A Logical Study of Program Equivalence

Guilhem Jaber

Ecole des Mines de Nantes
LINA - Ascola

PhD Defense
Institut Henri Poincaré (Paris)
July 11th 2014

Why study the Equivalence of Programs ?

- Specification of programs
  - $\rightsquigarrow$ Equivalence between a program we can trust and an optimized one.

Why study the Equivalence of Programs ?

- Specification of programs
  - $\rightsquigarrow$ Equivalence between a program we can trust and an optimized one.
- Compiler optimizations.
  - $\rightsquigarrow$ Towards verified compilers.

Why study the Equivalence of Programs ?

- Specification of programs
  - $\rightsquigarrow$ Equivalence between a program we can trust and an optimized one.
- Compiler optimizations.
  - $\rightsquigarrow$ Towards verified compilers.
- Representation independence of Data
  - $\rightsquigarrow$ Parametricity, Free theorems.

Why study the Equivalence of Programs ?

- Specification of programs
  - $\rightsquigarrow$ Equivalence between a program we can trust and an optimized one.
- Compiler optimizations.
  - $\rightsquigarrow$ Towards verified compilers.
- Representation independence of Data
  - $\rightsquigarrow$ Parametricity, Free theorems.
- Crucial in denotational semantics
  - $\rightsquigarrow$ Full-abstraction result.

What kind of Equivalence ?

- Contextual Equivalence
  - $\rightsquigarrow$ Programs seen as black boxes.

What kind of Equivalence ?

- Contextual Equivalence
  - $\rightsquigarrow$ Programs seen as black boxes.
- Extensional behavior of programs
  - $\rightsquigarrow$ Observational equivalence.

What kind of Equivalence ?

- Contextual Equivalence
  - ⇝ Programs seen as black boxes.
- Extensional behavior of programs
  - ⇝ Observational equivalence.
- Depends on the language contexts are written in
  - ⇝ discriminating power of contexts,
  - ⇝ from purely functional languages to assembly code.

For what kind of Language: RefML

A typed call-by-value λ-calculus: $(\lambda x : \tau.M)v \to M\{v/x\}$

For what kind of Language: RefML

A typed call-by-value λ-calculus: $(\lambda x : \tau . M) v \to M \{v/x\}$

with Integers and Booleans: $\texttt{if b then 0 else n} + 1$

For what kind of Language: RefML

A typed call-by-value λ-calculus: $(\lambda x : \tau.M)v \rightarrow M\{v/x\}$

with Integers and Booleans: $\texttt{if b then 0 else n} + 1$

with higher-order references: $\mathrm{ref}\,2, \mathrm{ref}\,(\lambda x.M)$

stored in heap via locations: $(\mathrm{ref}\; v, h) \rightarrow (\ell, h \cdot [\ell \mapsto v])$

(ℓ fresh in h)

mutable: $x := !x + 1$

For what kind of Language: RefML

A typed call-by-value λ-calculus:	$(\lambda x : \tau.M)v \rightarrow M\{v/x\}$
with Integers and Booleans:	$\texttt{if b then 0 else n} + 1$
with higher-order references:	$\mathrm{ref}\, 2, \mathrm{ref}\,(\lambda x.M)$
stored in heap via locations:	$(\mathrm{ref}\, v, h) \rightarrow (\ell, h \cdot [\ell \mapsto v])$
	(ℓ fresh in h)
mutable:	$x :=!x + 1$
No pointer arithmetic:	$(\ell + 1)$ is ill-typed
But equality test:	$\ell_1 == \ell_2$ is well-typed

For what kind of Language: RefML

A typed call-by-value λ-calculus: $(\lambda x : \tau.M)v \to M\{v/x\}$

with Integers and Booleans: $\texttt{if b then 0 else n} + 1$

with higher-order references: $\mathrm{ref}\,2, \mathrm{ref}\,(\lambda x.M)$

stored in heap via locations: $(\mathrm{ref}\,v, h) \to (\ell, h \cdot [\ell \mapsto v])$
(ℓ fresh in h)

mutable: $x :=!x + 1$

No pointer arithmetic: $(\ell + 1)$ is ill-typed

But equality test: $\ell_1 == \ell_2$ is well-typed

Full recursion (via “Landin” knot).

For what kind of Language: RefML

A typed call-by-value λ-calculus: $(\lambda x : \tau . M) v \to M\{v/x\}$

with Integers and Booleans: $\texttt{if b then 0 else n} + 1$

with higher-order references: $\mathrm{ref}\, 2, \mathrm{ref}\, (\lambda x.M)$

stored in heap via locations: $(\mathrm{ref}\, v, h) \to (\ell, h \cdot [\ell \mapsto v])$

(ℓ fresh in h)

mutable: $x := !x + 1$

No pointer arithmetic: $(\ell + 1)$ is ill-typed

But equality test: $\ell_1 == \ell_2$ is well-typed

Full recursion (via "Landin" knot).

Contextual equivalence of M_1, M_2:

$$\forall C. \forall h. (C[M_1] \Downarrow, h) \iff (C[M_2] \Downarrow, h)$$

Synchronization of Callbacks (I/II)

λ`f`.`f`() is **not** equivalent to λ`f`.`f`(); `f`()

Synchronization of Callbacks (I/II)

$$\lambda \texttt{f}.\texttt{f}() \qquad \text{is } \textbf{not} \text{ equivalent to} \qquad \lambda \texttt{f}.\texttt{f}(); \texttt{f}()$$

- $\rightsquigarrow$ Contexts can check how many time f is called.
- $\rightsquigarrow$ Callbacks are fully observable!

$$C[\bullet] \stackrel{def}{=} \texttt{let x} = \mathrm{ref}\, 0 \texttt{ in } \bullet\, (\lambda\_.\texttt{x} := !\texttt{x} + 1); \texttt{if } !\texttt{x} > 1 \texttt{ then } \Omega \texttt{ else}()$$

can discriminate them.

Synchronization of Callbacks (II/II)

$\lambda \texttt{f}.(\texttt{f 1}) + (\texttt{f 2})$ is **not** equivalent to $\lambda \texttt{f}.(\texttt{f 2}) + (\texttt{f 1})$

Synchronization of Callbacks (II/II)

$$\lambda \texttt{f}.(\texttt{f 1}) + (\texttt{f 2}) \qquad \text{is } \textbf{not} \text{ equivalent to} \qquad \lambda \texttt{f}.(\texttt{f 2}) + (\texttt{f 1})$$

$\rightsquigarrow$ Arguments given to callbacks must be related.

$$C[\bullet] \stackrel{def}{=} \texttt{let x} = \mathrm{ref}\,0 \texttt{ in } \bullet (\lambda \texttt{y}.\texttt{x} := \texttt{y}); \texttt{if !x} == \texttt{1 then } \Omega \texttt{ else}()$$

can discriminate them.

Disclosure of Locations (I/II)

$$\lambda\_.\texttt{let x} = \texttt{ref0 in 1} \quad \text{is equivalent to} \quad \lambda\_.\texttt{1}$$

- $\leadsto$ The creation of the reference bounded to x is not observable by the context.
- $\leadsto$ It is private to the term!

Disclosure of Locations (II/II)

$$\lambda \texttt{f}.\texttt{let x} \; = \; \texttt{ref0 in fx}; \texttt{x} := 1$$

is **not** equivalent to

$$\lambda \texttt{f}.\texttt{let x} \; = \; \texttt{ref0 in fx}; \texttt{x} := 2$$

Disclosure of Locations (II/II)

$$\lambda \mathtt{f.let\ x} \;=\; \mathtt{ref0\ in\ fx; x := 1}$$

is **not** equivalent to

$$\lambda \mathtt{f.let\ x} \;=\; \mathtt{ref0\ in\ fx; x := 2}$$

- ⇝ The reference bounded to x is disclosed to the context.
- ⇝ It can look inside afterward to see what is stored.

$C[\bullet] \stackrel{def}{=} \mathtt{let\ z} = \mathrm{ref}\,(\mathrm{ref}\,\mathtt{0})\ \mathtt{in}\ \bullet(\lambda \mathtt{y.z} := \mathtt{y}); \mathtt{if}\ !!\mathtt{z} == \mathtt{1}\ \mathtt{then}\ \Omega\ \mathtt{else}()$
can discriminate them.

How to prove equivalence of programs of RefML ?

Contextual equivalence is hard to reason on

$\rightsquigarrow$ Quantification over any contexts and heaps.

How to prove equivalence of programs of RefML ?

Contextual equivalence is hard to reason on

$\rightsquigarrow$ Quantification over any contexts and heaps.

- Nominal Game Semantics (Murawski & Tzevelekos, LICS'11)
  - $\rightsquigarrow$ Fully-abstract for RefML
  - $\rightsquigarrow$ Trace representation (Laird, ICALP'07)
  - $\rightsquigarrow$ automata-based interpretation: Algorithmic Game Semantics.

How to prove equivalence of programs of RefML ?

Contextual equivalence is hard to reason on

$\rightsquigarrow$ Quantification over any contexts and heaps.

- Nominal Game Semantics (Murawski & Tzevelekos, LICS'11)
  - $\rightsquigarrow$ Fully-abstract for RefML
  - $\rightsquigarrow$ Trace representation (Laird, ICALP'07)
  - $\rightsquigarrow$ automata-based interpretation: Algorithmic Game Semantics.
- Kripke Logical Relations
  - $\rightsquigarrow$ World as heap-invariants (Pitts & Stark)
  - $\rightsquigarrow$ Evolution of invariants (Ahmed, Dreyer, Neis & Birkedal).

How to prove equivalence of programs of RefML ?

Contextual equivalence is hard to reason on

$\rightsquigarrow$ Quantification over any contexts and heaps.

- Nominal Game Semantics (Murawski & Tzevelekos, LICS'11)
  - $\rightsquigarrow$ Fully-abstract for RefML
  - $\rightsquigarrow$ Trace representation (Laird, ICALP'07)
  - $\rightsquigarrow$ automata-based interpretation: Algorithmic Game Semantics.
- Kripke Logical Relations
  - $\rightsquigarrow$ World as heap-invariants (Pitts & Stark)
  - $\rightsquigarrow$ Evolution of invariants (Ahmed, Dreyer, Neis & Birkedal).
- Bisimulations
  - $\rightsquigarrow$ Environmental Bisimulations (Pierce & Sumii, Koutavas, Wand)
  - $\rightsquigarrow$ Open Bisimulations (Lassen, Levy, Stovring).
  - $\rightsquigarrow$ Parametric Bisimulations (Hur, Dreyer & Vafeiadis).

The Ultimate Goal of this Thesis

- Formalize proofs of equivalence of programs:
  - ⇝ in a Proof Assistant based on *Dependent Type Theory* (Coq),
  - ⇝ abstracting over bureaucracy details (step-indexing, evolution of worlds,...).

The Ultimate Goal of this Thesis

- Formalize proofs of equivalence of programs:
  - $\rightsquigarrow$ in a Proof Assistant based on *Dependent Type Theory* (Coq),
  - $\rightsquigarrow$ abstracting over bureaucracy details (step-indexing, evolution of worlds,...).

- Model-check equivalence of programs:
  - $\rightsquigarrow$ Only need to give precise enough invariants on heaps and their evolution w.r.t. control flow (i.e. worlds),
  - $\rightsquigarrow$ Model-check a formula, representing the equivalence of programs, with such worlds.

The Ultimate Goal of this Thesis

- Formalize proofs of equivalence of programs:
  - ⇝ in a Proof Assistant based on *Dependent Type Theory* (Coq),
  - ⇝ abstracting over bureaucracy details (step-indexing, evolution of worlds,...).

- Model-check equivalence of programs:
  - ⇝ Only need to give precise enough invariants on heaps and their evolution w.r.t. control flow (i.e. worlds),
  - ⇝ Model-check a formula, representing the equivalence of programs, with such worlds.

- Decide equivalence of programs:
  - ⇝ undecidable in general, even without recursion and with bounded integers (Murawski & Tzevelekos)
  - ⇝ but for fragments of the language
  - ⇝ by generating such worlds,
  - ⇝ need completeness of our approach.

Formalize Proofs of Equivalence of Programs (in MLTT)

Want to abstract over bureaucracy details:

- Step-indexing (Appel & McAlester, Ahmed)
  - $\rightsquigarrow$ Necessary to break circularity in definitions,
  - $\rightsquigarrow$ But "pollutes" the proof with tedious details.

Formalize Proofs of Equivalence of Programs (in MLTT)

Want to abstract over bureaucracy details:

- Step-indexing (Appel & McAlester, Ahmed)
  - $\rightsquigarrow$ Necessary to break circularity in definitions,
  - $\rightsquigarrow$ But "pollutes" the proof with tedious details.

- Solution: use modality $\triangleright$ to abstract over it.
  - $\rightsquigarrow$ Using Gödel-Lob Logic (Appel, Mellies, Richards & Vouillon, Nakano).

Formalize Proofs of Equivalence of Programs (in MLTT)

Want to abstract over bureaucracy details:

- Step-indexing (Appel & McAlester, Ahmed)
  - $\rightsquigarrow$ Necessary to break circularity in definitions,
  - $\rightsquigarrow$ But "pollutes" the proof with tedious details.
- Solution: use modality $\triangleright$ to abstract over it.
  - $\rightsquigarrow$ Using Gödel-Lob Logic (Appel, Mellies, Richards & Vouillon, Nakano).
- Problem: extend this solution to Type Theory
  - $\rightsquigarrow$ Guarded Recursive Types in Topos of Tree (Birkedal et al.).

Formalize Proofs of Equivalence of Programs (in MLTT)

Want to abstract over bureaucracy details:

- Step-indexing (Appel & McAlester, Ahmed)
  - $\rightsquigarrow$ Necessary to break circularity in definitions,
  - $\rightsquigarrow$ But "pollutes" the proof with tedious details.
- Solution: use modality $\triangleright$ to abstract over it.
  - $\rightsquigarrow$ Using Gödel-Lob Logic (Appel, Mellies, Richards & Vouillon, Nakano).
- Problem: extend this solution to Type Theory
  - $\rightsquigarrow$ Guarded Recursive Types in Topos of Tree (Birkedal et al.).
- Our solution:

  Generic extension of Martin-Löf Type Theory via presheaf translation

Formalize Proofs of Equivalence of Programs (in MLTT)

Want to abstract over bureaucracy details:

- Step-indexing (Appel & McAlester, Ahmed)
  - ⇝ Necessary to break circularity in definitions,
  - ⇝ But "pollutes" the proof with tedious details.
- Solution: use modality $\triangleright$ to abstract over it.
  - ⇝ Using Gödel-Lob Logic (Appel, Mellies, Richards & Vouillon, Nakano).
- Problem: extend this solution to Type Theory
  - ⇝ Guarded Recursive Types in Topos of Tree (Birkedal et al.).
- Our solution:

Generic extension of Martin-Löf Type Theory via presheaf translation

- Could be useful to other problems:
  - ⇝ Reasoning on binding and substitution (HOAS, Nominal Logic),
  - ⇝ Kripke semantics over worlds.

Model-Check Equivalence of Programs

Soundness of Temporal Logical Relations

Let $\vdash M_1, M_2 : \tau$ two **non-recursive** terms:

- Generate **automatically** a **formula** $\boxed{\mathbb{E}[\![\tau]\!](M_1, M_2)}$ in a logic with:
  - $\rightsquigarrow$ (branching time) temporal modalities $\square, \mathbf{X}, \ldots$,
  - $\rightsquigarrow$ heap constraints, ex: $\ell \hookrightarrow v \wedge v = 3$.

Let $\vdash M_1, M_2 : \tau$ two **non-recursive** terms:

- Generate **automatically** a **formula** $\boxed{\mathbb{E}[\![\tau]\!](M_1, M_2)}$ in a logic with:
  - $\rightsquigarrow$ (branching time) temporal modalities $\Box, \mathbf{X}, \ldots$,
  - $\rightsquigarrow$ heap constraints, ex: $\ell \hookrightarrow v \wedge v = 3$.
- **Kripke Semantics**: $w \models_{\mathcal{A}} \varphi$
  - $\rightsquigarrow$ w: current invariant $\Rightarrow$ meaning to heap constraints
  - $\rightsquigarrow$ $\mathcal{A}$: fixed transition system $\Rightarrow$ meaning to temporal modalities.

Model-Check Equivalence of Programs

Soundness of Temporal Logical Relations

Let $\vdash M_1, M_2 : \tau$ two **non-recursive** terms:

- Generate **automatically** a **formula** $\boxed{\mathbb{E}[\![\tau]\!](M_1, M_2)}$ in a logic with:
  - ⇝ (branching time) temporal modalities $\Box, \mathbf{X}, \ldots,$
  - ⇝ heap constraints, ex: $\ell \hookrightarrow v \wedge v = 3$.
- **Kripke Semantics**: $w \models_{\mathcal{A}} \varphi$
  - ⇝ w: current invariant $\Rightarrow$ meaning to heap constraints
  - ⇝ $\mathcal{A}$: fixed transition system $\Rightarrow$ meaning to temporal modalities.
- **Soundness**: If there exists a transition system $\mathcal{A}$ s.t. $w_0 \models_{\mathcal{A}} \mathbb{E}[\![\tau]\!](M_1, M_2)$ then M_1, M_2 are contextually equivalent.

Model-Check Equivalence of Programs

Soundness of Temporal Logical Relations

Let $\vdash M_1, M_2 : \tau$ two **non-recursive** terms:

- Generate **automatically** a **formula** $\boxed{\mathbb{E}[\![\tau]\!](M_1, M_2)}$ in a logic with:
  - $\leadsto$ (branching time) temporal modalities $\Box, \mathbf{X}, \ldots,$
  - $\leadsto$ heap constraints, ex: $\ell \hookrightarrow v \wedge v = 3$.
- **Kripke Semantics**: $w \models_{\mathcal{A}} \varphi$
  - $\leadsto$ w: current invariant $\Rightarrow$ meaning to heap constraints
  - $\leadsto$ $\mathcal{A}$: fixed transition system $\Rightarrow$ meaning to temporal modalities.
- **Soundness**: If there exists a transition system $\mathcal{A}$ s.t. $w_0 \models_{\mathcal{A}} \mathbb{E}[\![\tau]\!](M_1, M_2)$ then M_1, M_2 are contextually equivalent.
- **Model-checking**: taking $\mathcal{A}$ and w, automatically check that $w \models_{\mathcal{A}} \mathbb{E}[\![\tau]\!](M_1, M_2)$
  - $\leadsto$ Using SMT-solvers $\Rightarrow$ only possible with bounded heaps in w.

- **Completeness**:

  If M_1, M_2 are contextually equivalent then there exists a transition system $\mathcal{A}$ s.t. $w_0 \models_{\mathcal{A}} \mathbb{E}[\![\tau]\!](M_1, M_2)$.

- **Completeness**:

  If M_1, M_2 are contextually equivalent then there exists a transition system $\mathcal{A}$ s.t. $w_0 \models_{\mathcal{A}} \mathbb{E}[\![\tau]\!](M_1, M_2)$.

- Generate **automatically** the **transition system** $\mathcal{A}$ s.t. $w_0 \models_{\mathcal{A}} \mathbb{E}[\![\tau]\!](M_1, M_2)$ iff $M_1 \simeq_{ctx} M_2$.

- **Completeness**:

  If M_1, M_2 are contextually equivalent then there exists a transition system $\mathcal{A}$ s.t. $w_0 \models_{\mathcal{A}} \mathbb{E}[\![\tau]\!](M_1, M_2)$.

- Generate **automatically** the **transition system** $\mathcal{A}$ s.t. $w_0 \models_{\mathcal{A}} \mathbb{E}[\![\tau]\!](M_1, M_2)$ iff $M_1 \simeq_{ctx} M_2$.

  - For purely functional terms:
    - $\rightsquigarrow$ No heap-invariants needed,
    - $\rightsquigarrow$ $\mathcal{A}$: trivial single-state transition system.

- **Completeness**:

  If M_1, M_2 are contextually equivalent then there exists a transition system $\mathcal{A}$ s.t. $w_0 \models_{\mathcal{A}} \mathbb{E}[\![\tau]\!](M_1, M_2)$.

- Generate **automatically** the **transition system** $\mathcal{A}$ s.t. $w_0 \models_{\mathcal{A}} \mathbb{E}[\![\tau]\!](M_1, M_2)$ iff $M_1 \simeq_{ctx} M_2$.

  - For purely functional terms:
    - $\rightsquigarrow$ No heap-invariants needed,
    - $\rightsquigarrow$ $\mathcal{A}$: trivial single-state transition system.
  - Possible generalization of results from Algorithmic game semantics ?
    - $\rightsquigarrow$ Bounded heaps hypothesis rather than type restriction.

Soundness and Completeness: A long road

Contextual Equivalence
$\Gamma \vdash M_1 \simeq_{ctx} M_2 : \tau$

Game Semantics
$[\![\Gamma \vdash M_1 : \tau]\!] = [\![\Gamma \vdash M_2 : \tau]\!]$

Trace Semantics
$[\Gamma \vdash M_1 : \tau] = [\Gamma \vdash M_2 : \tau]$

Concrete Logical Relations
$(M_1, M_2) \in \mathcal{V}\,[\![\tau]\!]_e\, w$

Temporal Logical Relations
$w \models_{\mathcal{A}} \mathbb{E}[\![\tau]\!](M_1, M_2)$

Soundness and Completeness: A long road

Contextual Equivalence
$\Gamma \vdash M_1 \simeq_{ctx} M_2 : \tau$

$\updownarrow$

Game Semantics
$[\![\Gamma \vdash M_1 : \tau]\!] = [\![\Gamma \vdash M_2 : \tau]\!]$

$\updownarrow$

Trace Semantics
$[\Gamma \vdash M_1 : \tau] = [\Gamma \vdash M_2 : \tau]$

$\updownarrow$

Concrete Logical Relations
$(M_1, M_2) \in \mathcal{V}[\![\tau]\!]_e\, w$

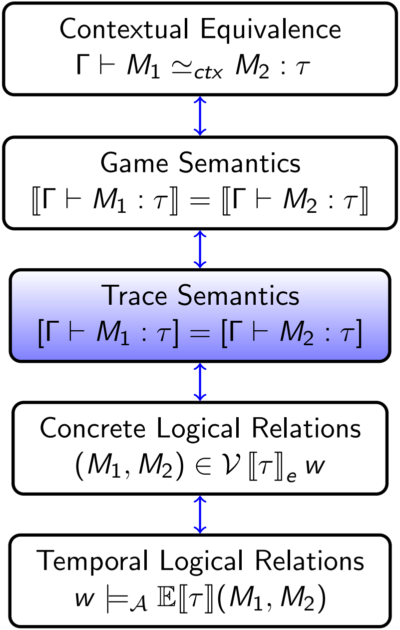
$\updownarrow$

Temporal Logical Relations
$w \models_{\mathcal{A}} \mathbb{E}[\![\tau]\!](M_1, M_2)$

Nominal Game Semantics

- $\rightsquigarrow$ Murawski & Tzevelekos (LICS'11)
- $\rightsquigarrow$ *Fully-abstract Intentional* model of RefML,
- $\rightsquigarrow$ No need of extensional quotient,
- $\rightsquigarrow$ Strategies as *Nominal Sets over Locations*.

Soundness and Completeness: A long road



Operational Nominal Game Semantics:

- ⇝ trace representation of interactions between a term and contexts,
- ⇝ generated by an *interactive reduction*,
- ⇝ a categorical structure on traces: *closed-Freyd category*,
- ⇝ a formal link with Nominal Game Semantics,
- ⇝ a treatment of visibility and ground references.

Soundness and Completeness: A long road

Contextual Equivalence
$\Gamma \vdash M_1 \simeq_{ctx} M_2 : \tau$

$\updownarrow$

Game Semantics
$[\![\Gamma \vdash M_1 : \tau]\!] = [\![\Gamma \vdash M_2 : \tau]\!]$

$\updownarrow$

Trace Semantics
$[\Gamma \vdash M_1 : \tau] = [\Gamma \vdash M_2 : \tau]$

$\updownarrow$

Concrete Logical Relations
$(M_1, M_2) \in \mathcal{V}\,[\![\tau]\!]_e\, w$

$\updownarrow$

Temporal Logical Relations
$w \models_{\mathcal{A}} \mathbb{E}[\![\tau]\!](M_1, M_2)$

Concrete Logical Relations

- $\rightsquigarrow$ avoid any quantification over complex elements in the definition,
- $\rightsquigarrow$ soundness and completeness via Operational Nominal Game Semantics.

Soundness and Completeness: A long road

Contextual Equivalence
$\Gamma \vdash M_1 \simeq_{ctx} M_2 : \tau$

↕

Game Semantics
$[\![\Gamma \vdash M_1 : \tau]\!] = [\![\Gamma \vdash M_2 : \tau]\!]$

↕

Trace Semantics
$[\Gamma \vdash M_1 : \tau] = [\Gamma \vdash M_2 : \tau]$

↕

Concrete Logical Relations
$(M_1, M_2) \in \mathcal{V} [\![\tau]\!]_e \, w$

↕

Temporal Logical Relations
$w \models_{\mathcal{A}} \mathbb{E}[\![\tau]\!](M_1, M_2)$

Temporal Logical Relations

- ⇝ *temporal modalities* to reason abstractly over worlds,
- ⇝ *symbolic execution* to reason abstractly over open ground variables.

Extending Type Theory with Forcing

First intuitions

- Guarded recursive types can be seen as Presheaves
  - ⇝ Work on Topos of Trees by Birkedal et al.
- Forcing Models
  - ⇝ Introduce by Cohen to build a model negating the Continuum Hypothesis
  - ⇝ Restatment of Lawvere and Tierney in terms of topos of (pre)sheafs.
- Computational meaning of Forcing
  - ⇝ Classical realizability by Krivine,
  - ⇝ Syntactic Forcing translations of proofs by Miquel,
  - ⇝ Computational interpretation of proofs of continuity, joint work with Coquand.

Extending Type Theory with Forcing

Joint work with Tabareau & Sozeau (LICS'11)

- Internalization of *Presheaf Models* in *Martin-Löf Type Theory*.

Extending Type Theory with Forcing

Joint work with Tabareau & Sozeau (LICS'11)

- Internalization of *Presheaf Models* in *Martin-Löf Type Theory*.

- Allow to extend syntactically MLTT with new principles.

Extending Type Theory with Forcing

Joint work with Tabareau & Sozeau (LICS'11)

- Internalization of *Presheaf Models* in *Martin-Löf Type Theory*.

- Allow to extend syntactically MLTT with new principles.

- Keep good properties of the theory:
  - $\rightsquigarrow$ Consistency,
  - $\rightsquigarrow$ Canonicity,
  - $\rightsquigarrow$ Decidability of the Type-Checking

Extending Type Theory with Forcing

Joint work with Tabareau & Sozeau (LICS'11)

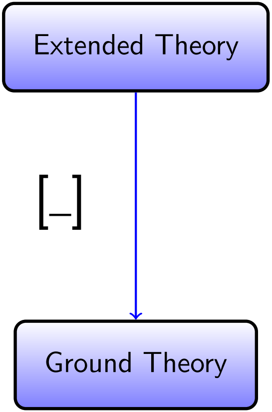
- Internalization of *Presheaf Models* in *Martin-Löf Type Theory*.

- Allow to extend syntactically MLTT with new principles.

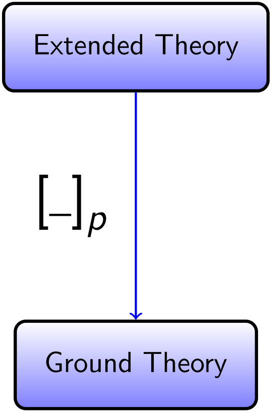
- Keep good properties of the theory:
  - $\leadsto$ Consistency,
  - $\leadsto$ Canonicity,
  - $\leadsto$ Decidability of the Type-Checking

- Impletementation for Coq (with proof-irrelevance in conversion).

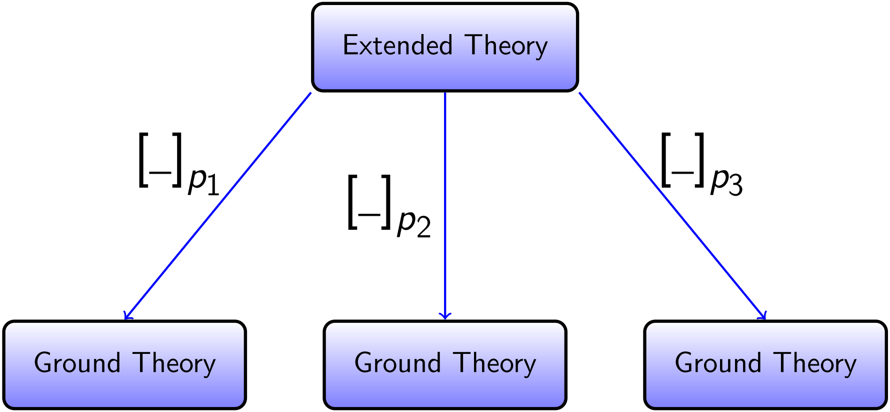
Presheaf Translation



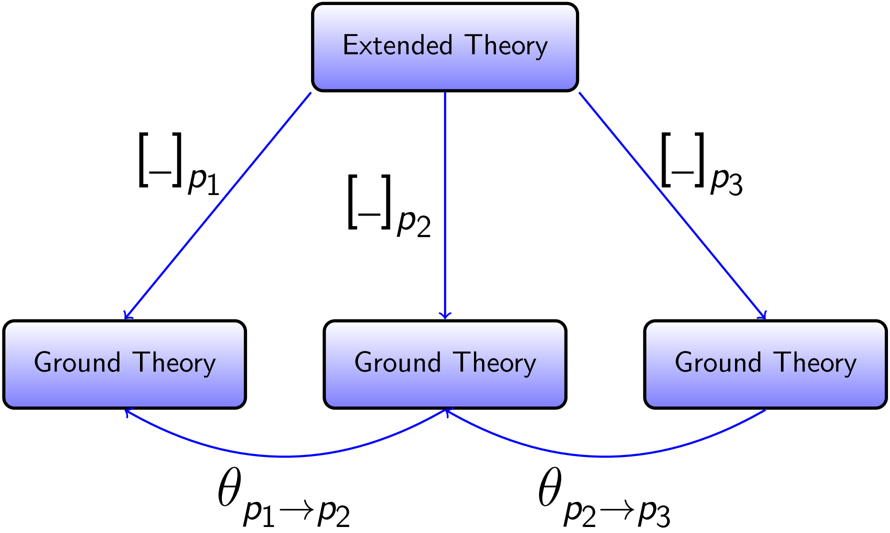
Presheaf Translation



Presheaf Translation



Presheaf Translation



Focus: Translation of the Dependent Product

- $[\![\Pi x : T.U]\!]_p^\sigma$ is defined as

$$\Pi q : \mathcal{P}_p . \Pi x : [\![T]\!]_q^\sigma . [\![U]\!]_q^{\sigma \cdot (x, T, q)}$$

$\rightsquigarrow$ Like $p \Vdash T \Rightarrow U$ is usually defined as

$$\forall q \leq p . (q \Vdash T) \Rightarrow (q \Vdash U)$$

Focus: Translation of the Dependent Product

- $[\![\Pi x : T.U]\!]_p^\sigma$ is defined as

$$\Pi q : \mathcal{P}_p.\Pi x : [\![T]\!]_q^\sigma.[\![U]\!]_q^{\sigma\cdot(x,T,q)}$$

$\rightsquigarrow$ Like $p \Vdash T \Rightarrow U$ is usually defined as

$$\forall q \leq p.(q \Vdash T) \Rightarrow (q \Vdash U)$$

- **comm**$_\Pi(f, T, U, p)$ enforces f to satisfy

$$\begin{array}{ccc}
[\![T]\!]_p^\sigma & \xrightarrow{f_p} & [\![U]\!]_p^\sigma i \\
{\scriptstyle \theta_{p\to q}^{\sigma,T}}\downarrow & & \downarrow{\scriptstyle \theta_{p\to q}^{\sigma,U}} \\
[\![T]\!]_q^\sigma & \xrightarrow[f_q]{} & [\![U]\!]_q^\sigma
\end{array}$$

Focus: Translation of the Dependent Product

- $[\![\Pi x : T.U]\!]_p^\sigma$ is defined as

$$\{f : \Pi q : \mathcal{P}_p.\Pi x : [\![T]\!]_q^\sigma.[\![U]\!]_q^{\sigma\cdot(x,T,q)} \mid \mathbf{comm}_\Pi(f, T, U, p)\}$$

$\rightsquigarrow$ Like $p \Vdash T \Rightarrow U$ is usually defined as

$$\forall q \leq p.(q \Vdash T) \Rightarrow (q \Vdash U)$$

- $\mathbf{comm}_\Pi(f, T, U, p)$ enforces f to satisfy

$$\begin{array}{ccc}
[\![T]\!]_p^\sigma & \xrightarrow{f_p} & [\![U]\!]_p^\sigma i \\
{\scriptstyle \theta_{p\to q}^{\sigma,T}}\downarrow & & \downarrow{\scriptstyle \theta_{p\to q}^{\sigma,U}} \\
[\![T]\!]_q^\sigma & \xrightarrow[f_q]{} & [\![U]\!]_q^\sigma
\end{array}$$

Forcing Layer

- $MLTT_{\mathcal{F}}$: extend $MLTT$ with new constants $\vdash_{\mathcal{F}} c_1 : T_1, \ldots c_n : T_n$
  $\rightsquigarrow$ c_k does not appear in T_j for $j \leq k$.

- $\mathcal{F}[\_]_p$ extends the translation $[\_]_p$ to $MLTT_{\mathcal{F}}$
  $\rightsquigarrow$ $\mathcal{F}[c_i]_p$ is provided.

Check that $p : \mathcal{P}_p \vdash \mathcal{F}[c_i]_p : \mathcal{F}[\![T_i]\!]_p$, then:

Theorem

If $\Gamma \vdash_{\mathcal{F}} M : T$, then $\mathcal{F}[\Gamma]^\sigma \vdash \mathcal{F}[M]_p^\sigma : \mathcal{F}[\![T]\!]_p^\sigma$.

An Example: Guarded Recursive Types

Define as the following forcing layer over $\mathcal{P} = (\mathrm{Nat}, \leq)$

An Example: Guarded Recursive Types

Define as the following forcing layer over $\mathcal{P} = (\mathrm{Nat}, \leq)$

- Guard on types: $\triangleright : \mathcal{U} \to \mathcal{U}$
- Fixpoints on univers: $\mathrm{fix} : \Pi T : \mathcal{U}.(\triangleright T \to T) \to T$
- $\mathrm{fold}, \mathrm{unfold}, \ldots$

An Example: Guarded Recursive Types

Define as the following forcing layer over $\mathcal{P} = (\mathrm{Nat}, \leq)$

- Guard on types: $\triangleright : \mathcal{U} \to \mathcal{U}$
- Fixpoints on univers: $\mathrm{fix} : \Pi T : \mathcal{U}.(\triangleright T \to T) \to T$
- $\mathrm{fold}, \mathrm{unfold}, \ldots$
- Relation with contractive maps by Birkedal & Mogelberg (LICS'13)

An Example: Guarded Recursive Types

Define as the following forcing layer over $\mathcal{P} = (\mathrm{Nat}, \leq)$

- Guard on types: $\triangleright : \mathcal{U} \to \mathcal{U}$
- Fixpoints on univers: $\mathrm{fix} : \Pi T : \mathcal{U}.(\triangleright T \to T) \to T$
- $\mathrm{fold}, \mathrm{unfold}, \ldots$
- Relation with contractive maps by Birkedal & Mogelberg (LICS'13)

Impletementation in Coq:

$$
\begin{array}{rl}
\mathcal{F}[\triangleright]_p^\sigma \stackrel{\mathit{def}}{=} & \lambda q : \mathrm{Nat}_p.\lambda T : [\![\mathcal{U}]\!]_q^\sigma. \\
& \quad (\lambda r : \mathrm{Nat}_q.\texttt{match } r \texttt{ with} \\
& \qquad \mid 0 \texttt{ => } \mathrm{Unit} \\
& \qquad \mid \mathrm{S}r' \texttt{ => } (\pi_1 T) r' \\
& \quad , \lambda r : \mathrm{Nat}_q.\lambda t : \mathrm{Nat}_r.\lambda x : U_r.\texttt{match } t \texttt{ with} \\
& \qquad \mid 0 \texttt{ => } \mathrm{unit} \\
& \qquad \mid \mathrm{S}t' \texttt{ => } (\pi_2 T)\,(\mathrm{Pred}\ r)\ t'\ x)
\end{array}
$$

Coherence issues

- Intentional type theory : Distinction betwen conversion ($\equiv$) and propositional equality ($=_T$).

Coherence issues

- Intentional type theory : Distinction betwen conversion ($\equiv$) and propositional equality ($=_T$).

- Coherence issues arise:
  - $\rightsquigarrow$ $[\![T\{M/x\}]\!]_p \not\equiv [\![T]\!]_p\{[M]_p/x\}$,
  - $\rightsquigarrow$ Link with coherence problems with categorical models of Dependent Types (work of Curien, Hofmann).

Coherence issues

- Intentional type theory : Distinction betwen conversion ($\equiv$) and propositional equality ($=_T$).
- Coherence issues arise:
  - $\rightsquigarrow$ $\llbracket T\{M/x\}\rrbracket_p \not\equiv \llbracket T\rrbracket_p \{[M]_p/x\}$,
  - $\rightsquigarrow$ Link with coherence problems with categorical models of Dependent Types (work of Curien, Hofmann).
- but we can build a term of type $\llbracket T\{M/x\}\rrbracket_p =_{\mathcal{U}} \llbracket T\rrbracket_p \{[M]_p/x\}$,
  - $\rightsquigarrow$ Use it to perform rewriting in the translation,
  - $\rightsquigarrow$ Need explicit coercions of conversion.

Towards automatizable proofs of equivalence

Concrete Logical Relations

Game Semantics
$[\![\Gamma \vdash M_1 : \tau]\!] = [\![\Gamma \vdash M_2 : \tau]\!]$

$\updownarrow$

Trace Semantics
$[\Gamma \vdash M_1 : \tau] = [\Gamma \vdash M_2 : \tau]$

$\updownarrow$

Concrete Logical Relations
$(M_1, M_2) \in \mathcal{V}[\![\tau]\!]_e\, w$

$\updownarrow$

Temporal Logical Relations
$w \models \mathbb{E}[\![\tau]\!](M_1, M_2)$

Logical Relations

Binary relations $\mathcal{E}[\![\tau]\!], \mathcal{V}[\![\tau]\!]$ on closed terms and values

$\rightsquigarrow$ inductively defined on types.

$$\mathcal{V}[\![\mathrm{Int}]\!] \stackrel{def}{=} \{(n, n) \mid n \in \mathbb{Z}\}$$

$$\mathcal{V}[\![\tau \to \sigma]\!] \stackrel{def}{=} \{(\lambda x_1.M_1, \lambda x_2.M_2) \mid \forall (v_1, v_2) \in \mathcal{V}[\![\tau]\!] . \\ ((\lambda x_1.M_1)v_1, (\lambda x_2.M_2)v_2) \in \mathcal{E}[\![\sigma]\!]\}$$

$$\mathcal{E}[\![\tau]\!] \stackrel{def}{=} \{(M_1, M_2) \mid (M_1 \Uparrow \wedge M_2 \Uparrow) \\ \vee((M_1 \mapsto^* v_1) \wedge (M_2 \mapsto^* v_2) \wedge (v_1, v_2) \in \mathcal{V}[\![\tau]\!]\}$$

Kripke Logical Relations (Pitts & Stark, Ahmed, Birkedal, Dreyer, ...)

Extension to languages with references.

- $\leadsto$ Need *worlds*, i.e. invariants on heaps,
- $\leadsto$ which can evolve w.r.t. control flow of programs
- $\leadsto$ Parametrize the definition of logical relations with such worlds.

$$\mathcal{E}\,[\![\tau]\!]\,w \overset{def}{=} \Big\{(M_1, M_2) \mid \forall(h_1, h_2) : w.\big((M_1, h_1) \Uparrow \wedge (M_2, h_2) \Uparrow\big) \\ \vee\big(((M_1, h_1) \mapsto^* (v_1, h_1')) \wedge ((M_2, h_2) \mapsto^* (v_2, h_2')) \\ \exists w' \sqsupseteq w.(h_1', h_2') : w' \wedge (v_1, v_2) \in \mathcal{V}\,[\![\tau]\!]\,w'\big)\Big\}$$

Toward *simple* proofs of equivalence

Starting Point : Kripke logical relations with STS of heap-invariants as world (Dreyer, Neis & Birkedal).

Remove all the quantifier on “complex” elements of their definition:

Toward *simple* proofs of equivalence

Starting Point : Kripke logical relations with STS of heap-invariants as world (Dreyer, Neis & Birkedal).

Remove all the quantifier on “complex” elements of their definition:

- Quantification over **applicative contexts** in $\mathcal{E}\,[\![\tau]\!]\,w$ (Biorthogonality)
  - $\leadsto$ Direct-style definition.

Toward *simple* proofs of equivalence

Starting Point : Kripke logical relations with STS of heap-invariants as world (Dreyer, Neis & Birkedal).

Remove all the quantifier on "complex" elements of their definition:

- Quantification over **applicative contexts** in $\mathcal{E}\,[\![\tau]\!]\,w$ (Biorthogonality)
  - $\rightsquigarrow$ Direct-style definition.
- Quantification over **functional values** in $\mathcal{V}\,[\![\tau \rightarrow \sigma]\!]\,w$
  - $\rightsquigarrow$ When τ is functional,
  - $\rightsquigarrow$ Use fresh free variables instead.

Toward *simple* proofs of equivalence

Starting Point : Kripke logical relations with STS of heap-invariants as world (Dreyer, Neis & Birkedal).

Remove all the quantifier on "complex" elements of their definition:

- Quantification over **applicative contexts** in $\mathcal{E}\llbracket\tau\rrbracket\, w$ (Biorthogonality)
  - $\leadsto$ Direct-style definition.
- Quantification over **functional values** in $\mathcal{V}\llbracket\tau \to \sigma\rrbracket\, w$
  - $\leadsto$ When τ is functional,
  - $\leadsto$ Use fresh free variables instead.
- Quantification over **functional values** in $\mathcal{V}\llbracket\mathrm{ref}(\tau \to \sigma)\rrbracket\, w$
  - $\leadsto$ Deal with disclosed locations externally, in the definition of $(h_1, h_2) : w$.
  - $\leadsto$ Remove the circularity between logical relations and worlds.

Toward *simple* proofs of equivalence

Starting Point : Kripke logical relations with STS of heap-invariants as world (Dreyer, Neis & Birkedal).

Remove all the quantifier on “complex” elements of their definition:

- Quantification over **applicative contexts** in $\mathcal{E}\llbracket\tau\rrbracket\, w$ (Biorthogonality)
  - $\rightsquigarrow$ Direct-style definition.
- Quantification over **functional values** in $\mathcal{V}\llbracket\tau \to \sigma\rrbracket\, w$
  - $\rightsquigarrow$ When τ is functional,
  - $\rightsquigarrow$ Use fresh free variables instead.
- Quantification over **functional values** in $\mathcal{V}\llbracket\mathrm{ref}(\tau \to \sigma)\rrbracket\, w$
  - $\rightsquigarrow$ Deal with disclosed locations externally, in the definition of $(h_1, h_2) : w$.
  - $\rightsquigarrow$ Remove the circularity between logical relations and worlds.
- Quantification over **new disjoint invariants** in $w' \sqsupseteq w$
  - $\rightsquigarrow$ Use a fixed transition system instead.

Toward *simple* proofs of equivalence

Starting Point : Kripke logical relations with STS of heap-invariants as world (Dreyer, Neis & Birkedal).

Remove all the quantifier on "complex" elements of their definition:

- Quantification over **applicative contexts** in $\mathcal{E}[\![\tau]\!]\,w$ (Biorthogonality)
  - $\rightsquigarrow$ Direct-style definition.
- Quantification over **functional values** in $\mathcal{V}[\![\tau \to \sigma]\!]\,w$
  - $\rightsquigarrow$ When τ is functional,
  - $\rightsquigarrow$ Use fresh free variables instead.
- Quantification over **functional values** in $\mathcal{V}[\![\mathrm{ref}(\tau \to \sigma)]\!]\,w$
  - $\rightsquigarrow$ Deal with disclosed locations externally, in the definition of $(h_1, h_2) : w$.
  - $\rightsquigarrow$ Remove the circularity between logical relations and worlds.
- Quantification over **new disjoint invariants** in $w' \sqsupseteq w$
  - $\rightsquigarrow$ Use a fixed transition system instead.

Give rise to our definition of "Concrete Logical Relations"

Concrete Logical Relations

- All what we need to reason on equivalence:

Concrete Logical Relations

- All what we need to reason on equivalence:
  - $\rightsquigarrow$ A transition system representing the control flow between the term and its environment (i.e. contexts)

Concrete Logical Relations

- All what we need to reason on equivalence:
  - $\rightsquigarrow$ A transition system representing the control flow between the term and its environment (i.e. contexts)
  - $\rightsquigarrow$ Private transitions: only the term can take them,

Concrete Logical Relations

- All what we need to reason on equivalence:
  - $\rightsquigarrow$ A transition system representing the control flow between the term and its environment (i.e. contexts)
  - $\rightsquigarrow$ Private transitions: only the term can take them,
  - $\rightsquigarrow$ Public transitions: execution to a value, so the environment can take them (well-bracketing),

Concrete Logical Relations

- All what we need to reason on equivalence:
  - $\rightsquigarrow$ A transition system representing the control flow between the term and its environment (i.e. contexts)
  - $\rightsquigarrow$ Private transitions: only the term can take them,
  - $\rightsquigarrow$ Public transitions: execution to a value, so the environment can take them (well-bracketing),
  - $\rightsquigarrow$ Labels on transitions: heap-invariants, disclosure process of locations.

Concrete Logical Relations

- All what we need to reason on equivalence:
  - ⇝ A transition system representing the control flow between the term and its environment (i.e. contexts)
  - ⇝ Private transitions: only the term can take them,
  - ⇝ Public transitions: execution to a value, so the environment can take them (well-bracketing),
  - ⇝ Labels on transitions: heap-invariants, disclosure process of locations.
- Reason on **open** terms with free **functional** variables
  - ⇝ Make the full control flow apparent in the operational reduction.

Soundness and Completeness

- Proofs by induction on typing judgment seems impossible (no "compatibility lemmas").

Soundness and Completeness

- Proofs by induction on typing judgment seems impossible (no "compatibility lemmas").
- Correspondence with Trace semantics.

Soundness and Completeness

- Proofs by induction on typing judgment seems impossible (no "compatibility lemmas").
- Correspondence with Trace semantics.
- Soundness:
  - $\rightsquigarrow$ Introduce Kripke trace semantics,
  - $\rightsquigarrow$ Perform "surgery" on traces.

Soundness and Completeness

- Proofs by induction on typing judgment seems impossible (no "compatibility lemmas").
- Correspondence with Trace semantics.
- Soundness:
  - $\rightsquigarrow$ Introduce Kripke trace semantics,
  - $\rightsquigarrow$ Perform "surgery" on traces.
- Completeness:
  - $\rightsquigarrow$ No more biorthogonality,
  - $\rightsquigarrow$ Need "adequate" LTS: dual of Kripke Logical Relations,
  - $\rightsquigarrow$ Show that it always exists: exhaustive worlds.

Callback with lock

$$M_1^{cbl} \quad = \quad C\ [\mathtt{f}(); \mathtt{x} :=!\mathtt{x} + 1]$$
$$M_2^{cbl} \quad = \quad C\ [\mathtt{let}\, \mathtt{n} =!\mathtt{x}\, \mathtt{in}\, \mathtt{f}(); \mathtt{x} := \mathtt{n} + 1] \qquad \text{where}$$

$$C \quad = \quad \mathtt{let}\, \mathtt{b} = \mathrm{ref}\, \textbf{true}\ \mathtt{in}\ \mathtt{let}\ \mathtt{x} = \mathrm{ref}\, 0\, \mathtt{in}$$
$$\langle \lambda \mathtt{f}.\mathtt{if}\ !\mathtt{b}\ \mathtt{then}\ \mathtt{b} := \textbf{false};\ \bullet\ ; \mathtt{b} := \textbf{true}\ \mathtt{else}\ (), \lambda\_.!\mathtt{x} \rangle$$

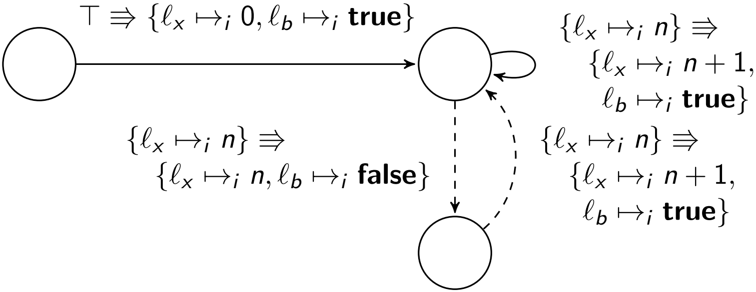
Callback with lock

$$M_1^{cbl} \;=\; C\;[\mathtt{f}();\mathtt{x} := !\mathtt{x}+1]$$

$$M_2^{cbl} \;=\; C\;[\mathtt{let\,n} = !\mathtt{x}\ \mathtt{in}\ \mathtt{f}();\mathtt{x} := \mathtt{n}+1] \qquad \text{where}$$

$$C \;=\; \mathtt{let\,b} = \mathrm{ref}\ \mathbf{true}\ \mathtt{in}\ \mathtt{let\ x} = \mathrm{ref}\ 0\ \mathtt{in}$$

$$\langle \lambda \mathtt{f}.\mathtt{if}\ !\mathtt{b}\ \mathtt{then}\ \mathtt{b} := \mathbf{false};\ \bullet\ ;\mathtt{b} := \mathbf{true}\ \mathtt{else}\ (), \lambda\_.!\mathtt{x}\rangle$$



Automatically generated temporal formula

$\mathbb{E}[\![\tau]\!](M_1^{clb}, M_2^{cbl})$ is equal to

$$\mathcal{H}\mathtt{N}_0.(\mathcal{N}_1\ell_2.(\mathcal{N}_1\ell_1.(\mathcal{N}_2\ell_4.(\mathcal{N}_2\ell_3.(\mathtt{X}((\ell_2 \mapsto_1 0) \wedge (\ell_1 \mapsto_1 \textbf{true}) \wedge (\ell_4 \mapsto_2 0) \wedge (\ell_3 \mapsto_2 \textbf{true}) \wedge$$
$$((\Box(\mathcal{H}\mathtt{N}_5.(\forall \mathtt{x}_6, \mathtt{x}_7, \mathtt{x}_8, \mathtt{x}_9.((((\ell_2 \mapsto_1 \mathtt{x}_6) \wedge (\ell_1 \mapsto_1 \mathtt{x}_7) \wedge (\ell_4 \mapsto_2 \mathtt{x}_8) \wedge (\ell_3 \mapsto_2 \mathtt{x}_9)) \Rightarrow$$
$$((\mathtt{X}(((\mathtt{x}_7 = \textbf{true}) \wedge (\mathtt{x}_9 = \textbf{true})) \Rightarrow ((\ell_2 \mapsto_1 \mathtt{x}_6) \wedge (\ell_1 \mapsto_1 \textbf{false}) \wedge (\ell_4 \mapsto_2 \mathtt{x}_8) \wedge (\ell_3 \mapsto_2 \textbf{false}) \wedge$$
$$(\Box_{\mathrm{pub}}(\forall \mathtt{x}_{10}, \mathtt{x}_{11}, \mathtt{x}_{13}, \mathtt{x}_{14}.((((\ell_2 \mapsto_1 \mathtt{x}_{10}) \wedge (\ell_1 \mapsto_1 \mathtt{x}_{11}) \wedge (\ell_4 \mapsto_2 \mathtt{x}_{13}) \wedge (\ell_3 \mapsto_2 \mathtt{x}_{14})) \Rightarrow$$
$$(\mathtt{X}(\forall \mathtt{x}_{12}, \mathtt{x}_{15}.(((\mathtt{x}_{12} = \mathtt{x}_{10} + 1) \wedge (\mathtt{x}_{15} = \mathtt{x}_8 + 1)) \Rightarrow ((\ell_2 \mapsto_1 \mathtt{x}_{12}) \wedge (\ell_1 \mapsto_1 \textbf{true}) \wedge$$
$$(\ell_4 \mapsto_2 \mathtt{x}_{15}) \wedge (\ell_3 \mapsto_2 \textbf{true}) \wedge (\mathbf{P}_{\mathrm{pub}}(\mathtt{N}_5)))))))))))) \wedge (\mathtt{not}((\mathtt{x}_7 = \textbf{true}) \wedge (\mathtt{x}_9 = \textbf{false}))) \wedge$$
$$(\mathtt{not}((\mathtt{x}_7 = \textbf{false}) \wedge (\mathtt{x}_9 = \textbf{true}))) \wedge (\mathtt{X}(((\mathtt{x}_7 = \textbf{false}) \wedge (\mathtt{x}_9 = \textbf{false})) \Rightarrow$$
$$((\ell_2 \mapsto_1 \mathtt{x}_6) \wedge (\ell_1 \mapsto_1 \mathtt{x}_7) \wedge (\ell_4 \mapsto_2 \mathtt{x}_8) \wedge (\ell_3 \mapsto_2 \mathtt{x}_9) \wedge (\mathbf{P}_{\mathrm{pub}}(\mathtt{N}_5))))))))))$$
$$\wedge(\Box(\mathcal{H}\mathtt{N}_16.(\forall[\mathtt{x}_{17}, \mathtt{x}_{18}, \mathtt{x}_{19}, \mathtt{x}_{20}].((((\ell_2 \mapsto_1 \mathtt{x}_{17}) \wedge (\ell_1 \mapsto_1 \mathtt{x}_{18}) \wedge (\ell_4 \mapsto_2 \mathtt{x}_{19}) \wedge (\ell_3 \mapsto_2 \mathtt{x}_{20})) \Rightarrow$$
$$(\mathtt{X}((\ell_2 \mapsto_1 \mathtt{x}_{17}) \wedge (\ell_1 \mapsto_1 \mathtt{x}_{18}) \wedge (\ell_4 \mapsto_2 \mathtt{x}_{19}) \wedge (\ell_3 \mapsto_2 \mathtt{x}_{20}) \wedge (\mathtt{x}_{17} = \mathtt{x}_{19}) \wedge (\mathbf{P}_{\mathrm{pub}}(\mathtt{N}_16)))))))))$$
$$\wedge(\mathbf{P}_{\mathrm{pub}}(\mathtt{N}_0))))))))$$

Translation to SMT-LIB

```
 (assert (exists ((s21 Int)(h22 Heap)(h23 Heap)(l2 Int)(l1 Int)) (and (not (= l1 l2))
(exists ((l4 Int)(l3 Int)) (and (not (= l3 l4)) (exists ((s25 Int)(h26 Heap)(h27 Heap)(S28 LocSpan))
(and (TransPriv s21 s25 h22 h23 h26 h27 S28) (and (= (select h26 l2) 0) (= (select h26 l1) 0)
(= (select h27 l4) 0) (= (select h27 l3) 0) (and (forall ((s29 Int)(h30 Heap)(h31 Heap)(S32 LocSpan))
(=> (TransPrivT s25 s29 h26 h27 h30 h31 S32) (forall ((x6 Int)(x7 Int)(x8 Int)(x9 Int))
(=> (and (= (select h30 l2) x6) (= (select h30 l1) x7) (= (select h31 l4) x8) (= (select h31 l3) x9) )
 (and (exists ((s33 Int)(h34 Heap)(h35 Heap)(S36 LocSpan)) (and (TransPriv s29 s33 h30 h31 h34 h35 S36)
 (=> (and (= x7 0) (= x9 0) ) (and (= (select h34 l2) x6) (= (select h34 l1) 1) (= (select h35 l4) x8)
(= (select h35 l3) 1) (forall ((s37 Int)(h38 Heap)(h39 Heap)(S40 LocSpan))
 (=> (TransPubT s33 s37 h34 h35 h38 h39 S40) (forall ((x10 Int)(x11 Int)(x13 Int)(x14 Int))
 (=> (and (= (select h38 l2) x10) (= (select h38 l1) x11) (= (select h39 l4) x13) (= (select h39 l3) x14))
 (exists ((s41 Int)(h42 Heap)(h43 Heap)(S44 LocSpan)) (and (TransPriv s37 s41 h38 h39 h42 h43 S44)
 (forall ((x12 Int)(x15 Int)) (=> (and (= x12 (+ x10 1)) (= x15 (+ x8 1)) ) (and (= (select h42 l2) x12)
 (= (select h42 l1) 0) (= (select h43 l4) x15) (= (select h43 l3) 0)
 (TransPub s29 s41 h30 h31 h42 h43 S44)))))))))))))
 (not (and (= x7 0) (= x9 1) )) (not (and (= x7 1) (= x9 0) ))
 (exists ((s45 Int)(h46 Heap)(h47 Heap)(S48 LocSpan)) (and (TransPriv s29 s45 h30 h31 h46 h47 S48)
 (=> (and (= x7 1) (= x9 1) ) (and (= (select h46 l2) x6) (= (select h46 l1) x7) (= (select h47 l4) x8)
 (= (select h47 l3) x9) (TransPub s29 s45 h30 h31 h46 h47 S48))))))))))
 (forall ((s49 Int)(h50 Heap)(h51 Heap)(S52 LocSpan))(=> (TransPrivT s25 s49 h26 h27 h50 h51 S52)
 (forall ((x17 Int)(x18 Int)(x19 Int)(x20 Int)) (=> (and (= (select h50 l2) x17) (= (select h50 l1) x18)
 (= (select h51 l4) x19) (= (select h51 l3) x20) ) (exists ((s53 Int)(h54 Heap)(h55 Heap)(S56 LocSpan))
 (and (TransPriv s49 s53 h50 h51 h54 h55 S56) (and (= (select h54 l2) x17) (= (select h54 l1) x18)
 (= (select h55 l4) x19) (= (select h55 l3) x20) (= x17 x19) (TransPub s49 s53 h50 h51 h54 h55 S56))))))))
 (TransPub s21 s25 h22 h23 h26 h27 S28)))))))))

 (check-sat)
```

What's Next?

- Extend the presheaves translation with forcing conditions as categories
  - $\rightsquigarrow$ Useful for an implementation of a Nominal Dependent Type Theory ?
- Reason on recursive programs:
  - $\rightsquigarrow$ Combination with Higher-order Recursive Schemes (Ong et al.) ?
- Decidability results:
  - $\rightsquigarrow$ Semi-decidability: generate worlds,
  - $\rightsquigarrow$ Full decidability: reason on *adequate worlds*,
  - $\rightsquigarrow$ Decidability of the contextual equivalence of ν-calculus ?
- Polymorphic languages ?

A Unified Theory ?

$$\left.\begin{array}{l}\text{Kripke Logical relations with worlds as} \\ \text{Bisimulations over} \\ \text{Traces generated by} \\ \text{Game Semantics as} \\ \text{Presheaves over}\end{array}\right\} \text{LTS}$$

- $\rightsquigarrow$ Open maps (Joyal, Nielsen & Winskel)
- $\rightsquigarrow$ Innocent Strategies as Presheaves (Hirchowitz & Pous)
- $\rightsquigarrow$ Transition systems over games (Levy & Staton, LICS'14)