Negations
in Refinement Type Systems

T. Tsukada (U. Tokyo)

14th March 2016
Shonan, JAPAN

This Talk

About refinement intersection type systems that refute judgements of other type systems.

$$\nvdash M : \tau$$

$$\iff \quad \Vdash M : \neg\tau$$

Background

Refinement intersection type systems are the basis for

- model checkers for higher-order model checking (cf. [Kobayashi 09] [Broadbent&Kobayashi 11] [Ramsay+ 14]),
- software model-checker for higher-order programs (cf. MoCHi [Kobayashi+ 11]).

In those type systems,

- a derivation gives a witness of derivability,
- but nothing witnesses that a given derivation is not derivable.

Motivation

A witness of underivability would be useful for

- a compact representation of an error trace
- an efficient model-checker in collaboration with the affirmative system
  - cf. [Ramsay+ 14] [Godefroid+ 10]
- development of a type system proving safety
  - In some cases (e.g. [T&Kobayashi 14]), a type system proving failure is easier to be developed.

Contribution

Development of type systems refuting derivability in some type systems such as

- a basic type system for the λ-calculus
- a type system for call-by-value reachability

Theoretical study of the development

Outline

- Negations in type systems for
  - the call-by-name $\lambda^{\rightarrow}$-calculus
    - Target language
    - Affirmative System
    - Negative System
  - the call-by-name $\lambda^{\rightarrow}$-calculus + recursion
- Semantic analysis
- Discussions

CbN $\lambda^{\to}$-calculus

A simply typed calculus equipped with $\beta\eta$-equivalence.

Kinds (i.e. simple types):

$$A, B ::= o \mid A \to A$$

Terms:

$$M, N ::= x \mid \lambda x^A.M \mid M\,M$$

CbN $\lambda^{\to}$-calculus

A simply typed calculus equipped with $\beta\eta$-equivalence.

Typing rules:

$$\frac{(x :: A) \in \Delta}{\Delta \vdash x :: A}$$

$$\frac{\Delta, x :: A \vdash M :: B}{\Delta \vdash \lambda x^A . M :: A \to B}$$

$$\frac{\Delta \vdash M :: A \to B \qquad \Delta \vdash N :: A}{\Delta \vdash M\,N :: B}$$

CbN $\lambda^{\to}$-calculus

A simply typed calculus equipped with $\beta\eta$-equivalence.

Equational theory:

$$(\lambda x.M)\,N = M[N/x]$$

$$\lambda x.M\,x = M \qquad \text{(if } x \notin \mathrm{fv}(M)\text{)}$$

Outline

- Negations in type systems for
  - the call-by-name $\lambda^{\rightarrow}$-calculus
    - Target language
    - Affirmative System
    - Negative System
  - the call-by-name $\lambda^{\rightarrow}$-calculus + recursion

- Semantic analysis

- Discussions

Affirmative system for CbN $\lambda^{\to}$

The type system for higher-order model checking (without the rule for recursion).

Types are parameterised by kinds and ground type sets:

$$\mathrm{Ty}_Q(o) := Q$$

$$\mathrm{Ty}_Q(A \to B) := \mathcal{P}(\mathrm{Ty}_Q(A)) \times \mathrm{Ty}_Q(B)$$

We use the following syntax for types:

$$\tau, \sigma ::= q \mid \bigwedge X \to \tau$$

$$X, Y \in \mathcal{P}(\mathrm{Ty}_Q(A))$$

Sets of Types via Refinement Relation

Let A be a kind.

The set $\mathrm{Ty}_Q(A)$ of types that refines A is given by

$$\mathrm{Ty}_Q(A) = \{\, \tau \mid \tau :: A \,\}$$

where is the refinement relation:

$$\frac{q \in Q}{q :: o} \qquad \frac{\forall \sigma \in X. \sigma :: A \qquad \tau :: B}{(\bigwedge X \to \tau) :: A \to B}$$

Subtyping

The subtyping relation is defined by induction on kinds.

$$\frac{}{q \preceq_o q}$$

$$\frac{X \succeq_{!A} Y \qquad \tau \preceq_B \sigma}{(\bigwedge X \to \tau) \preceq_{A \to B} (\bigwedge Y \to \sigma)}$$

$$\frac{\forall \sigma \in Y. \exists \tau \in X. \tau \preceq_A \sigma}{X \preceq_{!A} Y}$$

Type Environments

A (finite) map from variables to sets of types
(or intersection types).

$$\Gamma ::= x_1 : X_1, \ldots, x_n : X_n \quad (n \geq 0)$$

Typing rules

$$\frac{(x : X) \in \Gamma \qquad \tau \in X \qquad \tau \preceq \sigma}{\Gamma \vdash x : \sigma}$$

$$\frac{\Gamma, x : X \vdash M : \tau}{\Gamma \vdash \lambda x. M : \bigwedge X \to \tau}$$

$$\frac{\Gamma \vdash M : \bigwedge X \to \tau \qquad \Gamma \vdash N : \bigwedge X}{\Gamma \vdash M\, N : \tau}$$

$$\frac{\forall \tau \in X.\, \Gamma \vdash M : \tau}{\Gamma \vdash M : \bigwedge X}$$

Fact: Invariance under $\beta\eta$-equivalence

Suppose that $M =_{\beta\eta} N$. Then

$$\Gamma \vdash M : \tau \Leftrightarrow \Gamma \vdash N : \tau$$

- This fact will not be used in the sequel.

Convention: Subtyping closure

In what follows, sets of types are assumed to be closed under the subtyping relation.

$$\tau \succeq \sigma \in X \Rightarrow \tau \in X$$

Now posets of types are simply defined by:

$$\mathrm{Ty}_Q(o) := (Q, =)$$
$$\mathrm{Ty}_Q(A \to B) := u(\mathrm{Ty}_Q(A))^{op} \times \mathrm{Ty}_Q(B)$$

where $u(P, \leq) := (\{X \subseteq P \mid x \geq y \in X \Rightarrow x \in X\}, \supseteq)$

(cf. $X \subseteq Y$ implies $\bigwedge X \succcurlyeq \bigwedge Y$)

Convention: Subtyping closure

In what follows, sets of types are assumed to be closed under the subtyping relation.

$$\tau \succeq \sigma \in X \Rightarrow \tau \in X$$

The rule for variables becomes simpler.

$$\frac{(x : X) \in \Gamma \qquad \tau \in X}{\Gamma \vdash x : \tau}$$

Outline

- Negations in type systems for
  - the call-by-name $\lambda^{\rightarrow}$-calculus
    - Target language
    - Affirmative System
    - Negative System
  - the call-by-name $\lambda^{\rightarrow}$-calculus + recursion

- Semantic analysis

- Discussions

Negative Type System

Negative types are those constructed from the negative ground types $\overline{Q} := \{ \bar{q} \mid q \in Q \}$:

$$\overline{\mathrm{Ty}_Q(A)} := \mathrm{Ty}_{\overline{Q}}(A)$$

$$\bar{\tau}, \bar{\sigma} ::= \bar{q} \mid \bigwedge \bar{X} \to \bar{\tau}$$
$$\bar{X}, \bar{Y} \in u(\mathrm{Ty}_{\bar{Q}}(A))$$

Typing rules are the same as the affirmative system.

Negation of a type

We define the two anti-monotone bijections on types

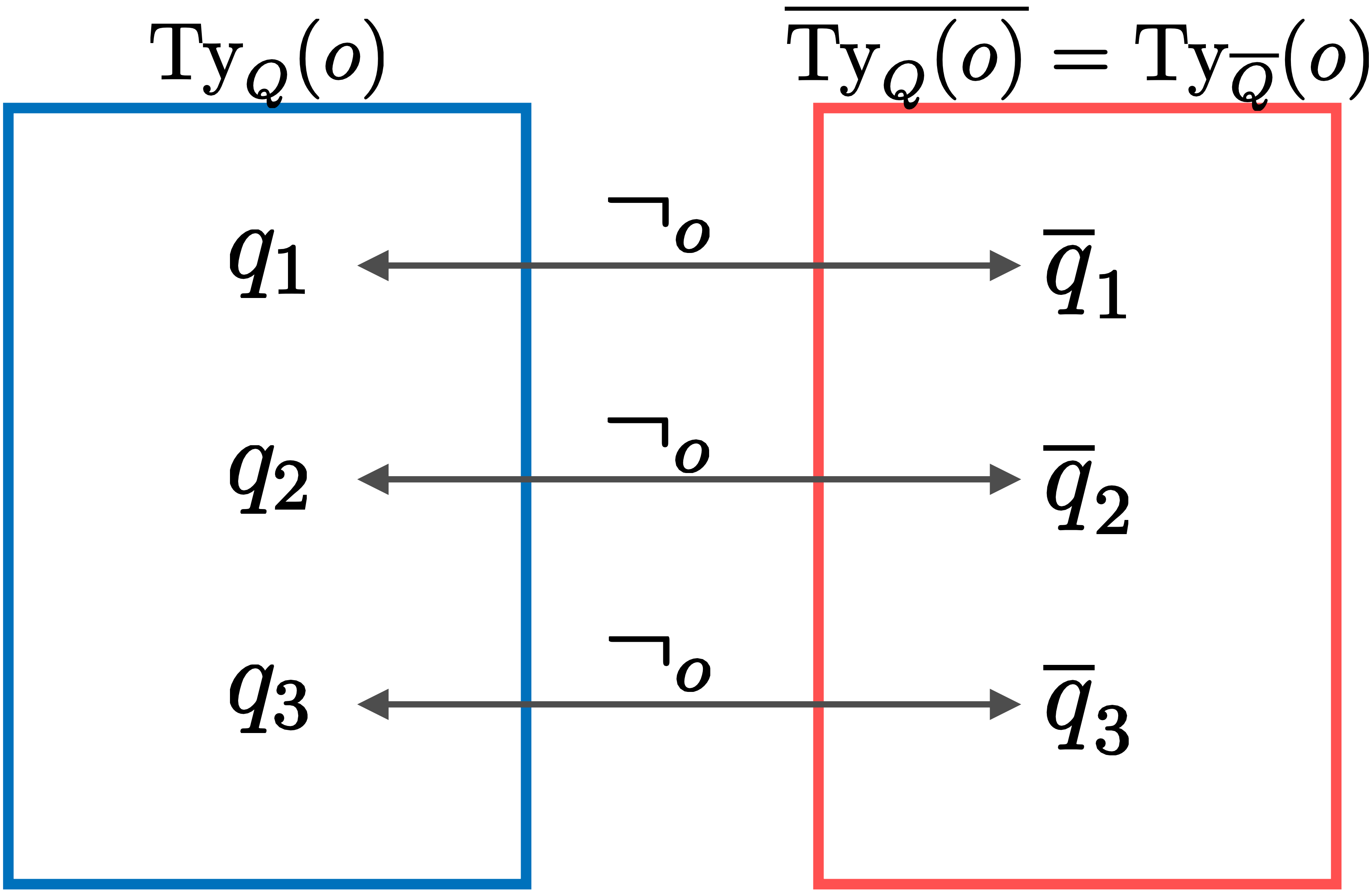
$$\neg_A : \mathrm{Ty}_Q(A) \longrightarrow \overline{\mathrm{Ty}_Q(A)}$$
$$\natural_A : u(\mathrm{Ty}_Q(A)) \longrightarrow u(\overline{\mathrm{Ty}_Q(A)})$$

as follows:

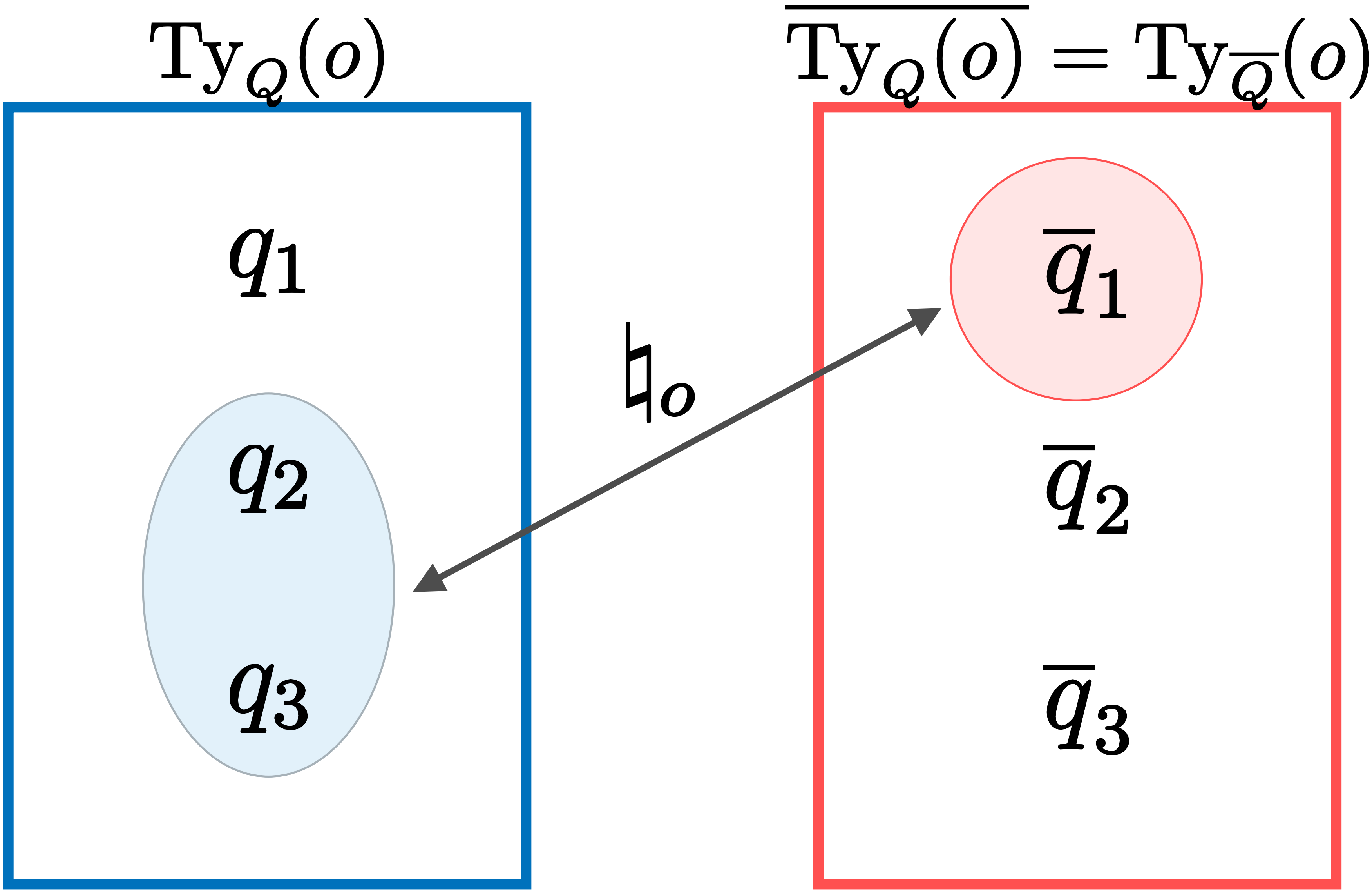
$$\neg_o q := \bar{q}$$
$$\neg_{A\to B}(\bigwedge X \to \tau) := \bigwedge(\natural_A X) \to (\neg_B \tau)$$

$$\natural_A X := \{\, \neg_A \tau \mid \tau \notin X \,\}$$

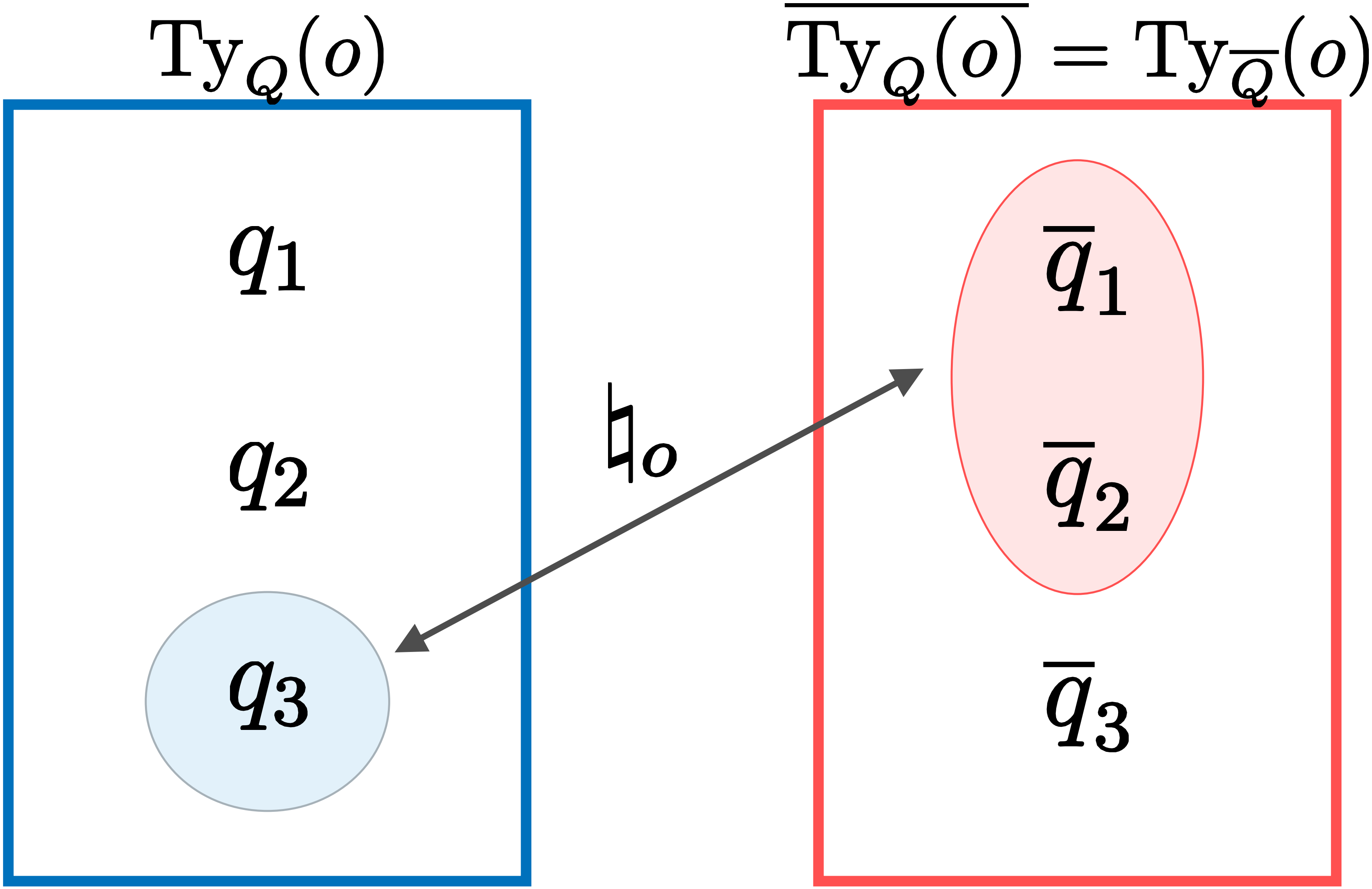
Negation $\neg_A : \mathrm{Ty}_Q(A) \longrightarrow \overline{\mathrm{Ty}_Q(A)}$



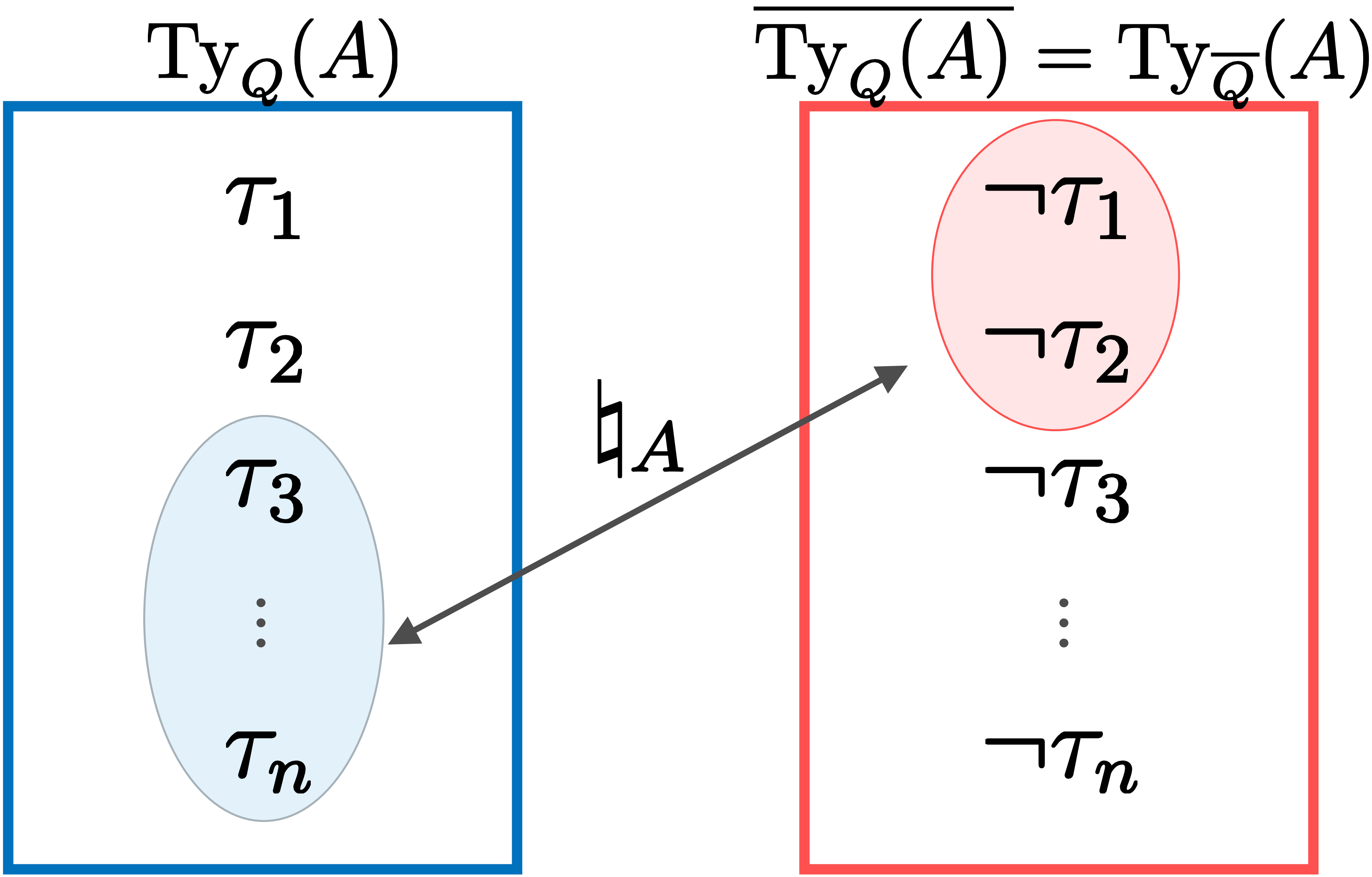
Natural $\natural_A : u(\mathrm{Ty}_Q(A)) \longrightarrow u(\overline{\mathrm{Ty}_Q(A)})$



Natural $\quad \natural_A : u(\mathrm{Ty}_Q(A)) \longrightarrow u(\overline{\mathrm{Ty}_Q(A)})$



Natural $\quad \natural_A : u(\mathrm{Ty}_Q(A)) \longrightarrow u(\overline{\mathrm{Ty}_Q(A)})$



Negation of a type

We define the two anti-monotone bijections on types

$$\neg_A : \mathrm{Ty}_Q(A) \longrightarrow \overline{\mathrm{Ty}_Q(A)}$$
$$\natural_A : u(\mathrm{Ty}_Q(A)) \longrightarrow u(\overline{\mathrm{Ty}_Q(A)})$$

as follows:

$$\neg_o q := \bar{q}$$
$$\neg_{A\to B}(\bigwedge X \to \tau) := \bigwedge(\natural_A X) \to (\neg_B \tau)$$

$$\natural_A X := \{\, \neg_A \tau \mid \tau \notin X \,\}$$

Negation of a type

We

$$x : \bigwedge X \vdash x : \neg\tau \;\Leftrightarrow\; x : \bigwedge X \nvdash x : \tau \;\Leftrightarrow\; \tau \notin X$$

$$\Leftrightarrow\; \neg\tau \in \natural X \;\Leftrightarrow\; x : \bigwedge(\natural X) \vdash x : \neg\tau$$

$$M : \neg(\bigwedge X \to \tau) \text{ iff } x : \bigwedge X \vdash M\,x : \neg\tau$$

$$\text{iff } x : \bigwedge(\natural X) \vdash M\,x : \neg\tau$$

as f

$$\neg_{A\to B}(\bigwedge X \to \tau) := \bigwedge(\natural_A X) \to (\neg_B \tau)$$

$$\natural_A X := \{\, \neg_A \tau \mid \tau \notin X \,\}$$

Main Theorem

Theorem

- $\Gamma \nvdash M : \tau$ if and only if $\natural\Gamma \vdash M : \neg\tau$,
  where $\natural(x_1 : X_1, \ldots, x_n : X_n) := x_1 : (\natural X_1), \ldots, x_n : (\natural X_n)$
- Let $X = \{\, \tau \mid \Gamma \vdash M : \tau \,\}$. Then
$$\natural\Gamma \vdash M : \bigwedge(\natural X)$$

Proof) By mutual induction on the structure of the term.

Main Theorem

Theorem

- $\Gamma \nvdash M : \tau$ if and only if $\natural\Gamma \vdash M : \neg\tau$,
  where $\natural(x_1 : X_1, \ldots, x_n : X_n) := x_1 : (\natural X_1), \ldots, x_n : (\natural X_n)$

- Let $X = \{\, \tau \mid \Gamma \vdash M : \tau \,\}$. Then

$$\natural\Gamma \vdash M : \bigwedge(\natural X)$$

$$\Gamma \vdash M : \bigwedge X \quad \text{iff} \quad \natural\Gamma \vdash M : \bigwedge(\natural X)$$

under a certain condition

Pr … m.

Outline

- Negations in type systems for
  - the call-by-name $\lambda^{\rightarrow}$-calculus
  - the call-by-name $\lambda^{\rightarrow}$-calculus + recursion

- Semantic analysis

- Discussions

$\lambda^{\rightarrow}$ + Recursion

Term:

$$M, N ::= x \mid \lambda x^A.M \mid M\,M \mid Y\,M$$

Equational theory:

$$\begin{aligned}
(\lambda x.M)\,N &= M[N/x] \\
\lambda x.M\,x &= M \qquad\qquad\qquad (\text{if } x \notin \mathrm{fv}(M)) \\
Y\,M &= M\,(Y\,M)
\end{aligned}$$

Recursion Rule in Affirmative System

The rule for recursion is given by:

$$\frac{\Gamma \vdash M : \bigwedge X \to \tau \qquad \Gamma \vdash Y\,M : \bigwedge X}{\Gamma \vdash Y\,M : \tau}$$

This is a co-inductive rule: a derivation can be infinite.

Recursion Rule in Negative System

The rule for recursion is given by:

$$\frac{\Gamma \Vdash M : \bigwedge X \to \tau \qquad \Gamma \Vdash Y\, M : \bigwedge X}{\Gamma \Vdash Y\, M : \tau}$$

This is a inductive rule: a derivation must be finite.

Main Theorem

Lemma

$$\nvdash \lambda f.Y\ f : \tau \quad \iff \quad \Vdash \lambda f.Y\ f : \neg\tau$$

Theorem

- $\Gamma \nvdash M : \tau$ if and only if $\natural\Gamma \Vdash M : \neg\tau$.
- Let $X = \{\tau \mid \Gamma \vdash M : \tau\}$. Then

$$\natural\Gamma \Vdash M : \bigwedge(\natural X)$$

Outline

- Negations in type systems for
  - the call-by-name $\lambda^{\rightarrow}$-calculus
  - the call-by-name $\lambda^{\rightarrow}$-calculus + recursion

- Semantic analysis

- Discussions

Category $\mathbf{ScottL}_u$

Definition The category $\mathbf{ScottL}_u$ is given by:

Object — Poset $(A, \leq_A)$.

Morphism — An upward-closed relation

$$R \subseteq u(A)^{op} \times B$$

Composition — Let $\begin{matrix} R \subseteq u(A)^{op} \times B \\ S \subseteq u(B)^{op} \times C \end{matrix}$. Then

$$\frac{\exists Y \in u(B).\Big(\forall b \in Y.(X, b) \in R \text{ and } (Y, c) \in S\Big)}{(X, c) \in (S \circ R)}$$

Interpretation of CbN $\lambda^{\rightarrow}$ in $\mathbf{ScottL}_u$

Fact $\mathbf{ScottL}_u$ is a cartesian closed category.

Interpretation of kinds is given by:

$$\llbracket o \rrbracket_Q := (Q, =)$$
$$\llbracket A \to B \rrbracket_Q := u(\llbracket A \rrbracket_Q)^{op} \times \llbracket B \rrbracket_Q$$

Hence $\llbracket A \rrbracket_Q \cong \mathrm{Ty}_Q(A)$.

Fact (see e.g. [Terui 2012])

$$\Gamma \vdash M : \tau \quad \Leftrightarrow \quad (\Gamma, \tau) \in \llbracket M \rrbracket$$

Negation Functor on $\mathbf{ScottL}_u$

The functor $\varphi : \mathbf{ScottL}_u \to \mathbf{ScottL}_u$ is defined by:

$$\varphi(A) := A^{op}$$

$$\varphi(R) := \{\, (A \backslash X, b) \in u(A)^{op} \times B \mid (X, b) \notin R \,\}$$

Lemma φ is an isomorphism on $\mathbf{ScottL}_u$.

If $\mathrm{R} \in u(A)^{op} \times B$ and $A = \emptyset$, then

$$\varphi(R) = \{\, (\emptyset, b) \mid (\emptyset, b) \notin R \,\}$$

which is essentially the complement of R.

Outline

- Negations in type systems for
  - the call-by-name $\lambda^{\rightarrow}$-calculus
  - the call-by-name $\lambda^{\rightarrow}$-calculus + recursion
  - a call-by-value language + nondeterminism
- Semantic analysis
- Discussions

Automata complementation

Corresponds to negation of a 2nd-order judgement.

Boolean Closedness of Types

Let A be a kind and B_A be the set of all Böhm trees of type A. A language is a subset of B_A.

Definition A language $L \subseteq B_A$ is type-definable if there exists a type τ such that

$$L = \{\, M \in B_A \mid\ \vdash M : \tau \,\}$$

in the type system for higher-order model checking

[Kobayashi&Ong 09] [T&Ong 14].

Corollary The class of type-definable languages are closed under Boolean operations on sets.

Further Applications

The technique presented in this talk is applicable to:

- the type system for the full higher-order model-checking [Kobayashi&Ong 09]
- a type system witnessing call-by-value reachability [T&Kobayashi 14]
- a dependent intersection type system in [Kobayashi+ 11], via the translation of dependent types to intersection and union types

Consistency and Inconsistency

The negation of a "small" type can be very large. So the negation may not be efficiently computable.

The notion of consistency and inconsistency may be useful in the practical use:

Definition Let $\tau \in \mathrm{Ty}_Q(A)$ and $\bar{\sigma} \in \overline{\mathrm{Ty}_Q(A)}$. They are consistent if $\neg\tau \preceq \bar{\sigma}$ and inconsistent otherwise.

Proposition If τ and $\bar{\sigma}$ are inconsistent, then

$$\Vdash M : \bar{\sigma} \quad \Longrightarrow \quad \nvdash M : \tau$$

Inductive Definition of Consistency

$$\frac{q \neq p}{q \lhd_o \bar{p}}$$

$$\frac{\forall \tau \in X. \forall \bar{\sigma} \in \bar{Y}.\ \tau \lhd_A \bar{\sigma}}{\bigwedge X \lhd_{!A} \bigwedge \bar{Y}}$$

$$\frac{\tau_1 \lhd_{!A} \bar{\sigma}_1 \quad \Rightarrow \quad \tau_2 \lhd_B \bar{\sigma}_2}{(\tau_1 \to \tau_2) \lhd_{A \to B} (\bar{\sigma}_1 \to \bar{\sigma}_2)}$$

Inductive definition of inconsistency is now trivial.

Related Work

"Krivine machines and higher-order schemes"

[Salvati&Walkiewicz 12]

- The notion of consistency and inconsistency can be found in their work (called complementarity for the former and the latter has no name).
- This talk is partially inspired by their work.

Conclusion

Negation is a definable operation in the refinement intersection type system for the call-by-name $\lambda^{\to}$.

This observation leads to the construction of negative type systems for other refinement type systems, e.g.,

- call-by-name $\lambda^{\to}$ + recursion
- the type system for HOMC
- a type system for a call-by-value language

Application to verification needs some work.