A Proof-theoretic Characterization of Independence in Type Theory

Yuting Wang \(^1\) \quad Kaustuv Chaudhuri \(^2\)

\(^1\)University of Minnesota, Twin Cities, USA

\(^2\)Inria & LIX/École polytechnique, France

TLCA, July 2015, Warsaw
Formalizing transportation of theorems and proofs about type theories in different contexts.
Formalizing transportation of theorems and proofs about type theories in different contexts.

Example:

\[
\begin{align*}
  z &: \text{nat} \\
  s &: \text{nat} \to \text{nat} \\
  \text{leaf} &: (\text{nat} \to \text{bt}) \to \text{bt} \\
  \text{node} &: \text{bt} \to \text{bt} \to \text{bt}
\end{align*}
\]
Formalizing transportation of theorems and proofs about type theories in different contexts.

*Example:*

\[
\begin{align*}
  z & : \text{nat} & s & : \text{nat} \rightarrow \text{nat} \\
  \text{leaf} & : (\text{nat} \rightarrow \text{bt}) \rightarrow \text{bt} & \text{node} & : \text{bt} \rightarrow \text{bt} \rightarrow \text{bt}
\end{align*}
\]

Suppose given some property \( P \) about \( \text{bt} \) we prove

\[
\forall b : \text{bt}. P(b).
\]
Formalizing transportation of theorems and proofs about type theories in different contexts.

*Example*: 

\[
\begin{align*}
z : \text{nat} & \quad s : \text{nat} \to \text{nat} \\
\text{leaf} : (\text{nat} \to \text{bt}) & \to \text{bt} \\
\text{node} : \text{bt} & \to \text{bt} \to \text{bt}
\end{align*}
\]

Suppose given some property \( P \) about \( \text{bt} \) we prove 

\[
\forall b : \text{bt}. P(b).
\]

*Question*: After adding \( c : \text{nat} \) does the theorem still hold?
Formalizing transportation of theorems and proofs about type theories in different contexts.

Example:

\[ z : \text{nat} \quad s : \text{nat} \to \text{nat} \]
\[ \text{leaf} : (\text{nat} \to \text{bt}) \to \text{bt} \quad \text{node} : \text{bt} \to \text{bt} \to \text{bt} \]

Suppose given some property \( P \) about \( \text{bt} \) we prove

\[ \forall b : \text{bt}. P(b). \]

Question: After adding \( c : \text{nat} \) does the theorem still hold?

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

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

Definition (Independence)

The type $\tau_2$ is independent of $\tau_1$ in the context $\Gamma$ if whenever $\Gamma, x: \tau_1 \vdash t: \tau_2$ holds for some $t$, the $\beta$-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 and can be used to formalize transportation of theorems.

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

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

Definition (Independence)

The type $\tau_2$ is independent of $\tau_1$ in the context $\Gamma$ if whenever $\Gamma, x: \tau_1 \vdash t : \tau_2$ holds for some $t$, the $\beta$-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 $\tau_2$ is independent of $\tau_1$ in the context $\Gamma$ if whenever $\Gamma, x:\tau_1 \vdash t : \tau_2$ holds for some $t$, the $\beta$-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 $\tau_2$ is **independent** of $\tau_1$ in the context $\Gamma$ if whenever $\Gamma, x:\tau_1 \vdash t : \tau_2$ holds for some $t$, the $\beta$-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:* \texttt{bt} is independent of \texttt{nat} in the last example.
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 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 $\lambda$-calculus (STLC) as an example.
Our contributions:

- A methodology for formalizing proofs of independence
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 $G$
Contributions (Overview)

Our contributions:

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

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

- An algorithm for automatically checking independence
  - Derive the independence relation from the typing context
  - Simultaneously generate a proof of independence
Our contributions:

- A methodology for formalizing proofs of independence
  - Encoding the type theory in a specification logic called $\mathcal{HH}$
  - 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 $\lambda$-calculus (STLC) as an example.
We want to prove the following lemma by induction:

$$\forall t, \text{ if } \Gamma, x: \tau_1 \vdash t : \tau_2 \text{ is derivable then so is } \Gamma \vdash t : \tau_2.$$
Elaboration of Independence Proofs

We want to prove the following lemma by induction:

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

Considering the independence of \( \tau_2 \) to \( \tau_1 \) alone is not enough.
We want to prove the following lemma by induction:

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

Considering the independence of $\tau_2$ to $\tau_1$ alone is not enough. 

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

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

Need to prove the independence of $\tau$ to $\tau_1$ for the new type $\tau$. 

We want to prove the following lemma by induction:

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

Considering the independence of $\tau_2$ to $\tau_1$ alone is not enough. 

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

$$\begin{align*}
\Gamma, x: \tau_1 & \vdash t_1 : \tau \rightarrow \tau_2 \\
\Gamma, x: \tau_1 & \vdash t_2 : \tau \\
\Gamma, x: \tau_1 & \vdash t_1 \ t_2 : \tau_2
\end{align*}$$

Need to prove the independence of $\tau$ to $\tau_1$ for the new type $\tau$.

*Solution*:

- Since the context $\Gamma$ 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:

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

Considering the independence of \(\tau_2\) to \(\tau_1\) alone is not enough.  

*Example*: when \(t\) is an application \(t_1 \ t_2\):

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

Need to prove the independence of \(\tau\) to \(\tau_1\) for the new type \(\tau\).

**Solution:**

- Since the context \(\Gamma\) 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 is called the logic of higher-order hereditary Harrop formulas (HH):

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 \( \Gamma \vdash F \) (\( \Gamma = F_1, \ldots, F_n \))

\( \Gamma \) is called the context and \( F \) is called the goal.

A derivation alternates between the following two phases:

1. Simplify the goal until it becomes atomic;
2. Perform backchaining on the atomic goal.
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 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 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:

  \[ F ::= \forall \vec{x} : \vec{\tau}. F_1 \Rightarrow \cdots \Rightarrow F_n \Rightarrow A. \]

- A sequent calculus for derive sequents of the form

  \[ \Gamma \vdash F \quad (\Gamma = F_1, \ldots, F_n) \]

  \( \Gamma \) 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

\[ \Gamma \vdash F \quad (\Gamma = F_1, \ldots, F_n) \]

\(\Gamma\) 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.
The encoding is based on **types-as-predicates** principle:
The encoding is based on **types-as-predicates** principle:

- Atomic types and constants are imported into HH signature.
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 \( \hat{b} : b \rightarrow \circ \)
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 $J$ from STLC types $\tau$ to predicates $\tau \to \circ$:

$$
\begin{align*}
[b] & = \lambda t. \hat{b} t \quad \text{if } b \text{ is an atomic type.} \\
[\tau_1 \to \tau_2] & = \lambda t. \forall x : \tau_1. [\tau_1] x \Rightarrow [\tau_2] (t x)
\end{align*}
$$
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 $\hat{b} : b \rightarrow o$
- Define a mapping $\llbracket \cdot \rrbracket$ from STLC types $\tau$ to predicates $\tau \rightarrow o$:

\[
\begin{align*}
\llbracket b \rrbracket & = \lambda t. \hat{b} \ t \quad \text{if } b \text{ is an atomic type.} \\
\llbracket \tau_1 \rightarrow \tau_2 \rrbracket & = \lambda t. \forall x:\tau_1. \llbracket \tau_1 \rrbracket x \Rightarrow \llbracket \tau_2 \rrbracket (t \ x)
\end{align*}
\]

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

\[
\llbracket \Gamma \rrbracket \vdash \llbracket \tau \rrbracket \ t
\]

where $\llbracket \Gamma \rrbracket = \{ \llbracket \tau_1 \rrbracket \ x_1, \ldots, \llbracket \tau_n \rrbracket \ x_n \}$
Assume the following STLC signature $\Gamma$:

\[
z : \text{nat} \quad s : \text{nat} \rightarrow \text{nat}
\]

\[
\text{leaf} : (\text{nat} \rightarrow \text{bt}) \rightarrow \text{bt} \quad \text{node} : \text{bt} \rightarrow \text{bt} \rightarrow \text{bt}
\]
Assume the following STLC signature $\Gamma$:

\[
\begin{align*}
  z &: \text{nat} & s &: \text{nat} \to \text{nat} \\
  \text{leaf} &: (\text{nat} \to \text{bt}) \to \text{bt} & \text{node} &: \text{bt} \to \text{bt} \to \text{bt}
\end{align*}
\]

- Define two predicates $\text{nat} : \text{nat} \to \text{o}$ and $\text{bt} : \text{bt} \to \text{o}$.
Assume the following STLC signature $\Gamma$:

$$
\begin{align*}
z & : \text{nat} \quad s & : \text{nat} \to \text{nat} \\
\text{leaf} & : \text{nat} \to \text{bt} \to \text{bt} \quad \text{node} & : \text{bt} \to \text{bt} \to \text{bt}
\end{align*}
$$

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

$$
\begin{align*}
\text{nât} \; z. \quad & \forall x. \; \text{nât} \; x \Rightarrow \text{nât} \; (s \; x). \\
\forall x. \; (\forall y. \; \text{nât} \; y \Rightarrow \text{bât} \; (x \; y)) & \Rightarrow \text{bât} \; (\text{leaf} \; x). \\
\forall x \; y. \; \text{bât} \; x & \Rightarrow \text{bât} \; y \Rightarrow \text{bât} \; (\text{node} \; y \; x).
\end{align*}
$$
Example of Encoding

Assume the following STLC signature $\Gamma$:

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

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

- Define two predicates $\hat{\text{nat}} : \text{nat} \to \text{o}$ and $\hat{\text{bt}} : \text{bt} \to \text{o}$.
- Constants are encoded as the following clauses
  
  $$
  \hat{\text{nat}} z. \quad \forall x. \hat{\text{nat}} x \Rightarrow \hat{\text{nat}} (s \ x).
  $$
  $$
  \forall x. (\forall y. \hat{\text{nat}} y \Rightarrow \hat{\text{bt}} (x \ y)) \Rightarrow \hat{\text{bt}} (\text{leaf} x).
  $$
  $$
  \forall x \ y. \hat{\text{bt}} x \Rightarrow \hat{\text{bt}} y \Rightarrow \hat{\text{bt}} (\text{node} \ y \ x).
  $$

- Example of encoding typing judgments:
  
  $$
  \Gamma, x : \text{nat} \to \text{bt}, y : \text{bt} \vdash \text{node} (\text{leaf} x) \ y : \text{bt}
  $$
  
  is encoded as the following HH sequent:
  
  $$
  [\Gamma], (\forall y. \hat{\text{nat}} y \Rightarrow \hat{\text{bt}} (x \ y)), \hat{\text{bt}} y \vdash \hat{\text{bt}} (\text{node} (\text{leaf} x) \ y)
  $$
Now $\tau_2$ is independent of $\tau_1$ can be stated as follows:

\[ \text{If } [\Gamma], [\tau_1] x \vdash [\tau_2] t \text{ is derivable in } \mathcal{HH}, \text{ then so is } [\Gamma] \vdash [\tau_2] t. \]
Independence as Strengthening Lemmas

Now $\tau_2$ is independent of $\tau_1$ can be stated as follows:

*If $[\Gamma], [\tau_1] \vdash [\tau_2] t$ is derivable in $\mathbb{HH}$, then so is $[\Gamma] \vdash [\tau_2] t$."

It is an instance of strengthening lemmas.
Independence as Strengthening Lemmas

Now $\tau_2$ is independent of $\tau_1$ can be stated as follows:

\[ \text{If } [\Gamma], [\tau_1] \ x \vdash [\tau_2] \ t \text{ is derivable in } \mathbb{H} \mathbb{H}, \text{ then so is } [\Gamma] \vdash [\tau_2] \ t. \]

It is an instance of strengthening lemmas.

Example: $bt$ is independent of $nat$:

\[ \text{If } [\Gamma], \hat{nat} \ x \vdash \hat{bt} \ t \text{ is derivable, then so is } [\Gamma] \vdash \hat{bt} \ t, \]

where $\Gamma$ is the signature in the last example.
Independence as Strengthening Lemmas

Now $\tau_2$ is independent of $\tau_1$ can be stated as follows:

\[
\text{If } \llbracket \Gamma \rrbracket, \llbracket \tau_1 \rrbracket \vdash \llbracket \tau_2 \rrbracket \ t \text{ is derivable in } \mathbb{H}, \text{ then so is } \llbracket \Gamma \rrbracket \vdash \llbracket \tau_2 \rrbracket \ t.
\]

It is an instance of strengthening lemmas.

**Example:** $bt$ is independent of $nat$:

\[
\text{If } \llbracket \Gamma \rrbracket, \hat{nat} \ x \vdash \hat{bt} \ t \text{ is derivable, then so is } \llbracket \Gamma \rrbracket \vdash \hat{bt} \ t, \text{ where } \Gamma \text{ is the signature in the last example.}
\]

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

\[
\forall x. (\forall y. \hat{nat} \ y \Rightarrow \hat{bt} (x \ y)) \Rightarrow \hat{bt} (\text{leaf } x).
\]
Independence as Strengthening Lemmas

Now $\tau_2$ is independent of $\tau_1$ 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 $[\Gamma], nat x \vdash bt t$ is derivable, then so is $[\Gamma] \vdash bt t$, where $\Gamma$ is the signature in the last example.

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

$$\forall x. (\forall y. nat y \Rightarrow bt (x y)) \Rightarrow bt (leaf x).$$

We prove a generalized lemma:

If $(\llbracket \Gamma \rrbracket, \Delta, nat x \vdash bt t)$ is derivable, then so is $(\llbracket \Gamma \rrbracket, \Delta \vdash bt t)$, where $\Delta$ is the dynamic context.
A Two-level Logic Approach

$G$ 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:

\[
\text{add} \ z \ N \ N \ \ddagger \ \top; \ \text{add} (s \ N \ 1) \ N \ 2 (s \ N \ 3) \ \ddagger \ \text{add} \ N \ 1 \ N \ 2 \ N \ 3
\]

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

A new quantifier $\nabla$ for variables representing names.

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

A $\text{HH}$ sequent $L \vdash G$ is encoded as $\text{seq} \ L \ G$.

Derivation rules are encoded as definitions for $\text{seq}$.

We write $\{ L \vdash G \}$ for $\text{seq} \ L \ G$.
$\mathcal{G}$ is an intuitionistic logic based on Church’s STT.

- Atomic predicates are interpreted through fixed-point definitions

Example: the definition for addition of naturals is:

\[ \text{add } z \in \mathbb{N} \times \mathbb{N} \approx \top; \]
\[ \text{add } (s \mathbb{N}^1 \mathbb{N}^2) \mathbb{N}^3 \approx \text{add } \mathbb{N}^1 \times \mathbb{N}^2 \times \mathbb{N}^3 \]

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

A new quantifier $\nabla$ for variables representing names.

$\mathcal{H}$ is encoded as a fixed-point definition for the predicate $\text{seq}$.

An $\mathcal{H}$ sequent $L \vdash G$ is encoded as $\text{seq} L G$.

Derivation rules are encoded as definitions for $\text{seq}$.

We write $\{ L \vdash G \}$ for $\text{seq} L G$.
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:

\[
\text{add} \ z \ N \ N \triangleq \top; \quad \text{add} \ (s \ N_1) \ N_2 \ (s \ N_3) \triangleq \text{add} \ N_1 \ N_2 \ N_3
\]
\( \mathcal{G} \) 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:

\[
\begin{align*}
\text{add } z \; N \; N & \triangleq \top; \\
\text{add } (s \; N_1) \; N_2 \; (s \; N_3) & \triangleq \text{add } N_1 \; N_2 \; N_3
\end{align*}
\]

- We can also give them a least (greatest) fixed point reading, leading to support for (co)-inductive reasoning.
A Two-level Logic Approach

$G$ 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:

  $$\text{add } z \, N \, N \triangleq \top; \quad \text{add } (s \, N_1) \, N_2 \, (s \, N_3) \triangleq \text{add } N_1 \, N_2 \, N_3$$

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

- A new quantifier $\nabla$ 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:

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

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

- A new quantifier $\nabla$ for variables representing names.

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

\[
\text{add } z \ N \ N \triangleq \top; \quad \text{add } (s \ N_1) \ N_2 \ (s \ N_3) \triangleq \text{add } N_1 \ N_2 \ N_3
\]

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

- A new quantifier \( \nabla \) for variables representing names.

\( \mathbb{H} \mathbb{H} \) is encoded as a fixed-point definition for the predicate \( \text{seq} \)

- An \( \mathbb{H} \mathbb{H} \) sequent \( L \vdash G \) is encoded as \( \text{seq} \ L \ G \)

Yuting Wang, Kaustuv Chaudhuri

Characterization of Independence in Type Theory 10/14
A Two-level Logic Approach

$G$ 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:

  \[ \text{add } z \ N \ N \overset{\triangleq}{=} \top; \quad \text{add } (s \ N_1) \ N_2 \ (s \ N_3) \overset{\triangleq}{=} \text{add } N_1 \ N_2 \ N_3 \]

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

- A new quantifier $\nabla$ for variables representing names.

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

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

- Derivation rules are encoded as definitions for $\text{seq}$
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:
  \[ \text{add } z \ N \ N \triangleq \top; \quad \text{add } (s \ N_1) \ N_2 (s \ N_3) \triangleq \text{add } N_1 \ N_2 \ N_3 \]

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

- A new quantifier \(\nabla\) for variables representing names.

\(\mathbb{HH}\) is encoded as a fixed-point definition for the predicate \(\text{seq}\)

- An \(\mathbb{HH}\) sequent \(L \vdash G\) is encoded as \(\text{seq } L \ G\)
- Derivation rules are encoded as definitions for \(\text{seq}\)
- We write \(\{L \vdash G\}\) for \(\text{seq } L \ G\).
\(\tau_2\) is independent of \(\tau_1\) can be stated as follows in \(\mathcal{G}\):

\[
\forall t. \nabla x. \{[\Gamma], [\tau_1] x \vdash [\tau_2] (t \times)\} \supset \exists t'. t = (\lambda y. t') \land \{[[\Gamma]] \vdash [[\tau_2]] t'\}.
\]
$\tau_2$ is independent of $\tau_1$ can be stated as follows in $G$

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

- The possibility that $t$ may contain $x$ is expressed by $t\ x$
$\tau_2$ is independent of $\tau_1$ can be stated as follows in $\mathcal{G}$

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

- 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 $G$

$\tau_2$ is independent of $\tau_1$ can be stated as follows in $G$

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

- 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 $nat$

$$\forall t. \nabla x. \{ [[\Gamma]], \text{nat} x \vdash bt (t \ x) \} \cup \exists t'. t = (\lambda y. t') \land \{ [[\Gamma]] \vdash bt t' \}$$
Formalizing Independence in $\mathcal{G}$

$\tau_2$ is independent of $\tau_1$ can be stated as follows in $\mathcal{G}$

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

- 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 $nat$

$$\forall t. \nabla x. \{[[\Gamma]], n\hat{a}t \; x \vdash b\hat{t} \; (t \; x)) \supset \exists t'. t = (\lambda y. t') \land \{[[\Gamma]] \vdash b\hat{t} \; t'\}$$

We prove a generalized lemma:

$$\forall \Delta \; t. \nabla x. \text{ctx} \; \Delta \supset \{[[\Gamma]], \Delta, n\hat{a}t \; x \vdash b\hat{t} \; (t \; x)) \supset \exists t'. t = (\lambda y. t') \land \{[[\Gamma]], \Delta \vdash b\hat{t} \; t'\}$$

where $\text{ctx}$ defines the dynamically extended context.
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 $b \not\in S(a)$, 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, $S(\hat{bt}) = \{\hat{bt}\}$. Thus $bt$ is independent of $nat$. 

Yuting Wang, Kaustuv Chaudhuri
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:
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 \( a \).
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.
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.
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 \( b \).
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 \( b \not\in S(a) \), 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, \( S(\hat{b}t) = \{\hat{b}t\} \). Thus \( bt \) is independent of \( \text{nat} \).
Subordination is a popular notion for characterizing dependence in type theory:

For every (sub)type \( \tau_1 \rightarrow \cdots \rightarrow \tau_n \rightarrow A \), derive that \( \tau_i \) 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: \texttt{nat} is subordinate to \texttt{bt} by the type of \texttt{leaf}.
Subordination is a popular notion for characterizing dependence in type theory:

- For every (sub)type $\tau_1 \rightarrow \cdots \rightarrow \tau_n \rightarrow A$, derive that $\tau_i$ is subordinate to $A$
- Subordination is closed under reflexivity and transitivity.
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 \( \tau_i \) 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 $\tau_i$ 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:
Subordination is a popular notion for characterizing dependence in type theory:

- For every (sub)type \( \tau_1 \rightarrow \cdots \rightarrow \tau_n \rightarrow A \), derive that \( \tau_i \) 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
Subordination is a popular notion for characterizing dependence in type theory:

- For every (sub)type $\tau_1 \rightarrow \cdots \rightarrow \tau_n \rightarrow A$, derive that $\tau_i$ 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:* $\text{nat}$ is subordinate to $\text{bt}$ by the type of $\text{leaf}$. 
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!

Yuting Wang, Kaustuv Chaudhuri

Characterization of Independence in Type Theory
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](http://abella-prover.org/independence)

Thank you!

Yuting Wang, Kaustuv Chaudhuri
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!