Negations
in Refinement Type Systems

T. Tsukada (U. Tokyo)

18th Nov, 2015

Oxford, UK

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
  - a call-by-value language + nondeterminism
- 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:

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

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

Outline

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

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

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
  - a call-by-value language + nondeterminism
- 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

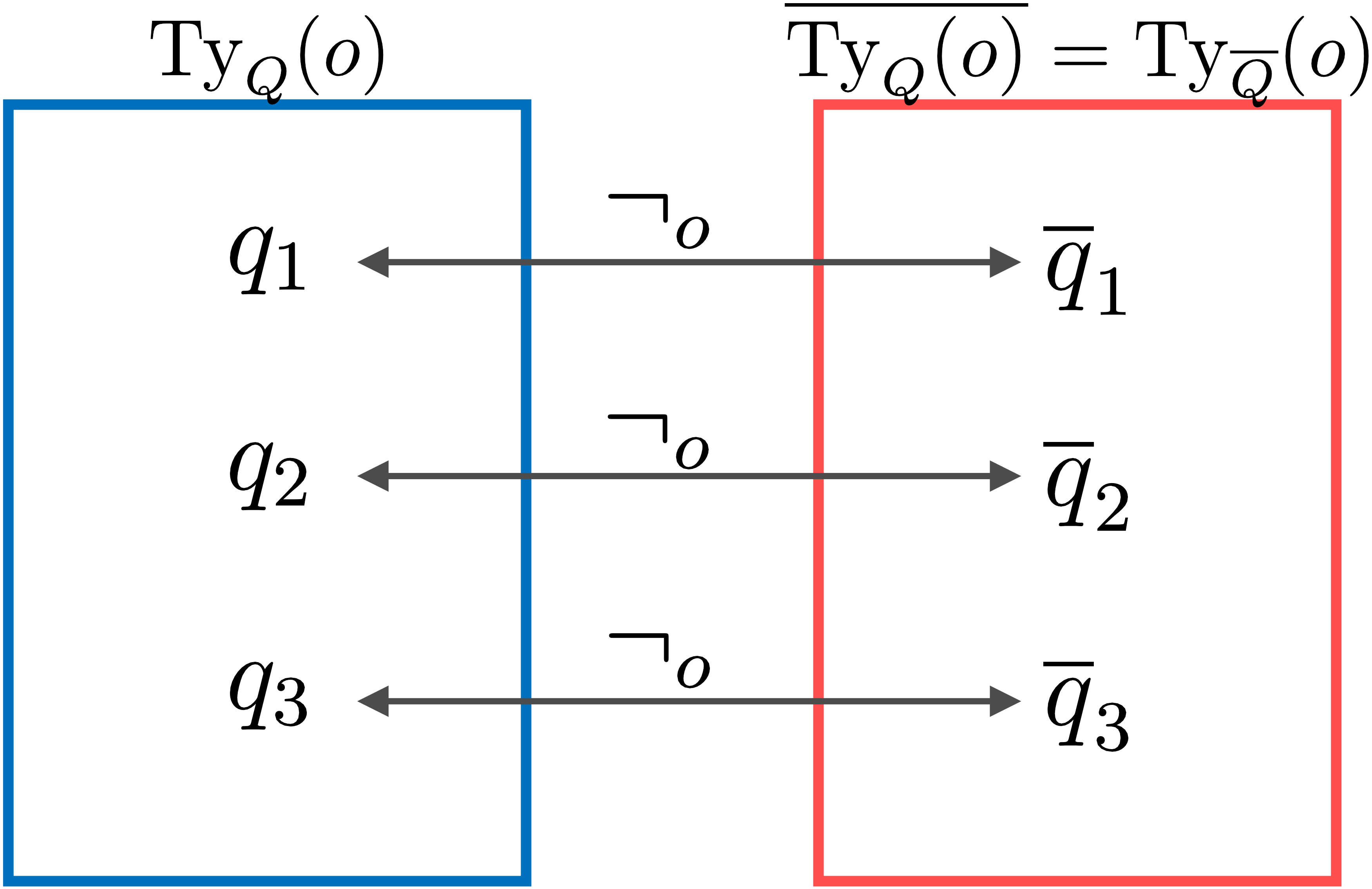
$$\neg_A : \mathrm{Ty}_Q(A) \longrightarrow \overline{\mathrm{Ty}_Q(A)}$$
$$\natural_A : u(\mathrm{Ty}_Q(A)) \longrightarrow u(\overline{\mathrm{Ty}_Q(A)})$$

as follows:

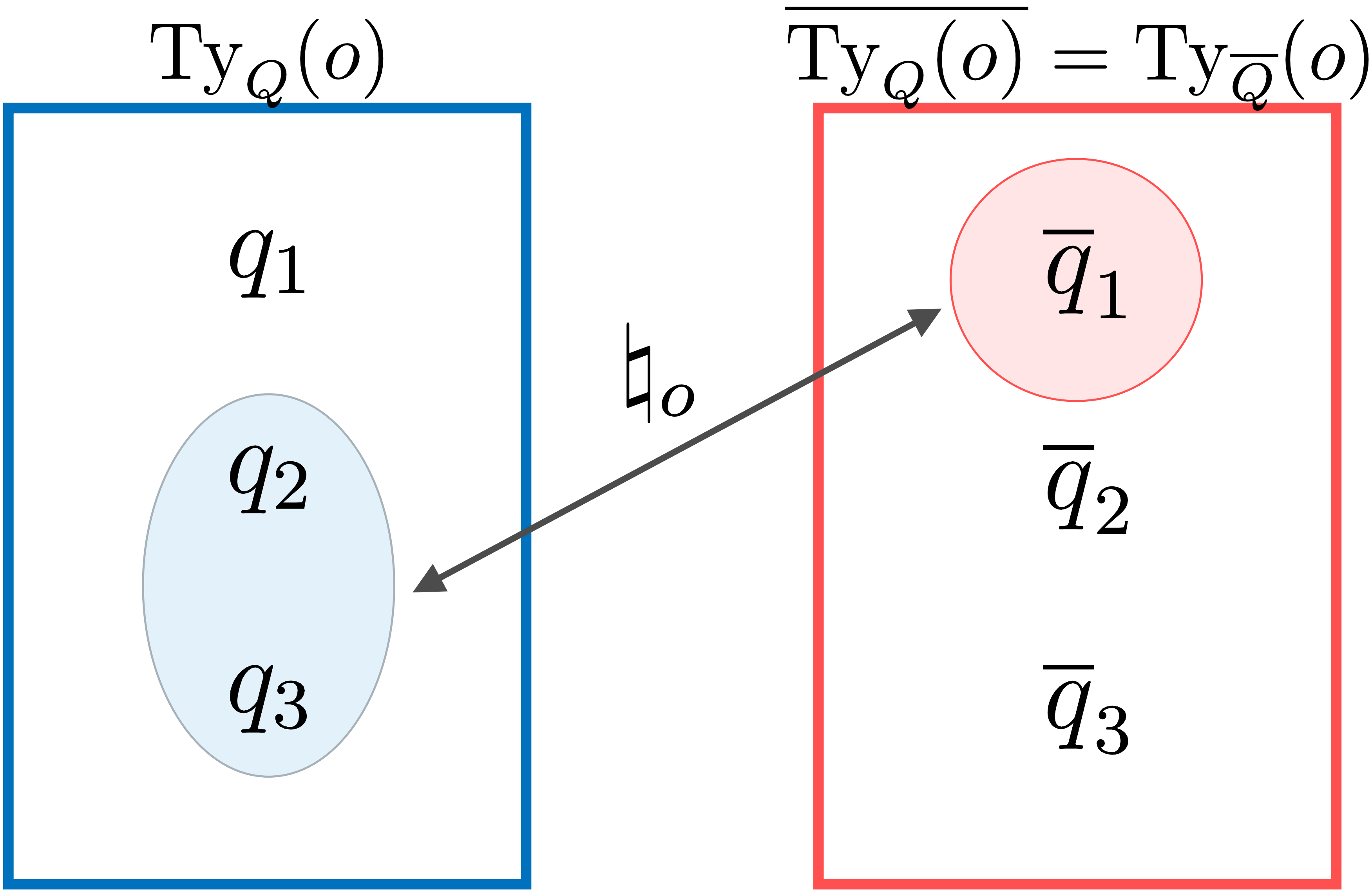
$$\neg_o q := \bar{q}$$
$$\neg_{A\to B}(\bigwedge X \to \tau) := \bigwedge(\natural_A X) \to (\neg_B \tau)$$

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

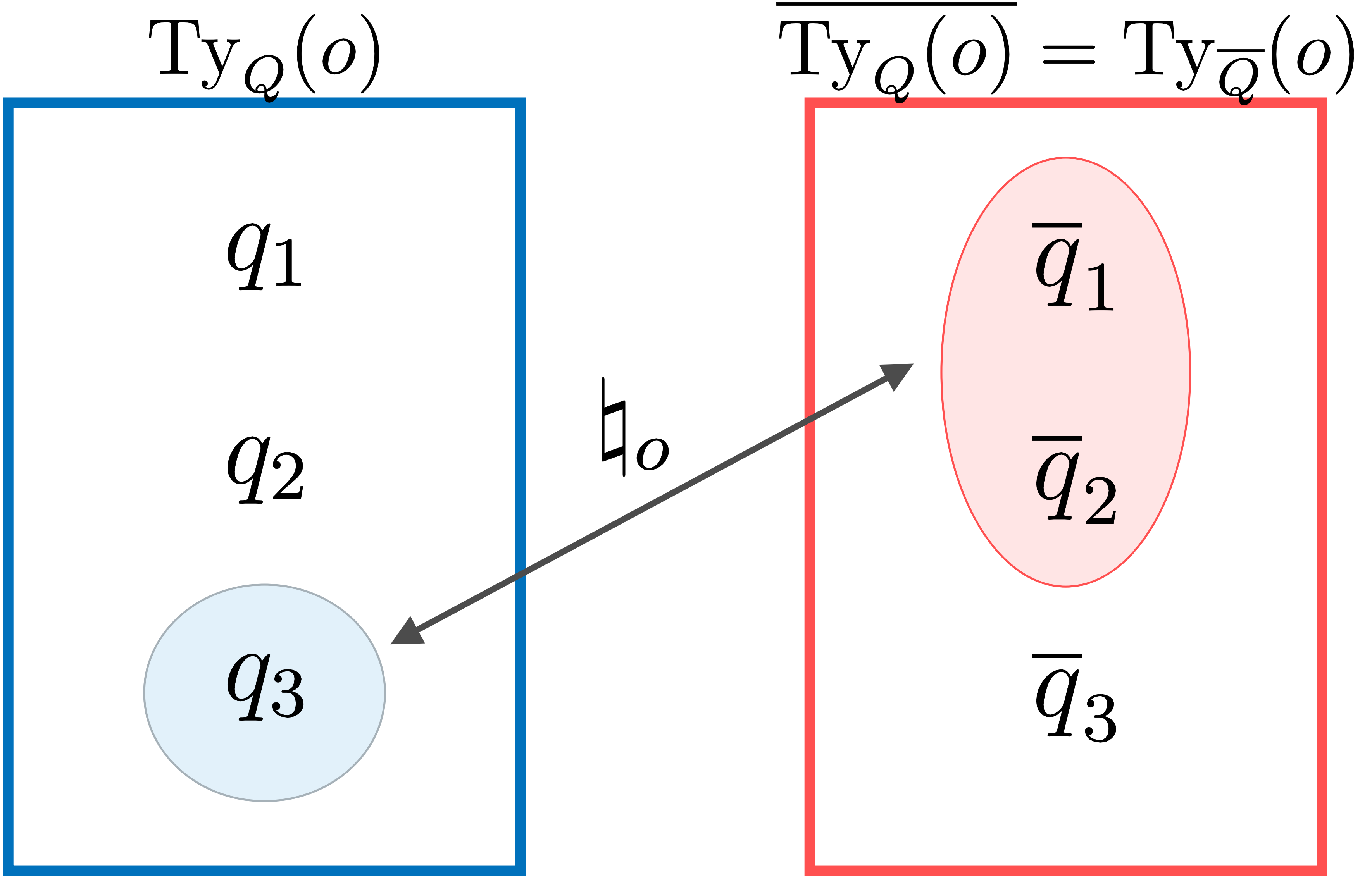
Negation $\neg_A : \mathrm{Ty}_Q(A) \longrightarrow \overline{\mathrm{Ty}_Q(A)}$



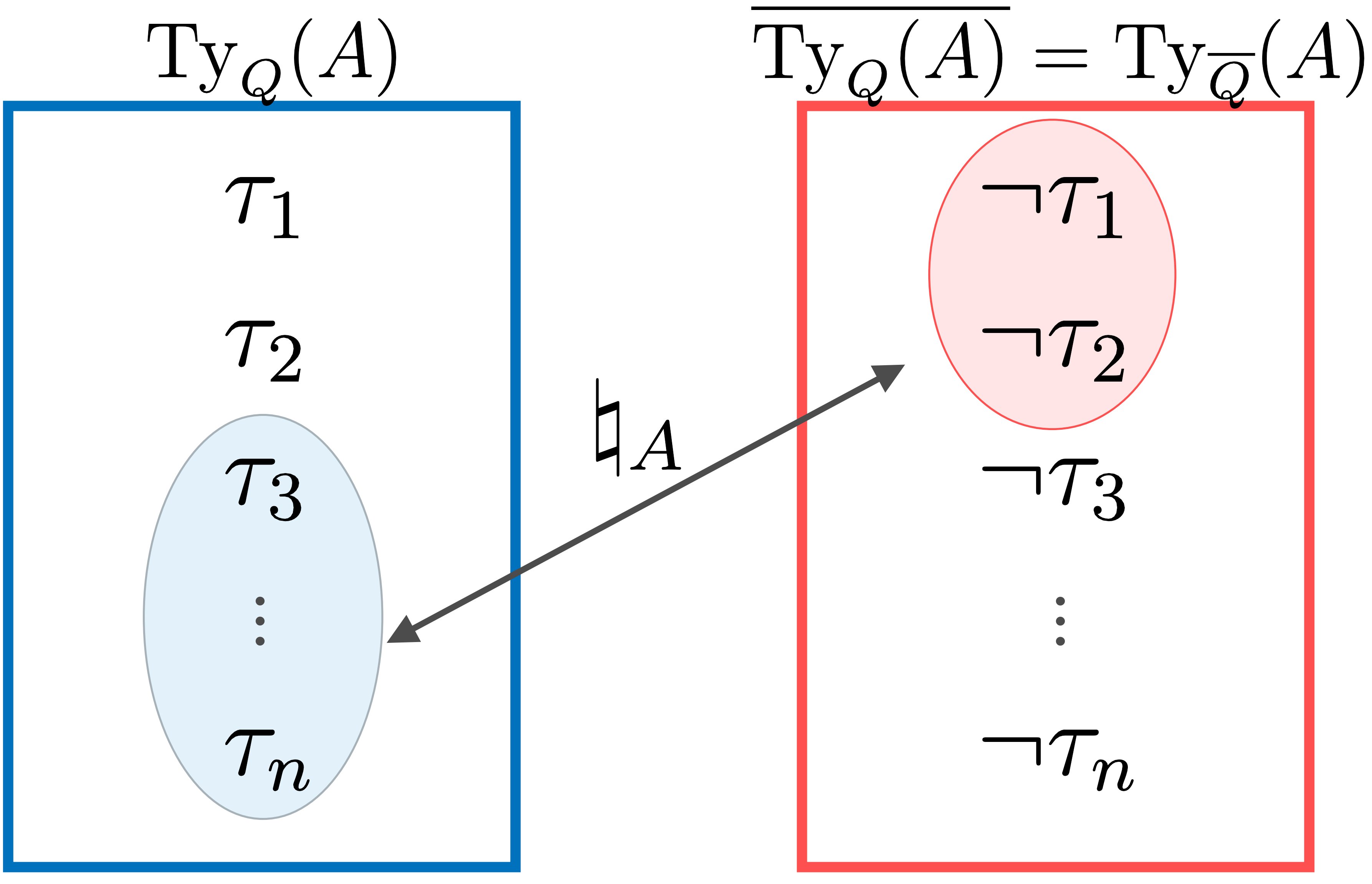
Natural $\natural_A : u(\mathrm{Ty}_Q(A)) \longrightarrow u(\overline{\mathrm{Ty}_Q(A)})$



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

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

as f

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

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

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

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
  - a call-by-value language + nondeterminism
- Semantic analysis
- Discussions

$\lambda^{\to}$ + Recursion

Term:

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

Equational theory:

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

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
  - a call-by-value language + nondeterminism
- Semantic analysis
- Discussions

Target Language

Kinds (or simple types):

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

Terms:

$$M ::= v \mid v\,v \mid \texttt{let}\ x = M\ \texttt{in}\ M \mid M \oplus M$$
$$\mid \mathit{if}\ v\ \texttt{then}\ M\ \texttt{else}\ M$$

$$v ::= \texttt{t} \mid \texttt{f} \mid x \mid \lambda x.M$$

Simple Type System

Value judgements ($\Delta \vdash M : A$):

$$\frac{(x :: A) \in \Delta}{\Delta \vdash x :: A} \qquad \frac{v \in \{\,\mathtt{t}, \mathtt{f}\,\}}{\Delta \vdash v :: o} \qquad \frac{\Delta, x :: A \vdash M :: TB}{\Delta \vdash M :: A \to TB}$$

Computation judgements ($\Delta \vdash M : TA$):

$$\frac{\Delta \vdash v :: A}{\Delta \vdash v :: TA} \qquad \frac{\Delta \vdash v_1 :: A \to TB \qquad \Delta \vdash v_2 :: A}{\Delta \vdash v_1\, v_2 :: TB}$$

$$\frac{\Delta \vdash M_i :: TA\ (i = 1, 2)}{\Delta \vdash M_1 \oplus M_2 :: TA} \qquad \frac{\Delta \vdash M :: TA \qquad \Delta, x :: A \vdash N :: TB}{\Delta \vdash \mathtt{let}\ x = M\ \mathtt{in}\ N :: TB}$$

$$\frac{\Delta \vdash v :: o \qquad \Delta \vdash M_i :: TA\ (i = 1, 2)}{\Delta \vdash \mathtt{if}\ v\ \mathtt{then}\ M_1\ \mathtt{else}\ M_2 :: TA}$$

Reduction Semantics

Base cases:

$$(\lambda x.M)\, v \longrightarrow M[v/x]$$

$$\texttt{let}\ x = v\ \texttt{in}\ M \longrightarrow M[v/x]$$

$$\texttt{if}\ \texttt{t}\ \texttt{then}\ M_1\ \texttt{else}\ M_2 \longrightarrow M_1$$

$$\texttt{if}\ \texttt{f}\ \texttt{then}\ M_1\ \texttt{else}\ M_2 \longrightarrow M_2$$

$$M_1 \oplus M_2 \longrightarrow M_1$$

$$M_1 \oplus M_2 \longrightarrow M_2$$

Evaluation context: $E ::= [] \mid \texttt{let}\ x = E\ \texttt{in}\ M$

Affirmative System

Types (formally defined by induction on types):

$$\tau, \sigma ::= \mathtt{t} \mid \mathtt{f} \mid \bigwedge X \mid \tau \to \tau$$

$$X, Y \in (\text{sets of types})$$

Refinement relation:

$$\frac{\tau \in \{\, \mathtt{t}, \mathtt{f} \,\}}{\tau :: o} \qquad \frac{\forall \tau \in X. \tau :: A}{\bigwedge X :: TA}$$

$$\frac{\forall \sigma \in X. \sigma :: A \qquad \tau :: TB}{(\bigwedge X \to \tau) :: A \to TB}$$

Example of a type

Let

$$\tau \;=\; \bigwedge \left\{ \begin{array}{c} \bigwedge\{\mathtt{t}\} \to \bigwedge\{\mathtt{f}\} \\ \bigwedge\{\mathtt{f}\} \to \bigwedge\{\mathtt{t}\} \end{array} \right\} \to \bigwedge\{\mathtt{t}\}$$

Then $\tau :: (o \to To) \to To$.

<u>Examples of derivable/underivable judgements</u>

$$\vdash \lambda f.(\mathtt{f} \;\;\oplus\;\; \mathtt{let}\; x = f\;\mathtt{t}\;\mathtt{in}\; f\; y) : \tau$$

$$\nvdash \lambda f.(\mathtt{f} \;\;\oplus\;\; f\;\mathtt{t}) : \tau$$

Typing Rules

Value judgements ($\Gamma \vdash M : \tau$ with $\tau :: A$):

$$\frac{(x : X) \in \Gamma \quad \tau \in X}{\Gamma \vdash x : \tau} \qquad \frac{}{\Gamma \vdash \mathtt{t} : \mathtt{t}} \qquad \frac{}{\Gamma \vdash \mathtt{f} : \mathtt{f}} \qquad \frac{\Gamma, x : X \vdash M : \tau}{\Gamma \vdash \lambda x.M : \bigwedge X \to \tau}$$

Computation judgements ($\Gamma \vdash M : \tau$ with $\tau :: TA$):

$$\frac{\forall \tau \in X. \Gamma \vdash v : \tau}{\Gamma \vdash v : \bigwedge X} \qquad \frac{\Gamma \vdash v_1 : \bigwedge X \to \tau \qquad \Gamma \vdash v_2 : \bigwedge X}{\Gamma \vdash v_1\, v_2 : \tau}$$

$$\frac{\exists i \in \{1, 2\}. \Gamma \vdash M_i : \tau}{\Gamma \vdash M_1 \oplus M_2 : \tau} \qquad \frac{\Gamma \vdash M : \bigwedge X \qquad \Gamma, x : X \vdash N : \tau}{\Gamma \vdash \mathtt{let}\ x = M\ \mathtt{in}\ N : \tau}$$

$$\frac{\Gamma \vdash v : \mathtt{t} \qquad \Gamma \vdash M_1 : \tau}{\Gamma \vdash \mathtt{if}\ v\ \mathtt{then}\ M_1\ \mathtt{else}\ M_2 : \tau} \qquad \frac{\Gamma \vdash v : \mathtt{f} \qquad \Gamma \vdash M_2 : \tau}{\Gamma \vdash \mathtt{if}\ v\ \mathtt{then}\ M_1\ \mathtt{else}\ M_2 : \tau}$$

Soundness and Completeness

Theorem

$$\vdash M : \bigwedge X \quad \Leftrightarrow \quad \exists v \in \langle M \rangle.\ \vdash v : \bigwedge X$$

(where $\langle M \rangle := \{\, v \mid M \to^* v \,\}$)

In particular,

$$\vdash M : \mathtt{t} \quad \Leftrightarrow \quad M \longrightarrow^* \mathtt{t}$$

$$\vdash M : \mathtt{f} \quad \Leftrightarrow \quad M \longrightarrow^* \mathtt{f}$$

Negative System

Types (formally defined by induction on types):

$$\bar{\tau}, \bar{\sigma} ::= \bar{\mathtt{t}} \mid \bar{\mathtt{f}} \mid \bigwedge \bar{X} \mid \bar{\tau} \to \bar{\tau} \mid \bigvee \bar{X}$$

$$\bar{X}, \bar{Y} \in (\text{sets of types})$$

Refinement relation:

$$\frac{\bar{\tau} \in \{\bar{\mathtt{t}}, \bar{\mathtt{f}}\}}{\bar{\tau} :: o} \qquad \frac{\forall \bar{\tau} \in \bar{X}. \bar{\tau} :: A}{\bigvee \bar{X} :: TA}$$

$$\frac{\forall \bar{\sigma} \in \bar{X}. \bar{\sigma} :: A \qquad \bar{\tau} :: TB}{(\bigwedge \bar{X} \to \bar{\tau}) :: A \to TB}$$

Typing Rules

Value judgements ($\Gamma \vdash M\!:\tau$ with $\tau :: A$):

$$\frac{(x : \bar{X}) \in \bar{\Gamma} \quad \bar{\tau} \in \bar{X}}{\bar{\Gamma} \Vdash x : \bar{\tau}} \qquad \frac{}{\bar{\Gamma} \Vdash \mathtt{t} : \bar{\mathtt{f}}} \qquad \frac{}{\bar{\Gamma} \Vdash \mathtt{f} : \bar{\mathtt{t}}} \qquad \frac{\bar{\Gamma}, x : \bar{X} \Vdash M : \bar{\tau}}{\bar{\Gamma} \Vdash \lambda x.M : \bigwedge \bar{X} \to \bar{\tau}}$$

Computation judgements ($\Gamma \vdash M\!:\tau$ with $\tau :: TA$):

$$\frac{\forall \bar{\tau} \in \bar{X}.\bar{\Gamma} \Vdash v : \bar{\tau}}{\bar{\Gamma} \Vdash v : \bigwedge \bar{X}} \qquad \frac{\exists \bar{\tau} \in \bar{X}.\bar{\Gamma} \Vdash v : \bar{\tau}}{\bar{\Gamma} \Vdash v : \bigvee \bar{X}}$$

$$\frac{\forall i \in \{1, 2\}.\bar{\Gamma} \Vdash M_i : \bar{\tau}}{\bar{\Gamma} \Vdash M_1 \oplus M_2 : \bar{\tau}}$$

$$\frac{\bar{\Gamma} \Vdash v_1 : \bigwedge \bar{X} \to \bar{\tau} \qquad \bar{\Gamma} \Vdash v_2 : \bigwedge \bar{X}}{\bar{\Gamma} \Vdash v_1\, v_2 : \bar{\tau}}$$

Typing rules (cont.)

Rules for conditional branch:

$$\frac{\bar{\Gamma} \Vdash v : \bar{\mathtt{f}} \qquad \bar{\Gamma} \Vdash M_1 : \bar{\tau}}{\bar{\Gamma} \Vdash \mathtt{if}\ v\ \mathtt{then}\ M_1\ \mathtt{else}\ M_2 : \bar{\tau}}$$

$$\frac{\bar{\Gamma} \Vdash v : \bar{\mathtt{t}} \qquad \bar{\Gamma} \Vdash M_2 : \bar{\tau}}{\bar{\Gamma} \Vdash \mathtt{if}\ v\ \mathtt{then}\ M_1\ \mathtt{else}\ M_2 : \bar{\tau}}$$

$$\frac{\bar{\Gamma} \Vdash v : \bar{\mathtt{t}} \wedge \bar{\mathtt{f}}}{\bar{\Gamma} \Vdash \mathtt{if}\ v\ \mathtt{then}\ M_1\ \mathtt{else}\ M_2 : \bar{\tau}}$$

$$\frac{\bar{\Gamma} \Vdash M_1 : \bar{\tau} \qquad \bar{\Gamma} \Vdash M_2 : \bar{\tau}}{\bar{\Gamma} \Vdash \mathtt{if}\ v\ \mathtt{then}\ M_1\ \mathtt{else}\ M_2 : \bar{\tau}}$$

Typing rules

Rule for let-expression:

$$\forall i \in I. \quad \bar{\Gamma} \Vdash M : \bigvee_{j \in J_i} \bar{\tau}_{i,j}$$

$$\bigwedge_{i \in I} \bigvee_{j \in J_i} \bar{\tau}_{i,j} \xmapsto{\text{dist. law}} \bigvee_{k \in K} \bigwedge_{l \in L_k} \bar{\sigma}_{k,l}$$

$$\frac{\forall k \in K. \quad \bar{\Gamma}, x : \{\, \bar{\sigma}_{k,l} \mid l \in L_k \,\} \Vdash N : \bar{\gamma}}{\bar{\Gamma} \Vdash \mathtt{let}\ x = M\ \mathtt{in}\ N : \bar{\gamma}}$$

Soundness and Completeness

Theorem

$$\Vdash M : \bigvee X \quad \Leftrightarrow \quad \forall v \in \langle M \rangle.\ \Vdash v : \bigvee X$$

(where $\langle M \rangle \coloneqq \{\, v \mid M \to^* v \,\}$)

In particular,

$$\Vdash M : \bar{\mathtt{t}} \quad \Leftrightarrow \quad M \not\to^* \mathtt{t}$$

$$\Vdash M : \bar{\mathtt{f}} \quad \Leftrightarrow \quad M \not\to^* \mathtt{f}$$

Negation of a type

Given a kind A, let

$$\mathrm{Ty}(A) := \{\, \tau \mid \tau :: A \,\}$$
$$\overline{\mathrm{Ty}(A)} := \{\, \bar{\tau} \mid \bar{\tau} :: U \,\}$$

We define two operations:

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

Definition of the Negation

$$\neg_o v := \bar{v} \qquad (v \in \{\, \mathtt{t}, \mathtt{f} \,\})$$

$$\neg_{A \to TB}\left(\bigwedge X \to \tau\right) := \bigwedge(\natural_A X) \to (\neg_{TB}\tau)$$

$$\neg_{TA}\left(\bigwedge X\right) := \bigvee\{\, \neg_A \tau \mid \tau \in X \,\}$$

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

Examples

$$\neg \mathtt{t} = \bar{t}$$

$$\neg \mathtt{f} = \bar{f}$$

$$\natural(\bigwedge\{\,\}) = \bar{t} \wedge \bar{f} \qquad \neg(\bigwedge\{\,\}) = \bigvee\{\,\}$$

$$\natural(\bigwedge\{\,\mathtt{t}\,\}) = \bigwedge\{\,\bar{\mathtt{f}}\,\} \qquad \neg(\bigwedge\{\,\mathtt{t}\,\}) = \bigvee\{\,\bar{\mathtt{t}}\,\}$$

$$\natural(\bigwedge\{\,\mathtt{f}\,\}) = \bigwedge\{\,\bar{\mathtt{t}}\,\} \qquad \neg(\bigwedge\{\,\mathtt{f}\,\}) = \bigvee\{\,\bar{\mathtt{f}}\,\}$$

$$\natural(\bigwedge\{\,\mathtt{t}, \mathtt{f}\,\}) = \bigwedge\{\,\} \qquad \neg(\bigwedge\{\,\mathtt{t}, \mathtt{f}\,\}) = \bigvee\{\,\bar{\mathtt{t}}, \bar{\mathtt{f}}\,\}$$

Examples

$$
\begin{aligned}
\neg\left(\bigwedge\{\texttt{t}\} \to \bigwedge\{\texttt{t}\}\right) &= \natural\left(\bigwedge\{\texttt{t}\}\right) \to \neg\left(\bigwedge\{\texttt{t}\}\right) \\
&= \bigwedge\{\bar{\texttt{f}}\} \to \bigvee\{\bar{\texttt{t}}\}
\end{aligned}
$$

$$
\neg\left(\bigwedge\{\,\} \to \bigwedge\{\,\}\right) = \bigwedge\{\,\bar{\texttt{t}}, \bar{\texttt{f}}\,\} \to \bigvee\{\,\}
$$

$$
\neg\left(\bigwedge\{\,\texttt{t}, \texttt{f}\,\} \to \bigwedge\{\,\texttt{t}, \texttt{f}\,\}\right) = \bigwedge\{\,\} \to \bigvee\{\,\bar{\texttt{t}}, \bar{\texttt{f}}\,\}
$$

Examples

Let

$$\tau = \bigwedge \left\{ \begin{array}{l} \bigwedge\{\mathtt{t}\} \to \bigwedge\{\mathtt{f}\} \\ \bigwedge\{\mathtt{f}\} \to \bigwedge\{\mathtt{t}\} \end{array} \right\} \to \bigwedge\{\mathtt{t}\}$$

Then

$$\neg\tau = \bigwedge \left\{ \begin{array}{l} \bigwedge\{\bar{\mathtt{f}}\} \to \bigvee\{\bar{\mathtt{t}}\} \\ \bigwedge\{\bar{\mathtt{t}}\} \to \bigvee\{\bar{\mathtt{f}}\} \\ \bigwedge\{\bar{\mathtt{t}}, \bar{\mathtt{f}}\} \to \bigvee\{\ \} \\ \bigwedge\{\ \} \to \bigvee\{\bar{\mathtt{t}}, \bar{\mathtt{f}}\} \end{array} \right\} \to \bigvee\{\bar{\mathtt{t}}\}$$

$$\nVdash \lambda f.(\mathtt{f} \quad \oplus \quad f\ \mathtt{t}) : \tau$$

$$\Vdash \lambda f.(\mathtt{f} \quad \oplus \quad f\ \mathtt{t}) : \neg\tau$$

Main Theorem

Theorem

- $\Gamma \nvdash M : \tau$ if and only if $\natural\Gamma \Vdash M : \neg\tau$.

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

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

Some (Possible) Extensions

1. CbV calculus with integers
    - Straightforward.
    - One needs infinite intersection and union.

2. CbV calculus with recursion
    - I believe that it is straightforward, though I have not yet checked.

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

How to Develop the Negative Systems

1. The CbN system has a categorical description.

$$\Gamma \vdash M : \tau \quad \Leftrightarrow \quad (\Gamma, \tau) \in [\![M]\!]_{\mathbf{ScottL}_u}$$

2. The negation induces an automorphism.

$$\varphi : \mathbf{ScottL}_u \xrightarrow{\cong} \mathbf{ScottL}_u$$

3. A CbV system is given by a monad on $\mathbf{ScottL}_u$.

4. The negative system is given by the monad

$$\mathbf{ScottL}_u \xrightarrow{\varphi^{-1}} \mathbf{ScottL}_u \xrightarrow{T} \mathbf{ScottL}_u \xrightarrow{\varphi} \mathbf{ScottL}_u$$

Category $\mathbf{ScottL}_u$

Definition The category **ScottL** 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:

$$[\![o]\!]_Q := (Q, =)$$

$$[\![A \to B]\!]_Q := u([\![A]\!]_Q)^{op} \times [\![B]\!]_Q$$

Hence $[\![A]\!]_Q \cong \mathrm{Ty}_Q(A)$.

Fact $\Gamma \vdash M : \tau \quad \Leftrightarrow \quad (\Gamma, \tau) \in [\![M]\!]$

Negation Functor on $\mathbf{ScottL}_u$

The functor $\varphi\colon \mathbf{ScottL}_u \to \mathbf{ScottL}_u$ 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.

Monad and Call-by-Value

A monad on a category C is a functor $C \to C$ with some additional structures.

A (strong) monad on a CCC gives rise to a model of a call-by-value calculus [Moggi 91].

A monad on $\mathbf{ScottL}_u$ can be seen as a refinement type system for a call-by-value calculus.

Negated Monad and Negative System

Let $T\colon \mathbf{ScottL}_u \to \mathbf{ScottL}_u$ be a strong monad. Then

$$\mathbf{ScottL}_u \xrightarrow{\varphi^{-1}} \mathbf{ScottL}_u \xrightarrow{T} \mathbf{ScottL}_u \xrightarrow{\varphi} \mathbf{ScottL}_u$$

has the canonical monad structure. Furthermore the respective Kliesli categories are isomorphic

$$(\mathbf{ScottL}_u)_T \quad \cong \quad (\mathbf{ScottL}_u)_{\varphi T \varphi^{-1}}$$

and the refinement type system corresponding to the right-hand-side is the negation of the left-hand-side.

Example

The previous type system for CbV calculus is given by the following monad.

$$
\begin{aligned}
T(A) &:= u(A) \\
T(R) &:= \{\, (\Xi, Y) \in u(u(A))^{op} \times u(B) \\
&\qquad\qquad\qquad \mid \exists X \in \Xi. \forall b \in Y. (X, b) \in R \,\}
\end{aligned}
$$

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 $\lambda^{\rightarrow}$.

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

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

Application to verification needs some work.