \\ \neg: \text{prop} \rightarrow \text{prop} \\ \wedge: \text{prop} \rightarrow \text{prop} \rightarrow \text{prop} \end{array} \right\}$$

In MMT/LF, Logical Relations over PL = realizations of

i.e. views with dom.  $\text{PL}_r$

$$\text{theory PL}_r = \left\{ \begin{array}{l} \text{include PL} \end{array} \right\}$$

1. see  $\text{PL}_r$  as an interface
2. This is one of the central slides of this talk. If there are any questions, please ask them now.



## Recall: Proof by Logical Relation

- 1 Define a logical relation  $P_{-}(-)$ :  
on every type  $T$ , a relation  $P_T(-)$
- 2 Prove the “Basic Lemma”:  $\Gamma \vdash t : T \implies P_T(t)$
- 3 Recover desired theorem as corollary

$$\text{theory PL} = \left\{ \begin{array}{l} \text{prop: type} \\ \neg: \text{prop} \rightarrow \text{prop} \\ \wedge: \text{prop} \rightarrow \text{prop} \rightarrow \text{prop} \end{array} \right\}$$

In MMT/LF, Logical Relations over PL = realizations of

i.e. views with dom.  $\text{PL}_r$

$$\text{theory PL}_r = \left\{ \begin{array}{l} \text{include PL} \\ \text{prop}_r : \text{prop} \rightarrow \text{type} \end{array} \right\}$$

1. see  $\text{PL}_r$  as an interface
2. This is one of the central slides of this talk. If there are any questions, please ask them now.

## Recall: Proof by Logical Relation

- 1 Define a logical relation  $P_{-}(-)$ :  
on every type  $T$ , a relation  $P_T(-)$
- 2 Prove the “Basic Lemma”:  $\Gamma \vdash t : T \implies P_T(t)$
- 3 Recover desired theorem as corollary

$$\text{theory PL} = \left\{ \begin{array}{l} \text{prop: type} \\ \neg: \text{prop} \rightarrow \text{prop} \\ \wedge: \text{prop} \rightarrow \text{prop} \rightarrow \text{prop} \end{array} \right\}$$

In MMT/LF, Logical Relations over PL = realizations of

i.e. views with dom.  $\text{PL}_r$

$$\text{theory PL}_r = \left\{ \begin{array}{l} \text{include PL} \\ \text{prop}_r: \text{prop} \rightarrow \text{type} \\ \neg_r: \Pi p: \text{prop}. \Pi p^*: \text{prop}_r p. \text{prop}_r (\neg p) \end{array} \right\}$$

1. see  $\text{PL}_r$  as an interface
2. This is one of the central slides of this talk. If there are any questions, please ask them now.

## Recall: Proof by Logical Relation

- 1 Define a logical relation  $P_{-}(-)$ :  
on every type  $T$ , a relation  $P_T(-)$
- 2 Prove the “Basic Lemma”:  $\Gamma \vdash t : T \implies P_T(t)$
- 3 Recover desired theorem as corollary

$$\text{theory PL} = \left\{ \begin{array}{l} \text{prop: type} \\ \neg: \text{prop} \rightarrow \text{prop} \\ \wedge: \text{prop} \rightarrow \text{prop} \rightarrow \text{prop} \end{array} \right\}$$

In MMT/LF, Logical Relations over PL = realizations of

i.e. views with dom.  $\text{PL}_r$

$$\text{theory PL}_r = \left\{ \begin{array}{l} \text{include PL} \\ \text{prop}_r : \text{prop} \rightarrow \text{type} \\ \neg_r : \Pi p : \text{prop}. \Pi p^* : \text{prop}_r p. \text{prop}_r (\neg p) \\ \wedge_r : \Pi p : \text{prop}. \Pi p^* : \text{prop}_r p. \Pi q : \text{prop}. \Pi q^* : \text{prop}_r q. \text{prop}_r (p \wedge q) \end{array} \right\}$$

1. see  $\text{PL}_r$  as an interface
2. This is one of the central slides of this talk. If there are any questions, please ask them now.

## Recall: Proof by Logical Relation

- 1 Define a logical relation  $P_{-}(-)$ :  
on every type  $T$ , a relation  $P_T(-)$
- 2 Prove the “Basic Lemma”:  $\Gamma \vdash t : T \implies P_T(t)$
- 3 Recover desired theorem as corollary

## In MMT/LF: Proofs by Logical Relation over PL

- 1 Define realization  $R$  of

i.e. a view  $\text{PL}_r \rightarrow R$

$$\text{theory } \text{PL}_r = \left\{ \begin{array}{l} \text{include PL} \\ \text{prop}_r : \text{prop} \rightarrow \text{type} \\ \neg_r : \prod p : \text{prop}. \prod p^* : \text{prop}_r p. \text{prop}_r (\neg p) \\ \wedge_r : \prod p : \text{prop}. \prod p^* : \text{prop}_r p. \prod q : \text{prop}. \prod q^* : \text{prop}_r q. \text{prop}_r (p \wedge q) \end{array} \right\}$$

- 2 **Basic Lemma**

- 3 **Open Question**

1. In the Basic Lemma, note that  $\vdash_{\text{PL}} \cdot \cdot$  is typing of LF
2. We have defined how proofs by logrels over PL work in MMT/LF now.
3. For all theories, we can define such a logrel interface theory and prove the Basic Lemma.
4. Understanding how the interface theory can be built in general isn't too complicated if you work through some examples.
5. But defining that formally is very complicated and is what Florian and Kristina did in their paper.

## Recall: Proof by Logical Relation

- 1 Define a logical relation  $P_{-}(-)$ :  
on every type  $T$ , a relation  $P_T(-)$
- 2 Prove the “Basic Lemma”:  $\Gamma \vdash t : T \implies P_T(t)$
- 3 Recover desired theorem as corollary

## In MMT/LF: Proofs by Logical Relation over PL

- 1 Define realization  $R$  of i.e. a view  $\text{PL}_r \rightarrow R$

$$\text{theory } \text{PL}_r = \left\{ \begin{array}{l} \text{include PL} \\ \text{prop}_r : \text{prop} \rightarrow \text{type} \\ \neg_r : \prod p : \text{prop}. \prod p^* : \text{prop}_r p. \text{prop}_r (\neg p) \\ \wedge_r : \prod p : \text{prop}. \prod p^* : \text{prop}_r p. \prod q : \text{prop}. \prod q^* : \text{prop}_r q. \text{prop}_r (p \wedge q) \end{array} \right\}$$

- 2 **Basic Lemma:**  $\vdash_{\text{PL}} p : \text{prop} \implies \vdash_R R(p) : \text{prop}_r p$  i.e.  $\text{prop}_r p$  inhabited

- 3 Open Question

1. In the Basic Lemma, note that  $\vdash_{\text{PL}} \cdot : \cdot$  is typing of LF
2. We have defined how proofs by logrels over PL work in MMT/LF now.
3. For all theories, we can define such a logrel interface theory and prove the Basic Lemma.
4. Understanding how the interface theory can be built in general isn't too complicated if you work through some examples.
5. But defining that formally is very complicated and is what Florian and Kristina did in their paper.

# The Basic Lemma in Detail

Given a realization  $R$  of  $\text{PL}_r$ , we have:

## Theorem (Basic Lemma for PL)

$$\vdash_{\text{PL}} p : \text{prop} \implies \vdash_R R(p) : \text{prop}_r p$$

i.e.  $\text{prop}_r p$  inhabited

Assume  $\vdash_{\text{PL}} p : \text{prop}$  and  $\vdash_{\text{PL}} q : \text{prop}$ .

Then all these terms are in the relation  $\text{prop}_r$ :

- $p$
- $p \wedge q$
- $p \wedge (\neg(q \wedge \neg\neg p))$
- $(q \wedge (p \wedge p \wedge \neg((q \wedge \neg q) \wedge p))) \wedge p \wedge (\neg(q \wedge \neg\neg p))$

## The Basic Lemma in Detail

Given a realization  $R$  of  $\text{PL}_r$ , we have:

### Theorem (Basic Lemma for PL)

$$\vdash_{\text{PL}} p : \text{prop} \implies \vdash_R R(p) : \text{prop}_r p$$

i.e.  $\text{prop}_r p$  inhabited

Assume  $\vdash_{\text{PL}} p : \text{prop}$  and  $\vdash_{\text{PL}} q : \text{prop}$ .

Then all these terms are in the relation  $\text{prop}_r$ :

- $p$
- $p \wedge q$
- $p \wedge (\neg(q \wedge \neg\neg p))$
- $(q \wedge (p \wedge p \wedge \neg((q \wedge \neg q) \wedge p))) \wedge p \wedge (\neg(q \wedge \neg\neg p))$

MMT/LF logical relations prove statements all-quantified over terms of an MMT/LF theory.

## Examples for MMT/LF Logical Relations

Any “realistic” proof by logical relations would be far too complicated  
Instead, we stay in PL and prove

### Theorem (Tertium Non Datur)

If  $a \vee \neg a$  for all atoms,  
then  $p \vee \neg p$  for all propositions.

### Theorem (Double Negation Elimination)

If  $a \Leftrightarrow \neg\neg a$  for all atoms,  
then  $p \Leftrightarrow \neg\neg p$  for all propositions.

Do you see how both are instances of the Basic Lemma?



## Tertium Non Datur (the goal)

If  $a \vee \neg a$  for all atoms,  
then  $p \vee \neg p$  for all propositions.

## Basic Lemma for TND (what we get)

$\vdash_{\text{PL}} p : \text{prop} \implies \vdash_{\text{TND}} \text{TND}(p) : \text{prop}_r p$

1. Non-constructive on atoms  $\implies$  non-constructive on all propositions.
2. Recall: atoms modeled implicitly
3. If we had  $a : \text{prop}$ , then to realize  $\text{PL}_r$ , we would also need to give a definition to  $a_r$ .

theory TND = {

- include PL
- include PLND

---

- realize  $\text{PL}_r$
- $\text{prop}_r : \text{prop} \rightarrow \text{type}$
- $= ?$
- $\neg_r : \prod p : \text{prop}. \prod p^* : \text{prop}_r p. \text{prop}_r (\neg p)$
- $= ?$
- $\wedge_r : \prod p : \text{prop}. \prod p^* : \text{prop}_r p. \prod q : \text{prop}. \prod q^* : \text{prop}_r q. \text{prop}_r (p \wedge q)$
- $= ?$

}

## Tertium Non Datur (the goal)

If  $a \vee \neg a$  for all atoms,  
then  $p \vee \neg p$  for all propositions.

## Basic Lemma for TND (what we get)

$$\vdash_{\text{PL}} p : \text{prop} \implies \vdash_{\text{TND}} \text{TND}(p) : \text{prop}_r p$$

1. Non-constructive on atoms  $\implies$  non-constructive on all propositions.
2. Recall: atoms modeled implicitly
3. If we had  $a : \text{prop}$ , then to realize  $\text{PL}_r$ , we would also need to give a definition to  $a_r$ .

theory TND = {

- include PL
- include PLND

---

- realize  $\text{PL}_r$
- $\text{prop}_r : \text{prop} \rightarrow \text{type}$
- $= \lambda p. \Vdash p \vee \neg p$
- $\neg_r : \Pi p : \text{prop}. \Pi p^* : \text{prop}_r p. \text{prop}_r (\neg p)$
- $= ?$
- $\wedge_r : \Pi p : \text{prop}. \Pi p^* : \text{prop}_r p. \Pi q : \text{prop}. \Pi q^* : \text{prop}_r q. \text{prop}_r (p \wedge q)$
- $= ?$

}

## Tertium Non Datur (the goal)

If  $a \vee \neg a$  for all atoms,  
then  $p \vee \neg p$  for all propositions.

## Basic Lemma for TND (what we get)

$\vdash_{\text{PL}} p : \text{prop} \implies \vdash_{\text{TND}} \text{TND}(p) : \Vdash p \vee \neg p$

1. Non-constructive on atoms  $\implies$  non-constructive on all propositions.
2. Recall: atoms modeled implicitly
3. If we had  $a : \text{prop}$ , then to realize  $\text{PL}_{\text{r}}$ , we would also need to give a definition to  $a_{\text{r}}$ .

theory TND = {

- include PL
- include PLND

---

- realize  $\text{PL}_{\text{r}}$
- $\text{prop}_{\text{r}} : \text{prop} \rightarrow \text{type}$
- $= \lambda p. \Vdash p \vee \neg p$
- $\neg_{\text{r}} : \Pi p : \text{prop}. \Pi p^* : \text{prop}_{\text{r}} p. \text{prop}_{\text{r}} (\neg p)$
- $= ?$
- $\wedge_{\text{r}} : \Pi p : \text{prop}. \Pi p^* : \text{prop}_{\text{r}} p. \Pi q : \text{prop}. \Pi q^* : \text{prop}_{\text{r}} q. \text{prop}_{\text{r}} (p \wedge q)$
- $= ?$

}

# Tertium Non Datur (the goal)

If  $a \vee \neg a$  for all atoms,  
then  $p \vee \neg p$  for all propositions.

theory TND = {

- include PL
- include PLND
- realize PL<sub>r</sub>
- prop<sub>r</sub> : prop → type
- =  $\lambda p. \Vdash p \vee \neg p$
- $\neg_r : \Pi p : \text{prop}. \Pi p^* : \text{prop}_r p. \text{prop}_r (\neg p)$
- = ?
- $\wedge_r : \Pi p : \text{prop}. \Pi p^* : \text{prop}_r p. \Pi q : \text{prop}. \Pi q^* : \text{prop}_r q. \text{prop}_r (p \wedge q)$
- = ?

1	$p \vee \neg p$	
2	$p$	
3	$\neg \neg p$	$\neg_I, 2$
4	$\neg p \vee \neg \neg p$	$\vee_{IR}, 3$
5	$\neg p$	
6	$\neg p \vee \neg \neg p$	$\vee_{IL}, 5$
7	$\neg p \vee \neg \neg p$	$\vee_E, 1, 2-4, 5-6$

1. Non-constructive on atoms  $\implies$  non-constructive on all propositions.
2. Recall: atoms modeled implicitly
3. If we had  $a : \text{prop}$ , then to realize PL<sub>r</sub>, we would also need to give a definition to  $a_r$ .

# Tertium Non Datur (the goal)

If  $a \vee \neg a$  for all atoms,  
then  $p \vee \neg p$  for all propositions.

theory TND = {

- include PL
- include PLND
- realize PL<sub>r</sub>
- prop<sub>r</sub> : prop → type
- =  $\lambda p. \Vdash p \vee \neg p$
- $\neg_r : \Pi p : \text{prop}. \Pi p^* : \text{prop}_r p. \text{prop}_r (\neg p)$
- =  $\lambda p. \lambda p^*. \vee_E p^* (\lambda p. \vee_{IR} (\neg_I p)) (\lambda np. \vee_{IL} np)$
- $\wedge_r : \Pi p : \text{prop}. \Pi p^* : \text{prop}_r p. \Pi q : \text{prop}. \Pi q^* : \text{prop}_r q. \text{prop}_r (p \wedge q)$
- = ?

1	$p \vee \neg p$	
2	$p$	
3	$\neg \neg p$	$\neg_I, 2$
4	$\neg p \vee \neg \neg p$	$\vee_{IR}, 3$
5	$\neg p$	
6	$\neg p \vee \neg \neg p$	$\vee_{IL}, 5$
7	$\neg p \vee \neg \neg p$	$\vee_E, 1, 2-4, 5-6$

1. Non-constructive on atoms  $\implies$  non-constructive on all propositions.
2. Recall: atoms modeled implicitly
3. If we had  $a : \text{prop}$ , then to realize PL<sub>r</sub>, we would also need to give a definition to  $a_r$ .

## Tertium Non Datur (the goal)

If  $a \vee \neg a$  for all atoms,  
then  $p \vee \neg p$  for all propositions.

## Basic Lemma for TND (what we get)

$$\vdash_{\text{PL}} p : \text{prop} \implies \vdash_{\text{TND}} \text{TND}(p) : \Vdash p \vee \neg p$$

1. Non-constructive on atoms  $\implies$  non-constructive on all propositions.
2. Recall: atoms modeled implicitly
3. If we had  $a : \text{prop}$ , then to realize  $\text{PL}_{\text{r}}$ , we would also need to give a definition to  $a_{\text{r}}$ .

theory TND = {

- include PL
- include PLND

---

- realize  $\text{PL}_{\text{r}}$
- $\text{prop}_{\text{r}} : \text{prop} \rightarrow \text{type}$
- $= \lambda p. \Vdash p \vee \neg p$
- $\neg_{\text{r}} : \Pi p : \text{prop}. \Pi p^* : \text{prop}_{\text{r}} p. \text{prop}_{\text{r}} (\neg p)$
- $= \lambda p. \lambda p^*. \vee_{\text{E}} p^* (\lambda \mathfrak{p}. \vee_{\text{IR}} (\neg_{\text{I}} \mathfrak{p})) (\lambda \mathfrak{np}. \vee_{\text{IL}} \mathfrak{np})$
- $\wedge_{\text{r}} : \Pi p : \text{prop}. \Pi p^* : \text{prop}_{\text{r}} p. \Pi q : \text{prop}. \Pi q^* : \text{prop}_{\text{r}} q. \text{prop}_{\text{r}} (p \wedge q)$
- $= \dots$

}

## Tertium Non Datur (the goal)

If  $a \vee \neg a$  for all atoms,  
then  $p \vee \neg p$  for all propositions.

## Basic Lemma for TND (what we get)

$$\vdash_{\text{PL}} p : \text{prop} \implies \vdash_{\text{TND}} \text{TND}(p) : \Vdash p \vee \neg p$$

1. Non-constructive on atoms  $\implies$  non-constructive on all propositions.
2. Recall: atoms modeled implicitly
3. If we had  $a : \text{prop}$ , then to realize  $\text{PL}_{\text{r}}$ , we would also need to give a definition to  $a_{\text{r}}$ .

theory TND = {

- include PL
- include PLND

---

- realize  $\text{PL}_{\text{r}}$
- $\text{prop}_{\text{r}} : \text{prop} \rightarrow \text{type}$
- $= \lambda p. \Vdash p \vee \neg p$
- $\neg_{\text{r}} : \Pi p : \text{prop}. \Pi p^* : \text{prop}_{\text{r}} p. \text{prop}_{\text{r}} (\neg p)$
- $= \lambda p. \lambda p^*. \vee_{\text{E}} p^* (\lambda \mathbf{p}. \vee_{\text{IR}} (\neg_{\text{I}} \mathbf{p})) (\lambda \mathbf{np}. \vee_{\text{IL}} \mathbf{np})$
- $\wedge_{\text{r}} : \Pi p : \text{prop}. \Pi p^* : \text{prop}_{\text{r}} p. \Pi q : \text{prop}. \Pi q^* : \text{prop}_{\text{r}} q. \text{prop}_{\text{r}} (p \wedge q)$
- $= \dots$
- $a_{\text{r}} : \text{prop}_{\text{r}} a = \langle \text{encode assumption} \rangle$

}

## Double Negation Elimination (the goal)

If  $a \Leftrightarrow \neg\neg a$  for all atoms,  
then  $p \Leftrightarrow \neg\neg p$  for all propositions.

## Basic Lemma for DNE (what we get)

$$\vdash_{\text{PL}} p : \text{prop} \implies \vdash_{\text{DNE}} \text{DNE}(p) : \text{prop}_r p$$

theory DNE = {

- include PL
- include PLND

---

- realize PL<sub>r</sub>
- prop<sub>r</sub> : prop → type
- = ?
- ¬<sub>r</sub> : Πp : prop. Πp\* : prop<sub>r</sub> p. prop<sub>r</sub> (¬p)
- = ?
- ∧<sub>r</sub> : Πp : prop. Πp\* : prop<sub>r</sub> p. Πq : prop. Πq\* : prop<sub>r</sub> q. prop<sub>r</sub> (p ∧ q)
- = ?

}



## Double Negation Elimination (the goal)

If  $a \Leftrightarrow \neg\neg a$  for all atoms,  
then  $p \Leftrightarrow \neg\neg p$  for all propositions.

## Basic Lemma for DNE (what we get)

$$\vdash_{\text{PL}} p : \text{prop} \implies \vdash_{\text{DNE}} \text{DNE}(p) : \text{prop}_r p$$

theory DNE = {

- include PL
- include PLND
- realize PL<sub>r</sub>
- prop<sub>r</sub> : prop → type
- =  $\lambda p. \Vdash p \Leftrightarrow \neg\neg p$
- $\neg_r : \Pi p : \text{prop}. \Pi p^* : \text{prop}_r p. \text{prop}_r (\neg p)$
- = ?
- $\wedge_r : \Pi p : \text{prop}. \Pi p^* : \text{prop}_r p. \Pi q : \text{prop}. \Pi q^* : \text{prop}_r q. \text{prop}_r (p \wedge q)$
- = ?

}

## Double Negation Elimination (the goal)

If  $a \Leftrightarrow \neg\neg a$  for all atoms,  
then  $p \Leftrightarrow \neg\neg p$  for all propositions.

## Basic Lemma for DNE (what we get)

$\vdash_{\text{PL}} p : \text{prop} \implies \vdash_{\text{DNE}} \text{DNE}(p) : \Vdash p \Leftrightarrow \neg\neg p$

theory DNE = {

- include PL
- include PLND

---

- realize PL<sub>r</sub>
- prop<sub>r</sub> : prop → type
- =  $\lambda p. \Vdash p \Leftrightarrow \neg\neg p$
- $\neg_r : \Pi p : \text{prop}. \Pi p^* : \text{prop}_r p. \text{prop}_r (\neg p)$
- = ?
- $\wedge_r : \Pi p : \text{prop}. \Pi p^* : \text{prop}_r p. \Pi q : \text{prop}. \Pi q^* : \text{prop}_r q. \text{prop}_r (p \wedge q)$
- = ?

}

## Double Negation Elimination (the goal)

If  $a \Leftrightarrow \neg\neg a$  for all atoms,  
then  $p \Leftrightarrow \neg\neg p$  for all propositions.

{	<b>include</b> PL <b>include</b> PLND <hr/> <b>realize</b> PL <sub>r</sub> prop <sub>r</sub> : prop → type $= \lambda p. \Vdash p \Leftrightarrow \neg\neg p$ $\neg_r : \prod p : \text{prop}. \prod p^* : \text{prop}_r p. \text{prop}_r (\neg p)$ $= ?$ $\wedge_r : \prod p : \text{prop}. \prod p^* : \text{prop}_r p. \prod q : \text{prop}. \prod q^* : \text{prop}_r q. \text{prop}_r (p \wedge q)$ $= ?$	<table border="1" style="border-collapse: collapse; width: 100%;"> <tr> <td style="padding: 5px;">1</td> <td style="padding: 5px;"><math>p \Leftrightarrow \neg\neg p</math></td> <td style="padding: 5px;"></td> </tr> <tr> <td style="padding: 5px;">2</td> <td style="padding: 5px;"><math>\neg p</math></td> <td style="padding: 5px;"></td> </tr> <tr> <td style="padding: 5px;">3</td> <td style="padding: 5px;"><math>\neg\neg(\neg p)</math></td> <td style="padding: 5px;"><math>\Rightarrow_I, 2</math></td> </tr> <tr> <td style="padding: 5px;">4</td> <td style="padding: 5px;"><math>\neg\neg(\neg p)</math></td> <td style="padding: 5px;"></td> </tr> <tr> <td style="padding: 5px;">5</td> <td style="padding: 5px;"><math>\neg(\neg\neg p)</math></td> <td style="padding: 5px;">reit., 4</td> </tr> <tr> <td style="padding: 5px;">6</td> <td style="padding: 5px;"><math>\neg p</math></td> <td style="padding: 5px;"><math>\Leftrightarrow_{EL}, 1, 5</math></td> </tr> <tr> <td style="padding: 5px;">7</td> <td style="padding: 5px;"><math>(\neg p) \Leftrightarrow \neg\neg(\neg p)</math></td> <td style="padding: 5px;"><math>\Leftrightarrow_I, 2-3, 4-6</math></td> </tr> </table>	1	$p \Leftrightarrow \neg\neg p$		2	$\neg p$		3	$\neg\neg(\neg p)$	$\Rightarrow_I, 2$	4	$\neg\neg(\neg p)$		5	$\neg(\neg\neg p)$	reit., 4	6	$\neg p$	$\Leftrightarrow_{EL}, 1, 5$	7	$(\neg p) \Leftrightarrow \neg\neg(\neg p)$	$\Leftrightarrow_I, 2-3, 4-6$
	1	$p \Leftrightarrow \neg\neg p$																					
2	$\neg p$																						
3	$\neg\neg(\neg p)$	$\Rightarrow_I, 2$																					
4	$\neg\neg(\neg p)$																						
5	$\neg(\neg\neg p)$	reit., 4																					
6	$\neg p$	$\Leftrightarrow_{EL}, 1, 5$																					
7	$(\neg p) \Leftrightarrow \neg\neg(\neg p)$	$\Leftrightarrow_I, 2-3, 4-6$																					

## Double Negation Elimination (the goal)

If  $a \Leftrightarrow \neg\neg a$  for all atoms,  
then  $p \Leftrightarrow \neg\neg p$  for all propositions.

theory DNE = {

- include PL
- include PLND
- realize PL<sub>r</sub>
- prop<sub>r</sub> : prop → type
- = λp. ⊢ p ⇔ ¬¬p
- ¬<sub>r</sub> : Πp: prop. Πp\* : prop<sub>r</sub> p. prop<sub>r</sub> (¬p)
- = λp. λp\*. ⇔<sub>I</sub> (λnp. ⇔<sub>I</sub> np) (λnnnp. ⇔<sub>EL</sub> p\*nnnp)
- ∧<sub>r</sub> : Πp: prop. Πp\* : prop<sub>r</sub> p. Πq: prop. Πq\* : prop<sub>r</sub> q. prop<sub>r</sub> (p ∧ q)
- = ?

}

1	$p \Leftrightarrow \neg\neg p$	
2	$\neg p$	
3	$\neg\neg(\neg p)$	$\Rightarrow_I, 2$
4	$\neg\neg(\neg p)$	
5	$\neg(\neg\neg p)$	reit., 4
6	$\neg p$	$\Leftrightarrow_{EL}, 1, 5$
7	$(\neg p) \Leftrightarrow \neg\neg(\neg p)$	$\Leftrightarrow_I, 2-3, 4-6$

## Double Negation Elimination (the goal)

If  $a \Leftrightarrow \neg\neg a$  for all atoms,  
then  $p \Leftrightarrow \neg\neg p$  for all propositions.

## Basic Lemma for DNE (what we get)

$\vdash_{\text{PL}} p : \text{prop} \implies \vdash_{\text{DNE}} \text{DNE}(p) : \Vdash p \Leftrightarrow \neg\neg p$

**theory DNE =** {

- include** PL
- include** PLND

---

- realize** PL<sub>r</sub>
- prop<sub>r</sub> : prop → type
- $= \lambda p. \Vdash p \Leftrightarrow \neg\neg p$
- $\neg_r : \Pi p : \text{prop}. \Pi p^* : \text{prop}_r p. \text{prop}_r (\neg p)$
- $= \lambda p. \lambda p^*. \Leftrightarrow_I (\lambda np. \Rightarrow_I np) (\lambda nnp. \Leftrightarrow_{\text{EL}} p^* nnp)$
- $\wedge_r : \Pi p : \text{prop}. \Pi p^* : \text{prop}_r p. \Pi q : \text{prop}. \Pi q^* : \text{prop}_r q. \text{prop}_r (p \wedge q)$
- $= \dots$

}

## Double Negation Elimination (the goal)

If  $a \Leftrightarrow \neg\neg a$  for all atoms,  
then  $p \Leftrightarrow \neg\neg p$  for all propositions.

## Basic Lemma for DNE (what we get)

$\vdash_{\text{PL}} p : \text{prop} \implies \vdash_{\text{DNE}} \text{DNE}(p) : \Vdash p \Leftrightarrow \neg\neg p$

theory DNE = {

- include PL
- include PLND

---

- realize PL<sub>r</sub>
- prop<sub>r</sub> : prop → type
- = λp.  $\Vdash p \Leftrightarrow \neg\neg p$
- ¬<sub>r</sub> : Πp : prop. Πp\* : prop<sub>r</sub> p. prop<sub>r</sub> (¬p)
- = λp. λp\*.  $\Leftrightarrow_{\text{I}} (\lambda np. \Rightarrow_{\text{I}} np) (\lambda nnnp. \Leftrightarrow_{\text{EL}} p^* nnnp)$
- ∧<sub>r</sub> : Πp : prop. Πp\* : prop<sub>r</sub> p. Πq : prop. Πq\* : prop<sub>r</sub> q. prop<sub>r</sub> (p ∧ q)
- = ...
- a<sub>r</sub> : prop<sub>r</sub> a = ⟨encode assumption⟩

}

## Conclusion

**Logical Relations:** a class of proof methods

- used to prove meta theorems about type theories strong normalization, type safety, ...
- called for when induction over terms fails and IHs based on their type are required
- pattern of a **proof by logical relation**:
  - 1 Define a logical relation  $P_{-}(-)$ : for every type  $T$ , a relation  $P_T(-)$  on terms of  $T$
  - 2 Prove the “Basic Lemma”:  $\Gamma \vdash t : T \implies P_T(t)$
  - 3 Recover desired theorem as corollary

**Logical Relations in MMT/LF:**

only given for PL here

- 1 Define realization  $R$  of

$$\text{theory PL}_r = \left\{ \begin{array}{l} \text{include PL} \\ \text{prop}_r : \text{prop} \rightarrow \text{type} \\ \neg_r : \Pi p : \text{prop}. \Pi p^* : \text{prop}_r p. \text{prop}_r (\neg p) \end{array} \right\}$$

- 2 **Basic Lemma:**  $\vdash_{\text{PL}} p : \text{prop} \implies \vdash_R R(p) : \text{prop}_r p$

1. In general, for theory  $T$  can generate interface theory  $T_r$
2. Follows the same ideas we discussed, but a bit cumbersome to define formally; not suited for talk

## Appetizer for more Pursuit

- Logical relations that relate “modulo view application”

So far: given an MMT/LF theory  $T$ ,

MMT/LF logical relations over  $T$  prove statements about  $t$   
all-quantified over all  $T$ -terms  $t$

Now: given MMT/LF view  $v: S \rightarrow T$ :

MMT/LF logical relations **over**  $v: S \rightarrow T$  prove statements about  $v(s)$   
all-quantified over all  $S$ -terms  $s$

e.g.  $v = \text{TypeEras}: \text{Church} \rightarrow \text{Curry}$ , then prove “TypeEras(Church term) typable”.

- How can we make the Basic Lemma accessible *within* the formalization?

open question!

1. Only scratched surface
2. There are many more points; here two of them for my M.Sc. thesis
3. E.g. given a typed pair in Church we can type-erase that pair to get an untyped pair in Curry. That’s what this TypeEras view does.
4. But we’d like to say more: the untyped pair can actually be typed against the original type. It’s just that the type is no longer part of the term.
5. This we can express using logical relations along views



- Logical Relations:

- Concise and to-the-point proof of SN of STLC: [Zil12]

Note the relation for function types there is missing the condition of strong normalization, which I corrected for in my slides; if in doubt, compare with [Sko19].

- Another proof of SN of STLC: [Ahm13] (videos), third-party transcript at [Sko19, ch. 2]

Here, the lambda calculus is equipped with if-then-else constructs, too, making the proof a bit more complicated (on a first read). After SN of STLC, Ahmed considers several extensions of STLC, among others, with recursive types and reference types, and uses logical relations to prove meta theorems about them.

- Unrelated, but interesting: connection of logical relations with automata simulations: [tcs.SE]

- Logical Frameworks:

- MMT System: [Rab17; RM18]
- Edinburgh LF: [HHP93]

- Logical Relations for a Logical Framework:

## Further Pointers II

- The article on which this talk is based: [RS13]
- Proof of SN in Twelf: [SS08]

Concerning several ideas, the Twelf system is an ancestor of the MMT system. Hence, it is plausible that any method given in Twelf to prove SN of STLC can be carried over to MMT.

- Details of the mentioned Church to Curry idea: [Rou20, ch. 7]

This concrete exposition of mine gives a logical relation-free account of the Church to Curry idea for the MMT system. A version with logical relations in mind has been implemented as of January 2021, but not yet reported on.

- [Ahm13] Amal Ahmed. “Logical Relations”. Oregon Programming Languages Summer School 2013. July 2013. URL: <https://www.cs.uoregon.edu/research/summerschool/summer13/curriculum.html>.
- [HHP93] Robert Harper, Furio Honsell, and Gordon Plotkin. “A framework for defining logics”. In: *Journal of the Association for Computing Machinery* 40.1 (1993), pp. 143–184.
- [Rab17] Florian Rabe. “How to Identify, Translate, and Combine Logics?” In: *Journal of Logic and Computation* 27.6 (2017), pp. 1753–1798.

## Further Pointers III

- [RM18] Florian Rabe and Dennis Müller. “Structuring Theories with Implicit Morphisms”. In: *24th International Workshop on Algebraic Development Techniques 2018*. 2018. URL: [https://kwarc.info/people/frabe/Research/RM\\_implicit\\_18.pdf](https://kwarc.info/people/frabe/Research/RM_implicit_18.pdf).
- [Rou20] Navid Roux. *Structure-Preserving Diagram Operators*. Master Project Report. July 17, 2020. URL: [https://gl.kwarc.info/supervision/projectarchive/-/blob/master/2020/Roux\\_Navid.pdf](https://gl.kwarc.info/supervision/projectarchive/-/blob/master/2020/Roux_Navid.pdf).
- [RS13] Florian Rabe and Kristina Sojakova. “Logical Relations for a Logical Framework”. In: *ACM Transactions on Computational Logic* (2013). URL: [https://kwarc.info/frabe/Research/RS\\_logrels\\_12.pdf](https://kwarc.info/frabe/Research/RS_logrels_12.pdf).
- [Sko19] Lau Skorstengaard. *An Introduction to Logical Relations*. 2019.
- [SS08] C. Schürmann and J. Sarnat. “Structural Logical Relations”. In: *2008 23rd Annual IEEE Symposium on Logic in Computer Science*. 2008, pp. 69–80. DOI: [10.1109/LICS.2008.44](https://doi.org/10.1109/LICS.2008.44). (Visited on 01/26/2020).
- [tcs.SE] Hongjin Liang (<https://cstheory.stackexchange.com/users/2703/hongjin-liang>). *What are the differences between logical relations and simulations?* Theoretical Computer Science Stack Exchange. URL: <https://cstheory.stackexchange.com/q/5427> (version: 2012-06-04). URL: <https://cstheory.stackexchange.com/q/5427>.
- [Zil12] Beta Ziliani. “Strong Normalization for Simply Typed Lambda Calculus”. based on lectures by Derek Dreyer. July 2012. URL: <https://web.archive.org/web/20170829154544/https://people.mpi-sws.org/~dg/teaching/pt2012/sn.pdf> (visited on 01/26/2021).