A Proof-theoretic Characterization of Independence in Type Theory

Yuting Wang [1] Kaustuv Chaudhuri [2]

[1]University of Minnesota, Twin Cities, USA

[2]Inria & LIX/École polytechnique, France

TLCA, July 2015, Warsaw

Motivation

Formalizing transportation of theorems and proofs about type theories in different contexts.

Motivation

Formalizing transportation of theorems and proofs about type theories in different contexts.

Example:

$$\texttt{z} : \texttt{nat} \quad \texttt{s} : \texttt{nat} \to \texttt{nat}$$

$$\texttt{leaf} : (\texttt{nat} \to \texttt{bt}) \to \texttt{bt} \quad \texttt{node} : \texttt{bt} \to \texttt{bt} \to \texttt{bt}$$

Motivation

Formalizing transportation of theorems and proofs about type theories in different contexts.

Example:

$$\mathtt{z} : \mathtt{nat} \quad \mathtt{s} : \mathtt{nat} \to \mathtt{nat}$$
$$\mathtt{leaf} : (\mathtt{nat} \to \mathtt{bt}) \to \mathtt{bt} \quad \mathtt{node} : \mathtt{bt} \to \mathtt{bt} \to \mathtt{bt}$$

Suppose given some property P about `bt` we prove

$$\forall b : \mathtt{bt}. P(b).$$

Motivation

Formalizing transportation of theorems and proofs about type theories in different contexts.

Example:

$$\mathtt{z} : \mathtt{nat} \quad \mathtt{s} : \mathtt{nat} \to \mathtt{nat}$$

$$\mathtt{leaf} : (\mathtt{nat} \to \mathtt{bt}) \to \mathtt{bt} \quad \mathtt{node} : \mathtt{bt} \to \mathtt{bt} \to \mathtt{bt}$$

Suppose given some property P about `bt` we prove

$$\forall b : \mathtt{bt}.P(b).$$

Question: After adding $\mathtt{c} : \mathtt{nat}$ does the theorem still hold?

Motivation

Formalizing transportation of theorems and proofs about type theories in different contexts.

Example:

$$\mathtt{z} : \mathtt{nat} \quad \mathtt{s} : \mathtt{nat} \to \mathtt{nat}$$

$$\mathtt{leaf} : (\mathtt{nat} \to \mathtt{bt}) \to \mathtt{bt} \quad \mathtt{node} : \mathtt{bt} \to \mathtt{bt} \to \mathtt{bt}$$

Suppose given some property P about $\mathtt{bt}$ we prove

$$\forall b : \mathtt{bt}. P(b).$$

Question: After adding $\mathtt{c} : \mathtt{nat}$ does the theorem still hold?

Answer: Yes. Because $\mathtt{bt}$-terms (in normal form) cannot contain $\mathtt{nat}$-terms.

Independence

Terms of a certain type can not depend on that of another type.

Independence

Terms of a certain type can not depend on that of another type.

Definition (Independence)

The type τ_2 is **independent** of τ_1 in the context Γ if whenever $\Gamma, x{:}\tau_1 \vdash t : \tau_2$ holds for some t, the β-normal form of t does not contain x, *i.e.*, $\Gamma \vdash t : \tau_2$ holds. □

Independence

Terms of a certain type can not depend on that of another type.

Definition (Independence)

The type τ_2 is **independent** of τ_1 in the context Γ if whenever $\Gamma, x{:}\tau_1 \vdash t : \tau_2$ holds for some t, the β-normal form of t does not contain x, *i.e.*, $\Gamma \vdash t : \tau_2$ holds. □

Independence

- is a derived property of the given type theory
- can be used to formalize transportation of theorems

Independence

Terms of a certain type can not depend on that of another type.

Definition (Independence)

The type τ_2 is **independent** of τ_1 in the context Γ if whenever $\Gamma, x{:}\tau_1 \vdash t : \tau_2$ holds for some t, the β-normal form of t does not contain x, *i.e.*, $\Gamma \vdash t : \tau_2$ holds. □

Independence

- is a derived property of the given type theory
- can be used to formalize transportation of theorems

Example: `bt` is independent of `nat` in the last example.

Contributions (Overview)

Our contributions:

Contributions (Overview)

Our contributions:

- A methodology for formalizing proofs of independence

Contributions (Overview)

Our contributions:

- A methodology for formalizing proofs of independence
  - Encoding the type theory in a specification logic called HH
  - Proving independence in a reasoning logic called $\mathcal{G}$

Contributions (Overview)

Our contributions:

- A methodology for formalizing proofs of independence
  - Encoding the type theory in a specification logic called HH
  - Proving independence in a reasoning logic called $\mathcal{G}$
- An algorithm for automatically checking independence

Contributions (Overview)

Our contributions:

- A methodology for formalizing proofs of independence
  - Encoding the type theory in a specification logic called HH
  - Proving independence in a reasoning logic called $\mathcal{G}$
- An algorithm for automatically checking independence
  - Derive the independence relation from the typing context
  - Simultaneously generate a proof of independence

Contributions (Overview)

Our contributions:

- A methodology for formalizing proofs of independence
  - Encoding the type theory in a specification logic called HH
  - Proving independence in a reasoning logic called $\mathcal{G}$
- An algorithm for automatically checking independence
  - Derive the independence relation from the typing context
  - Simultaneously generate a proof of independence

We use the simply-typed λ-calculus (STLC) as an example.

Elaboration of Independence Proofs

We want to prove the following lemma by induction:

$\forall t$, if $\Gamma, x{:}\tau_1 \vdash t : \tau_2$ is derivable then so is $\Gamma \vdash t : \tau_2$.

Elaboration of Independence Proofs

We want to prove the following lemma by induction:

$\forall t$, if $\Gamma, x{:}\tau_1 \vdash t : \tau_2$ is derivable then so is $\Gamma \vdash t : \tau_2$.

Considering the independence of τ_2 to τ_1 alone is not enough.

Elaboration of Independence Proofs

We want to prove the following lemma by induction:

$\forall t$, if $\Gamma, x{:}\tau_1 \vdash t : \tau_2$ is derivable then so is $\Gamma \vdash t : \tau_2$.

Considering the independence of τ_2 to τ_1 alone is not enough.

Example: when t is an application $t_1\ t_2$:

$$\frac{\Gamma, x{:}\tau_1 \vdash t_1 : \tau \to \tau_2 \quad \Gamma, x{:}\tau_1 \vdash t_2 : \tau}{\Gamma, x{:}\tau_1 \vdash t_1\ t_2 : \tau_2}$$

Need to prove the independence of τ to τ_1 for the new type τ.

Elaboration of Independence Proofs

We want to prove the following lemma by induction:

$\forall t$, *if* $\Gamma, x{:}\tau_1 \vdash t : \tau_2$ *is derivable then so is* $\Gamma \vdash t : \tau_2$*.*

Considering the independence of τ_2 to τ_1 alone is not enough.

Example: when t is an application $t_1\ t_2$:

$$\frac{\Gamma, x{:}\tau_1 \vdash t_1 : \tau \to \tau_2 \quad \Gamma, x{:}\tau_1 \vdash t_2 : \tau}{\Gamma, x{:}\tau_1 \vdash t_1\ t_2 : \tau_2}$$

Need to prove the independence of τ to τ_1 for the new type τ.

Solution:

- Since the context Γ is fixed, it is possible to finitely characterize the types involved in the proof
- Prove the independence lemmas for these types simultaneously

Elaboration of Independence Proofs

We want to prove the following lemma by induction:

$\forall t$, *if* $\Gamma, x{:}\tau_1 \vdash t : \tau_2$ *is derivable then so is* $\Gamma \vdash t : \tau_2$*.*

Considering the independence of τ_2 to τ_1 alone is not enough.

Example: when t is an application $t_1\ t_2$:

$$\frac{\Gamma, x{:}\tau_1 \vdash t_1 : \tau \to \tau_2 \quad \Gamma, x{:}\tau_1 \vdash t_2 : \tau}{\Gamma, x{:}\tau_1 \vdash t_1\ t_2 : \tau_2}$$

Need to prove the independence of τ to τ_1 for the new type τ.

Solution:

- Since the context Γ is fixed, it is possible to finitely characterize the types involved in the proof
- Prove the independence lemmas for these types simultaneously

Realization: encode typing for the fixed context in a spec logic and do inductive proof on the encoding.

The Specification Logic HH

The specification logic is called the logic of *higher-order hereditary Harrop formulas* (HH):

The Specification Logic HH

The specification logic is called the logic of *higher-order hereditary Harrop formulas* (HH):

- Provides an adequate set of devices for formalizing SOS-style rules

The Specification Logic HH

The specification logic is called the logic of *higher-order hereditary Harrop formulas* (HH):

- Provides an adequate set of devices for formalizing SOS-style rules
- Formulas has the following normal form:

$$F \ ::= \ \forall \bar{x}{:}\bar{\tau}.\, F_1 \Rightarrow \cdots \Rightarrow F_n \Rightarrow A.$$

The Specification Logic HH

The specification logic is called the logic of *higher-order hereditary Harrop formulas* (HH):

- Provides an adequate set of devices for formalizing SOS-style rules
- Formulas has the following normal form:

$$F \ ::= \ \forall \bar{x}{:}\bar{\tau}.\, F_1 \Rightarrow \cdots \Rightarrow F_n \Rightarrow A.$$

- A sequent calculus for derive sequents of the form

$$\Gamma \vdash F \qquad (\Gamma = F_1, ..., F_n)$$

Γ is called the context and F is called the goal

The Specification Logic HH

The specification logic is called the logic of *higher-order hereditary Harrop formulas* (HH):

- Provides an adequate set of devices for formalizing SOS-style rules
- Formulas has the following normal form:

$$F ::= \forall \bar{x}{:}\bar{\tau}.\, F_1 \Rightarrow \cdots \Rightarrow F_n \Rightarrow A.$$

- A sequent calculus for derive sequents of the form

$$\Gamma \vdash F \qquad (\Gamma = F_1, ..., F_n)$$

  Γ is called the context and F is called the goal

- A derivation alternates between the following two phases:
  - Simplify the goal until it becomes atomic;
  - Perform backchaining on the atomic goal.

An Encoding of STLC in HH

The encoding is based on **types-as-predicates** principle:

An Encoding of STLC in HH

The encoding is based on **types-as-predicates** principle:

- Atomic types and constants are imported into HH signature

An Encoding of STLC in HH

The encoding is based on **types-as-predicates** principle:

- Atomic types and constants are imported into HH signature
- For every atomic type b, define a predicate $\hat{b} : b \to \circ$

An Encoding of STLC in HH

The encoding is based on **types-as-predicates** principle:

- Atomic types and constants are imported into HH signature
- For every atomic type b, define a predicate $\hat{b} : b \to \circ$
- Define a mapping $[\![-]\!]$ from STLC types τ to predicates $\tau \to \circ$:

$$
\begin{array}{lcll}
[\![b]\!] & = & \lambda t.\, \hat{b}\ t & \text{if } b \text{ is an atomic type.} \\
[\![\tau_1 \to \tau_2]\!] & = & \lambda t.\, \forall x{:}\tau_1.\, [\![\tau_1]\!]\ x \Rightarrow [\![\tau_2]\!]\ (t\ x) &
\end{array}
$$

An Encoding of STLC in HH

The encoding is based on **types-as-predicates** principle:

- Atomic types and constants are imported into HH signature
- For every atomic type b, define a predicate $\hat{b} : b \to \circ$
- Define a mapping $[\![-]\!]$ from STLC types τ to predicates $\tau \to \circ$:

$$\begin{array}{lcl} [\![b]\!] & = & \lambda t.\, \hat{b}\; t \quad \text{if } b \text{ is an atomic type.} \\ [\![\tau_1 \to \tau_2]\!] & = & \lambda t.\, \forall x{:}\tau_1.\, [\![\tau_1]\!]\; x \Rightarrow [\![\tau_2]\!]\; (t\; x) \end{array}$$

- A typing judgment $\Gamma \vdash t : \tau$ is encoded as an HH sequent

$$[\![\Gamma]\!] \vdash [\![\tau]\!]\; t$$

where $[\![\Gamma]\!] = \{[\![\tau_1]\!]\; x_1, \ldots, [\![\tau_n]\!]\; x_n\}$

Example of Encoding

Assume the following STLC signature Γ:

$$\mathtt{z} : \mathtt{nat} \quad \mathtt{s} : \mathtt{nat} \to \mathtt{nat}$$

$$\mathtt{leaf} : (\mathtt{nat} \to \mathtt{bt}) \to \mathtt{bt} \quad \mathtt{node} : \mathtt{bt} \to \mathtt{bt} \to \mathtt{bt}$$

Example of Encoding

Assume the following STLC signature Γ:

$$\mathtt{z} : \mathtt{nat} \quad \mathtt{s} : \mathtt{nat} \to \mathtt{nat}$$

$$\mathtt{leaf} : (\mathtt{nat} \to \mathtt{bt}) \to \mathtt{bt} \quad \mathtt{node} : \mathtt{bt} \to \mathtt{bt} \to \mathtt{bt}$$

- Define two predicates $\hat{\mathtt{nat}} : \mathtt{nat} \to \mathtt{o}$ and $\hat{\mathtt{bt}} : \mathtt{bt} \to \mathtt{o}$.

Example of Encoding

Assume the following STLC signature Γ:

$$\mathtt{z} : \mathtt{nat} \quad \mathtt{s} : \mathtt{nat} \to \mathtt{nat}$$

$$\mathtt{leaf} : (\mathtt{nat} \to \mathtt{bt}) \to \mathtt{bt} \quad \mathtt{node} : \mathtt{bt} \to \mathtt{bt} \to \mathtt{bt}$$

- Define two predicates $\hat{\mathtt{nat}} : \mathtt{nat} \to \mathrm{o}$ and $\hat{\mathtt{bt}} : \mathtt{bt} \to \mathrm{o}$.
- Constants are encoded as the following clauses

$$\hat{\mathtt{nat}}\ \mathtt{z}. \qquad \forall x. \hat{\mathtt{nat}}\ x \Rightarrow \hat{\mathtt{nat}}\ (\mathtt{s}\ x).$$

$$\forall x. (\forall y. \hat{\mathtt{nat}}\ y \Rightarrow \hat{\mathtt{bt}}\ (x\ y)) \Rightarrow \hat{\mathtt{bt}}\ (\mathtt{leaf}\ x).$$

$$\forall x\ y. \hat{\mathtt{bt}}\ x \Rightarrow \hat{\mathtt{bt}}\ y \Rightarrow \hat{\mathtt{bt}}\ (\mathtt{node}\ y\ x).$$

Example of Encoding

Assume the following STLC signature Γ:

$$\mathtt{z} : \mathtt{nat} \quad \mathtt{s} : \mathtt{nat} \to \mathtt{nat}$$

$$\mathtt{leaf} : (\mathtt{nat} \to \mathtt{bt}) \to \mathtt{bt} \quad \mathtt{node} : \mathtt{bt} \to \mathtt{bt} \to \mathtt{bt}$$

- Define two predicates $\hat{\mathtt{nat}} : \mathtt{nat} \to \mathtt{o}$ and $\hat{\mathtt{bt}} : \mathtt{bt} \to \mathtt{o}$.
- Constants are encoded as the following clauses

$$\hat{\mathtt{nat}}\ \mathtt{z}. \qquad \forall x.\, \hat{\mathtt{nat}}\ x \Rightarrow \hat{\mathtt{nat}}\ (\mathtt{s}\ x).$$

$$\forall x.\, (\forall y.\, \hat{\mathtt{nat}}\ y \Rightarrow \hat{\mathtt{bt}}\ (x\ y)) \Rightarrow \hat{\mathtt{bt}}\ (\mathtt{leaf}\ x).$$

$$\forall x\ y.\, \hat{\mathtt{bt}}\ x \Rightarrow \hat{\mathtt{bt}}\ y \Rightarrow \hat{\mathtt{bt}}\ (\mathtt{node}\ y\ x).$$

- Example of encoding typing judgments:

$$\Gamma, x : \mathtt{nat} \to \mathtt{bt}, y : \mathtt{bt} \vdash \mathtt{node}\ (\mathtt{leaf}\ x)\ y : \mathtt{bt}$$

  is encoded as the following HH sequent:

$$[\![\Gamma]\!], (\forall y.\, \hat{\mathtt{nat}}\ y \Rightarrow \hat{\mathtt{bt}}\ (x\ y)), \hat{\mathtt{bt}}\ y \vdash \hat{\mathtt{bt}}\ (\mathtt{node}\ (\mathtt{leaf}\ x)\ y)$$

Independence as Strengthening Lemmas

Now τ_2 is independent of τ_1 can be stated as follows:

If $[\![\Gamma]\!], [\![\tau_1]\!]\ x \vdash [\![\tau_2]\!]\ t$ *is derivable in* HH, *then so is* $[\![\Gamma]\!] \vdash [\![\tau_2]\!]\ t$*.*

Independence as Strengthening Lemmas

Now τ_2 is independent of τ_1 can be stated as follows:

If $[\![\Gamma]\!], [\![\tau_1]\!]\ x \vdash [\![\tau_2]\!]\ t$ is derivable in HH*, then so is $[\![\Gamma]\!] \vdash [\![\tau_2]\!]\ t$.*

It is an instance of **strengthening lemmas.**

Independence as Strengthening Lemmas

Now τ_2 is independent of τ_1 can be stated as follows:

If $[\![\Gamma]\!], [\![\tau_1]\!]\ x \vdash [\![\tau_2]\!]\ t$ *is derivable in* HH*, then so is* $[\![\Gamma]\!] \vdash [\![\tau_2]\!]\ t$*.*

It is an instance of **strengthening lemmas.**

Example: `bt` is independent of `nat`:

If $[\![\Gamma]\!], \hat{\mathtt{nat}}\ x \vdash \hat{\mathtt{bt}}\ t$ *is derivable, then so is* $[\![\Gamma]\!] \vdash \hat{\mathtt{bt}}\ t$*, where* Γ *is the signature in the last example.*

Independence as Strengthening Lemmas

Now τ_2 is independent of τ_1 can be stated as follows:

If $[\![\Gamma]\!], [\![\tau_1]\!]\ x \vdash [\![\tau_2]\!]\ t$ *is derivable in* HH*, then so is* $[\![\Gamma]\!] \vdash [\![\tau_2]\!]\ t$*.*

It is an instance of **strengthening lemmas.**

Example: `bt` is independent of `nat`:

If $[\![\Gamma]\!], \hat{\mathtt{nat}}\ x \vdash \hat{\mathtt{bt}}\ t$ *is derivable, then so is* $[\![\Gamma]\!] \vdash \hat{\mathtt{bt}}\ t$*, where* Γ *is the signature in the last example.*

Proof by Induction: the context may be dynamically extended when backchaining on:

$$\forall x. (\forall y. \hat{\mathtt{nat}}\ y \Rightarrow \hat{\mathtt{bt}}\ (x\ y)) \Rightarrow \hat{\mathtt{bt}}\ (\mathtt{leaf}\ x).$$

Independence as Strengthening Lemmas

Now τ_2 is independent of τ_1 can be stated as follows:

If $[\![\Gamma]\!], [\![\tau_1]\!]\ x \vdash [\![\tau_2]\!]\ t$ is derivable in HH*, then so is $[\![\Gamma]\!] \vdash [\![\tau_2]\!]\ t$.*

It is an instance of **strengthening lemmas.**

Example: `bt` is independent of `nat`:

If $[\![\Gamma]\!], \hat{\mathtt{nat}}\ x \vdash \hat{\mathtt{bt}}\ t$ is derivable, then so is $[\![\Gamma]\!] \vdash \hat{\mathtt{bt}}\ t$, where Γ is the signature in the last example.

Proof by Induction: the context may be dynamically extended when backchaining on:

$$\forall x. (\forall y. \hat{\mathtt{nat}}\ y \Rightarrow \hat{\mathtt{bt}}\ (x\ y)) \Rightarrow \hat{\mathtt{bt}}\ (\mathtt{leaf}\ x).$$

We prove a generalized lemma:

If $([\![\Gamma]\!], \Delta, \hat{\mathtt{nat}}\ x \vdash \hat{\mathtt{bt}}\ t)$ is derivable, then so is $([\![\Gamma]\!], \Delta \vdash \hat{\mathtt{bt}}\ t)$, where Δ is the dynamic context.

A Two-level Logic Approach

$\mathcal{G}$ is an intuitionistic logic base on Church's STT.

A Two-level Logic Approach

$\mathcal{G}$ is an intuitionistic logic base on Church's STT.

- Atomic predicates are interpreted through fixed-point definitions

A Two-level Logic Approach

$\mathcal{G}$ is an intuitionistic logic base on Church's STT.

- Atomic predicates are interpreted through fixed-point definitions

  Example: the definition for addition of naturals is:

  $$\texttt{add}\ \texttt{z}\ N\ N \triangleq \top; \qquad \texttt{add}\ (\texttt{s}\ N_1)\ N_2\ (\texttt{s}\ N_3) \triangleq \texttt{add}\ N_1\ N_2\ N_3$$

A Two-level Logic Approach

$\mathcal{G}$ is an intuitionistic logic base on Church's STT.

- Atomic predicates are interpreted through fixed-point definitions

  Example: the definition for addition of naturals is:

  $$\texttt{add}\ \texttt{z}\ N\ N \triangleq \top; \qquad \texttt{add}\ (\texttt{s}\ N_1)\ N_2\ (\texttt{s}\ N_3) \triangleq \texttt{add}\ N_1\ N_2\ N_3$$

- We can also give them a least (greatest) fixed point reading, leading to support for (co)-inductive reasoning

A Two-level Logic Approach

$\mathcal{G}$ is an intuitionistic logic base on Church's STT.

- Atomic predicates are interpreted through fixed-point definitions

  Example: the definition for addition of naturals is:

  $$\texttt{add}\ \texttt{z}\ N\ N \triangleq \top; \qquad \texttt{add}\ (\texttt{s}\ N_1)\ N_2\ (\texttt{s}\ N_3) \triangleq \texttt{add}\ N_1\ N_2\ N_3$$

- We can also give them a least (greatest) fixed point reading, leading to support for (co)-inductive reasoning
- A new quantifier ∇ for variables representing names.

A Two-level Logic Approach

$\mathcal{G}$ is an intuitionistic logic base on Church's STT.

- Atomic predicates are interpreted through fixed-point definitions

  Example: the definition for addition of naturals is:

  $$\texttt{add}\ \texttt{z}\ N\ N \triangleq \top; \qquad \texttt{add}\ (\texttt{s}\ N_1)\ N_2\ (\texttt{s}\ N_3) \triangleq \texttt{add}\ N_1\ N_2\ N_3$$

- We can also give them a least (greatest) fixed point reading, leading to support for (co)-inductive reasoning
- A new quantifier ∇ for variables representing names.

HH is encoded as a fixed-point definition for the predicate $\texttt{seq}$

A Two-level Logic Approach

$\mathcal{G}$ is an intuitionistic logic base on Church's STT.

- Atomic predicates are interpreted through fixed-point definitions

  Example: the definition for addition of naturals is:

  $$\texttt{add}\ \texttt{z}\ N\ N \triangleq \top; \qquad \texttt{add}\ (\texttt{s}\ N_1)\ N_2\ (\texttt{s}\ N_3) \triangleq \texttt{add}\ N_1\ N_2\ N_3$$

- We can also give them a least (greatest) fixed point reading, leading to support for (co)-inductive reasoning
- A new quantifier ∇ for variables representing names.

HH is encoded as a fixed-point definition for the predicate `seq`

- An HH sequent $L \vdash G$ is encoded as $\texttt{seq}\ L\ G$

A Two-level Logic Approach

$\mathcal{G}$ is an intuitionistic logic base on Church's STT.

- Atomic predicates are interpreted through fixed-point definitions

  Example: the definition for addition of naturals is:

  $$\texttt{add z } N\ N \triangleq \top; \qquad \texttt{add } (\texttt{s } N_1)\ N_2\ (\texttt{s } N_3) \triangleq \texttt{add } N_1\ N_2\ N_3$$

- We can also give them a least (greatest) fixed point reading, leading to support for (co)-inductive reasoning
- A new quantifier ∇ for variables representing names.

HH is encoded as a fixed-point definition for the predicate `seq`

- An HH sequent $L \vdash G$ is encoded as $\texttt{seq}\ L\ G$
- Derivation rules are encoded as definitions for `seq`

A Two-level Logic Approach

$\mathcal{G}$ is an intuitionistic logic base on Church's STT.

- Atomic predicates are interpreted through fixed-point definitions

  Example: the definition for addition of naturals is:

  $$\texttt{add}\ \texttt{z}\ N\ N \triangleq \top; \qquad \texttt{add}\ (\texttt{s}\ N_1)\ N_2\ (\texttt{s}\ N_3) \triangleq \texttt{add}\ N_1\ N_2\ N_3$$

- We can also give them a least (greatest) fixed point reading, leading to support for (co)-inductive reasoning
- A new quantifier ∇ for variables representing names.

HH is encoded as a fixed-point definition for the predicate $\texttt{seq}$

- An HH sequent $L \vdash G$ is encoded as $\texttt{seq}\ L\ G$
- Derivation rules are encoded as definitions for $\texttt{seq}$
- We write $\{L \vdash G\}$ for $\texttt{seq}\ L\ G$.

Formalizing Independence in $\mathcal{G}$

τ_2 is independent of τ_1 can be stated as follows in $\mathcal{G}$

$$\forall t. \nabla x. \{[\![\Gamma]\!], [\![\tau_1]\!]\ x \vdash [\![\tau_2]\!]\ (t\ x)\} \supset \exists t'. t = (\lambda y. t') \wedge \{[\![\Gamma]\!] \vdash [\![\tau_2]\!]\ t'\}.$$

Formalizing Independence in $\mathcal{G}$

τ_2 is independent of τ_1 can be stated as follows in $\mathcal{G}$

$$\forall t.\, \nabla x.\, \{[\![\Gamma]\!], [\![\tau_1]\!]\ x \vdash [\![\tau_2]\!]\ (t\ x)\} \supset \exists t'.\, t = (\lambda y.\, t') \wedge \{[\![\Gamma]\!] \vdash [\![\tau_2]\!]\ t'\}.$$

- The possibility that t may contain x is expressed by $t\ x$

Formalizing Independence in $\mathcal{G}$

τ_2 is independent of τ_1 can be stated as follows in $\mathcal{G}$

$$\forall t.\, \nabla x.\, \{[\![\Gamma]\!], [\![\tau_1]\!]\ x \vdash [\![\tau_2]\!]\ (t\ x)\} \supset \exists t'.\, t = (\lambda y.\, t') \wedge \{[\![\Gamma]\!] \vdash [\![\tau_2]\!]\ t'\}.$$

- The possibility that t may contain x is expressed by $t\ x$
- The ordering of binders t' and y in $\exists t'.\, t = (\lambda y.\, t')$ conclude that t does not contain x.

Formalizing Independence in $\mathcal{G}$

τ_2 is independent of τ_1 can be stated as follows in $\mathcal{G}$

$$\forall t.\, \nabla x.\, \{[\![\Gamma]\!], [\![\tau_1]\!]\ x \vdash [\![\tau_2]\!]\ (t\ x)\} \supset \exists t'.\, t = (\lambda y.\, t') \wedge \{[\![\Gamma]\!] \vdash [\![\tau_2]\!]\ t'\}.$$

- The possibility that t may contain x is expressed by $t\ x$
- The ordering of binders t' and y in $\exists t'.\, t = (\lambda y.\, t')$ conclude that t does not contain x.

Example: `bt` is independent of `nat`

$$\forall t.\, \nabla x.\, \{[\![\Gamma]\!], \hat{\texttt{nat}}\ x \vdash \hat{\texttt{bt}}\ (t\ x)\} \supset \exists t'.\, t = (\lambda y.\, t') \wedge \{[\![\Gamma]\!] \vdash \hat{\texttt{bt}}\ t'\}$$

Formalizing Independence in $\mathcal{G}$

τ_2 is independent of τ_1 can be stated as follows in $\mathcal{G}$

$$\forall t.\, \nabla x.\, \{[\![\Gamma]\!], [\![\tau_1]\!]\ x \vdash [\![\tau_2]\!]\ (t\ x)\} \supset \exists t'.\, t = (\lambda y.\, t') \wedge \{[\![\Gamma]\!] \vdash [\![\tau_2]\!]\ t'\}.$$

- The possibility that t may contain x is expressed by $t\ x$
- The ordering of binders t' and y in $\exists t'.\, t = (\lambda y.\, t')$ conclude that t does not contain x.

Example: bt is independent of nat

$$\forall t.\, \nabla x.\, \{[\![\Gamma]\!], \hat{\mathtt{nat}}\ x \vdash \hat{\mathtt{bt}}\ (t\ x)\} \supset \exists t'.\, t = (\lambda y.\, t') \wedge \{[\![\Gamma]\!] \vdash \hat{\mathtt{bt}}\ t'\}$$

We prove a generalized lemma:

$$\forall \Delta\ t.\, \nabla x.\, \mathtt{ctx}\ \Delta \supset \{[\![\Gamma]\!], \Delta, \hat{\mathtt{nat}}\ x \vdash \hat{\mathtt{bt}}\ (t\ x)\}$$
$$\supset \exists t'.\, t = (\lambda y.\, t') \wedge \{[\![\Gamma]\!], \Delta \vdash \hat{\mathtt{bt}}\ t'\}$$

where ctx defines the dynamically extended context

Automatically Checking Independence

Main Idea: To prove the strengthening lemma

$$\{\Gamma, a_1\ x \vdash a_2\ t\} \supset \{\Gamma \vdash a_2\ t\}$$

Show $a_1\ x$ is never used in the derivation of $\Gamma, a_1\ x \vdash a_2\ t$.

Automatically Checking Independence

Main Idea: To prove the strengthening lemma

$$\{\Gamma, a_1\ x \vdash a_2\ t\} \supset \{\Gamma \vdash a_2\ t\}$$

Show $a_1\ x$ is never used in the derivation of $\Gamma, a_1\ x \vdash a_2\ t$.

Algorithm for deriving the independence relation:

Automatically Checking Independence

Main Idea: To prove the strengthening lemma

$$\{\Gamma, a_1\ x \vdash a_2\ t\} \supset \{\Gamma \vdash a_2\ t\}$$

Show $a_1\ x$ is never used in the derivation of $\Gamma, a_1\ x \vdash a_2\ t$.

Algorithm for deriving the independence relation:

- For every predicate a, compute the context of sequents with atomic goals of head a.

Automatically Checking Independence

Main Idea: To prove the strengthening lemma

$$\{\Gamma, a_1\ x \vdash a_2\ t\} \supset \{\Gamma \vdash a_2\ t\}$$

Show $a_1\ x$ is never used in the derivation of $\Gamma, a_1\ x \vdash a_2\ t$.

Algorithm for deriving the independence relation:

- For every predicate a, compute the context of sequents with atomic goals of head a.
- By examining the context, compute a set $S(a)$ of all predicates that a can depend on.

Automatically Checking Independence

Main Idea: To prove the strengthening lemma

$$\{\Gamma, a_1\ x \vdash a_2\ t\} \supset \{\Gamma \vdash a_2\ t\}$$

Show $a_1\ x$ is never used in the derivation of $\Gamma, a_1\ x \vdash a_2\ t$.

Algorithm for deriving the independence relation:

- For every predicate a, compute the context of sequents with atomic goals of head a.
- By examining the context, compute a set $S(a)$ of all predicates that a can depend on.
- For any $b \notin S(a)$, every predicate in $S(a)$ is independent of b. Generate a proof for this by mutual induction.

Automatically Checking Independence

Main Idea: To prove the strengthening lemma

$$\{\Gamma, a_1\ x \vdash a_2\ t\} \supset \{\Gamma \vdash a_2\ t\}$$

Show $a_1\ x$ is never used in the derivation of $\Gamma, a_1\ x \vdash a_2\ t$.

Algorithm for deriving the independence relation:

- For every predicate a, compute the context of sequents with atomic goals of head a.
- By examining the context, compute a set $S(a)$ of all predicates that a can depend on.
- For any $b \notin S(a)$, every predicate in $S(a)$ is independent of b. Generate a proof for this by mutual induction.
- Since $a \in S(a)$, a is independent of b.

Automatically Checking Independence

Main Idea: To prove the strengthening lemma

$$\{\Gamma, a_1\ x \vdash a_2\ t\} \supset \{\Gamma \vdash a_2\ t\}$$

Show $a_1\ x$ is never used in the derivation of $\Gamma, a_1\ x \vdash a_2\ t$.

Algorithm for deriving the independence relation:

- For every predicate a, compute the context of sequents with atomic goals of head a.
- By examining the context, compute a set $S(a)$ of all predicates that a can depend on.
- For any $b \notin S(a)$, every predicate in $S(a)$ is independent of b. Generate a proof for this by mutual induction.
- Since $a \in S(a)$, a is independent of b.

Example: For our example, $S(\hat{\mathtt{bt}}) = \{\hat{\mathtt{bt}}\}$. Thus $\mathtt{bt}$ is independent of $\mathtt{nat}$.

Related Work: Subordination

Subordination is a popular notion for characterizing dependence in type theory:

Related Work: Subordination

Subordination is a popular notion for characterizing dependence in type theory:

- For every (sub)type $\tau_1 \to \cdots \to \tau_n \to A$, derive that τ_i is subordinate to A
- Subordination is closed under reflexivity and transitivity.

Related Work: Subordination

Subordination is a popular notion for characterizing dependence in type theory:

- For every (sub)type $\tau_1 \to \cdots \to \tau_n \to A$, derive that τ_i is subordinate to A
- Subordination is closed under reflexivity and transitivity.

Non-subordination is used to show the transportation of proofs.

Example: In Canonical LF, non-subordination is used to show the adequacy of encodings.

Related Work: Subordination

Subordination is a popular notion for characterizing dependence in type theory:

- For every (sub)type $\tau_1 \to \cdots \to \tau_n \to A$, derive that τ_i is subordinate to A
- Subordination is closed under reflexivity and transitivity.

Non-subordination is used to show the transportation of proofs.

Example: In Canonical LF, non-subordination is used to show the adequacy of encodings.

Problems with subordination:

Related Work: Subordination

Subordination is a popular notion for characterizing dependence in type theory:

- For every (sub)type $\tau_1 \to \cdots \to \tau_n \to A$, derive that τ_i is subordinate to A
- Subordination is closed under reflexivity and transitivity.

Non-subordination is used to show the transportation of proofs.

Example: In Canonical LF, non-subordination is used to show the adequacy of encodings.

Problems with subordination:

- It is built into the given type theory, thus completely trusted

Related Work: Subordination

Subordination is a popular notion for characterizing dependence in type theory:

- For every (sub)type $\tau_1 \to \cdots \to \tau_n \to A$, derive that τ_i is subordinate to A
- Subordination is closed under reflexivity and transitivity.

Non-subordination is used to show the transportation of proofs.

Example: In Canonical LF, non-subordination is used to show the adequacy of encodings.

Problems with subordination:

- It is built into the given type theory, thus completely trusted
- (Non-)subordination is an (under)over-approximation of the (in)dependence.

  Example: `nat` is subordinate to `bt` by the type of `leaf`.

Conclusion

Conclusion

- Developed a methodology for formalizing independence
  - Implementation in a framework based on proof theory
  - Use STLC as an example
- Developed an algorithm to derive and prove independence
  - Automatically generate the independence relation
  - Automatically derive the proof of independence

Conclusion

- Developed a methodology for formalizing independence
  - Implementation in a framework based on proof theory
  - Use STLC as an example
- Developed an algorithm to derive and prove independence
  - Automatically generate the independence relation
  - Automatically derive the proof of independence

Future Work:

- Using the methodology in other logical frameworks
- Extension to other type theories (*e.g.* LF).

Conclusion

- Developed a methodology for formalizing independence
  - Implementation in a framework based on proof theory
  - Use STLC as an example
- Developed an algorithm to derive and prove independence
  - Automatically generate the independence relation
  - Automatically derive the proof of independence

Future Work:

- Using the methodology in other logical frameworks
- Extension to other type theories (*e.g.* LF).

Examples in Abella:

http://abella-prover.org/independence

Conclusion

- Developed a methodology for formalizing independence
  - Implementation in a framework based on proof theory
  - Use STLC as an example
- Developed an algorithm to derive and prove independence
  - Automatically generate the independence relation
  - Automatically derive the proof of independence

Future Work:

- Using the methodology in other logical frameworks
- Extension to other type theories (*e.g.* LF).

Examples in Abella:

http://abella-prover.org/independence

Thank you!