Relatively Complete Refinement Type System for Verification of Higher-Order Non-deterministic Programs

Hiroshi Unno (University of Tsukuba)
Yuki Satake (University of Tsukuba)
Tachio Terauchi (Waseda University)

Background

- Recent advances in (semi-)automated methods for verifying higher-order functional programs
  - safety [Rondon+ '08; U. & Kobayashi '08,'09; Terauchi '10; Ong & Ramsay '11; Jhala+ '11; Kobayashi+ '11; U.+ '13; …]
  - termination [Sereni & Jones '05; Giesl+ '11; Kuwahara+ '14; Vazou+ '14]
  - non-termination [Kuwahara+ '15; Hashimoto & U. '15]
  - temporal properties [Koskinen & Terauchi '14; Murase+ '16]
- Different techniques are used to verify the different classes of properties, and are hard to combine in a unified framework
  - dependent refinement types,
  - predicate abstraction for higher-order model checking,
  - program transformation for (binary) reachability analysis,…

Our Contributions

- Novel dependent refinement type system that can:
  - uniformly express and verify universal and existential branching properties of call-by-value, higher-order, and non-deterministic programs:
    - (cond.) safety, non-safety, termination, and non-termination
  - seamlessly combine universal and existential reasoning
    - e.g., Prove non-safety via termination
    - e.g., Prove non-termination via safety
    - e.g., Prove termination and non-termination simultaneously
- Meta-theoretic properties of the type system:
  - Closure of types under complement
  - Soundness
  - Relative completeness

Our Contributions

- Novel dependent refinement type system that can:
  - uniformly express and verify universal and existential branching properties of call-by-value, higher-order, and non-deterministic programs:
    - (cond.) safety, non-safety, termination, and non-termination
  - seamlessly combine universal and existential reasoning
    - e.g., Prove non-safety via termination
    - e.g., Prove non-termination via safety
    - e.g., Prove termination and non-termination simultaneously
- Meta-theoretic properties of the type system:
  - Closure of types under complement
  - Soundness
  - Relative completeness

Dependent Refinement Types τ

- $\{x : \mathbf{int} \mid x \geq 0\}$
  Non-negative integers

predicates on program values

- $(x : \mathbf{int}) \rightarrow \{r : \mathbf{int} \mid r \geq x\}$
  Functions that take an integer x and (if terminated) return r not less than x

☺ A type system ensures that a type-checked expression behaves according to the type
☹ Only universal branching properties can be expressed

Overview: Our Type System

- Extends dependent refinement types with:
  - Qualified types $\tau^{Q_1 Q_2}$
    to express universal/existential branching behaviors and partial/total correctness
  - Qualified bindings $x{:}^{Q}\tau$
    to cope with non-determinism from program inputs
  - Gödel encoding of function-type values & guarded intersection types
    to achieve relative completeness

Overview: Our Type System

- **Extends dependent refinement types with:**
  - **Qualified types $\tau^{Q_1 Q_2}$**
    to express universal/existential branching behaviors and partial/total correctness
  - **Qualified bindings $x :^{Q} \tau$**
    to cope with non-determinism from program inputs
  - **Gödel encoding of function-type values & guarded intersection types**
    to achieve relative completeness

Qualified Types $\tau^{Q_1 Q_2}$

- $Q_1 \in \{\forall, \exists\}$ (universal/existential non-det.) specifies whether the expression being typed behaves according to the type:
  - for any non-det. evaluation ($Q_1 = \forall$), or
  - for some non-det. evaluation ($Q_1 = \exists$)
- $Q_2 \in \{\forall, \exists\}$ (partial/total correctness) specifies whether:
  - τ holds for all value obtained ($Q_2 = \forall$), or
  - there exists a final value for which τ holds ($Q_2 = \exists$)

Qualified Types $\tau^{Q_1 Q_2}$

- $Q_1 \in \{\forall, \exists\}$ (universal/existential non-det.) specifies whether the expression being typed behaves according to the type:
  - for any non-det. evaluation ($Q_1 = \forall$), or
  - for some non-det. evaluation ($Q_1 = \exists$)
- $Q_2 \in \{\forall, \exists\}$ (partial/total correctness) specifies whether:
  - the evaluation diverges or τ is satisfied ($Q_2 = \forall$), or
  - the evaluation terminates and τ is satisfied ($Q_2 = \exists$)

Examples: Qualified Types $\tau^{Q_1 Q_2}$

- $\vdash e : \{u : \mathrm{int} \mid u > 0\}^{\forall\forall}$
  for any non-deterministic evaluation of e,
  if any integer u is obtained, then u is positive

- $\vdash e : \{u : \mathrm{int} \mid u > 0\}^{\exists\exists}$
  for some non-deterministic evaluation of e,
  some integer u is obtained, and u is positive

Typing Integer Constants

Rule:

Integer constant

$$\Delta \vdash n : \{x : \mathbf{int} \mid x = n\}^{\forall\exists}$$

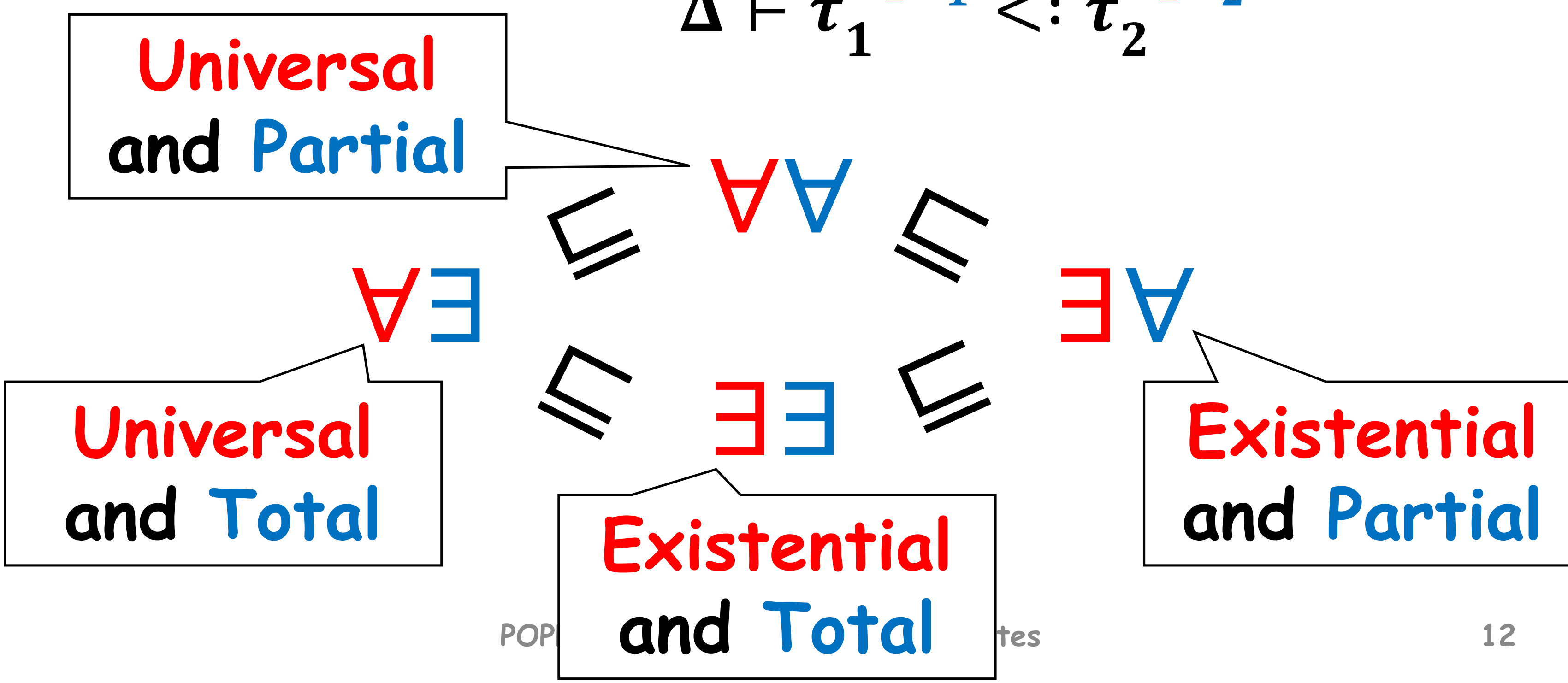
Type environment

Universal and Total

Converting Qualified Types

Subtyping Rule:

$$\frac{\Delta \vdash \tau_1 <: \tau_2 \qquad {\color{red}Q_1}{\color{blue}Q_1'} \sqsubseteq {\color{red}Q_2}{\color{blue}Q_2'}}{\Delta \vdash \tau_1^{{\color{red}Q_1}{\color{blue}Q_1'}} <: \tau_2^{{\color{red}Q_2}{\color{blue}Q_2'}}}$$



Typing Let-Bindings

Rule:

$$\frac{\Delta \vdash e_1 : \tau_1^{Q_1 Q_2} \qquad \Delta, x : \tau_1 \vdash e_2 : \tau_2^{Q_1 Q_2} \qquad x \notin fvs(\tau_2)}{\Delta \vdash \mathbf{let}\ x = e_1\ \mathbf{in}\ e_2 : \tau_2^{Q_1 Q_2}}$$

Typing Recursive Functions for Partial Correctness

Rule:

$$\frac{\Delta, x : \tau_1, f : (x : \tau_1) \rightarrow \tau_2^{Q\forall} \vdash e : \tau_2^{Q\forall}}{\Delta \vdash \mathbf{rec}(f, x, e) : (x : \tau_1) \rightarrow \tau_2^{Q\forall}}$$

(recursive) function
$\mathrm{let\,rec}\ f\ x = e$

Typing Recursive Functions for Total Correctness (cf. [Xi '01])

Well-founded relation witnessing the termination of f, as a recursion guard

Rule:

$$\tau_{rec} = (x' : \tau_1{}') \to \phi \rhd (\tau_2{}')^{Q\exists} =_\alpha (x : \tau_1) \to \tau_2^{Q\exists}$$

$$\frac{\Delta \models WF(\lambda(x, x').\phi) \quad \Delta, x : \tau_1, f : \tau_{rec} \vdash e : \tau_2^{Q\exists}}{\Delta \vdash \mathrm{rec}(f, x, e) : (x : \tau_1) \to \tau_2^{Q\exists}}$$

Example:

$(x' : \{x' \mid x' \geq 0\}) \to x > x' \geq 0 \rhd \{y' \mid y' \geq x'\}^{\forall\exists}$

$$\frac{x : \{x \mid x \geq 0\}, sum : \tau_{rec} \vdash \text{if } x = 0 \text{ then } 0 \text{ else} \ldots : \mathrm{int}^{\forall\exists} \qquad \models WF(\lambda(x, x').x > x' \geq 0)}{\vdash \mathrm{rec}\big(sum, x, \text{if } x = 0 \text{ then } 0 \text{ else } x + sum\ (x - 1)\big) : \tau}$$

$(x : \{x \mid x \geq 0\}) \to \mathrm{int}^{\forall\exists}$

Overview: Our Type System

- **Extends dependent refinement types with:**
  - Qualified types $\tau^{Q_1 Q_2}$
    to express universal/existential branching behaviors
    and partial/total correctness
  - Qualified bindings $x :^{Q} \tau$
    to cope with non-determinism from program inputs
  - Gödel encoding of function-type values
    & guarded intersection types
    to achieve relative completeness

Qualified Bindings $x{:}^{Q}\tau$

- Occur in type environments and the argument of dependent function types
- $Q \in \{\forall, \exists\}$ specifies whether a certain fact must hold for:
  - any input x that satisfies τ ($Q = \forall$), or
  - some input x that satisfies τ ($Q = \exists$)

Examples: Qualified Bindings $x{:}^{Q}\tau$

- $(x{:}^{\forall}\mathbf{int}) \rightarrow \{u : \mathbf{int} \mid u > x\}^{\forall\exists}$
  functions that, for any integer x and
  for any run with the argument x,
  return some integer u,
  which is greater than x
- $(x{:}^{\exists}\mathbf{int}) \rightarrow \{u : \mathbf{int} \mid u > x\}^{\forall\exists}$
  functions that, there exists an integer x,
  for any run with the argument x,
  return some integer u,
  which is greater than x

Skolemizing Existential Bindings

Rule:

Skolemization Predicate

$$\frac{\Delta, x :^{\exists} \tau \vDash \phi \qquad \Delta, x :^{\forall} \tau, \phi, \Gamma \vdash e : \sigma}{\Delta, x :^{\exists} \tau, \Gamma \vdash e : \sigma}$$

Type environment consisting of only ∀-bindings

Type environment consisting of both ∀- and ∃-bindings

Example:

$$\frac{x :^{\forall} \mathbf{int}, y :^{\exists} \mathbf{int} \vDash y = -x \qquad x :^{\forall} \mathbf{int}, y :^{\forall} \mathbf{int}, y = -x \vdash x + y : \{z \mid z = 0\}^{\forall\exists}}{x :^{\forall} \mathbf{int}, y :^{\exists} \mathbf{int} \vdash x + y : \{z \mid z = 0\}^{\forall\exists}}$$

Typing Non-deterministic Choice

Rule:

$$\frac{\Delta, x{:}^{Q_1}\mathbf{int} \vdash e : \tau^{Q_1 Q_2} \qquad x \notin fvs(\tau)}{\Delta \vdash \mathbf{let}\ x = *\ \mathbf{in}\ e : \tau^{Q_1 Q_2}}$$

Example:

$$\frac{x{:}^{\forall}\mathbf{int}, y{:}^{\exists}\mathbf{int} \vdash x + y : \{z \mid z = 0\}^{\exists\exists}}{x : ^{\forall}\mathbf{int} \vdash \mathbf{let}\ y = *\ \mathbf{in}\ x + y : \{z \mid z = 0\}^{\exists\exists}}$$

Typing Function Applications (Universal Bindings)

Rule:

$$\frac{\Delta \vdash v_1 : (x :^{\forall} \tau) \to \sigma \quad \Delta \vdash v_2 : \tau}{\Delta \vdash v_1\, v_2 : [v_2/x]\sigma}$$

Typing Function Applications (Existential Bindings)

Observational equivalence

Rule:

$$\frac{\Delta \vdash v_1 : (x{:}^{\exists}\tau) \rightarrow \sigma \qquad \Delta, x{:}^{\forall}\tau, \Gamma \vDash x \sim v_2}{\Delta, \Gamma \vdash v_1\ v_2 : [v_2/x]\sigma}$$

Example:

$$\frac{\dfrac{f{:}^{\forall}\tau \vdash f : \tau \qquad f{:}^{\forall}\tau, x{:}^{\forall}\mathbf{int}, y{:}^{\exists}\mathbf{int} \vDash x = y}{f{:}^{\forall}\tau, y{:}^{\exists}\mathbf{int} \vdash f\ \ y : \{z \mid\ \bot\}^{\exists\forall}}}{f{:}^{\forall}\tau \vdash \mathbf{let}\ y = *\ \ \mathbf{in}\ f\ y : \{z \mid\ \bot\}^{\exists\forall}}$$

$(x :^{\exists} \mathbf{int}) \rightarrow \{z \mid\ \bot\}^{\exists\forall}$

Converting Function Types (1/4)

Subtyping Rule:

$$\frac{\Delta \vdash \tau_2 <: \tau_1 \quad \Delta, x :^{\forall} \tau_2 \vdash \sigma_1 <: \sigma_2}{\Delta \vdash (x{:}^{\forall}\tau_1) \to \sigma_1 <: (x{:}^{\forall}\tau_2) \to \sigma_2}$$

Example:

$$\vdash (x{:}^{\forall}\mathbf{int}) \to \{y \mid y = x\}^{\forall\exists}$$
$$<: (x{:}^{\forall}\{x \mid x \geq \mathbf{0}\}) \to \{y \mid y \geq \mathbf{0}\}^{\forall\exists}$$

Converting Function Types (2/4)

Subtyping Rule:

$$\frac{\Delta \vdash \tau_1 <: \tau_2 \quad \Delta, x :^{\forall} \tau_1 \vdash \sigma_1 <: \sigma_2}{\Delta \vdash (x{:}^{\exists}\tau_1) \to \sigma_1 <: (x{:}^{\exists}\tau_2) \to \sigma_2}$$

Example:

$$\vdash (x{:}^{\exists}\{x \mid x \geq 0\}) \to \{y \mid y = x\}^{\forall\exists}$$
$$<: (x{:}^{\exists}\mathbf{int}) \to \{y \mid y \geq 0\}^{\forall\exists}$$

Converting Function Types (3/4)

Subtyping Rule:

$$\frac{\Delta \vdash \tau <: \tau_1 \quad \Delta \vdash \tau <: \tau_2 \quad \Delta, x :^{\exists} \tau \vdash \sigma_1 <: \sigma_2}{\Delta \vdash (x{:}^{\forall}\tau_1) \to \sigma_1 <: (x{:}^{\exists}\tau_2) \to \sigma_2}$$

Example:

$$\frac{\vdash \{x \mid x \geq 0\} <: \mathbf{int} \quad \vdash \{x \mid x \geq 0\} <: \mathbf{int} \quad x{:}^{\exists}\{x \mid x \geq 0\} \vdash \{y \mid y = x\}^{\forall\exists} <: \{y \mid y = 0\}^{\forall\exists}}{\vdash (x{:}^{\forall}\mathbf{int}) \to \{y \mid y = x\}^{\forall\exists} <: (x{:}^{\exists}\mathbf{int}) \to \{y \mid y = 0\}^{\forall\exists}}$$

Converting Function Types (4/4)

Subtyping Rule:

$$\frac{\begin{array}{c}\Delta, x{:}^{\forall}(\tau_1 \wedge \tau_2), y{:}^{\forall}(\tau_1 \wedge \tau_2) \vDash x \sim y \\ \Delta, x{:}^{\forall}(\tau_1 \wedge \tau_2) \vdash \sigma_1 <: \sigma_2 \\ \Delta, x{:}^{\forall}(\tau_1 \setminus \tau_2) \vdash \sigma_1 <: \bot \\ \Delta, x{:}^{\forall}(\tau_2 \setminus \tau_1) \vdash \top <: \sigma_2\end{array}}{\Delta \vdash (x{:}^{\exists}\tau_1) \to \sigma_1 <: (x{:}^{\forall}\tau_2) \to \sigma_2}$$

Observational equivalence

Example:

$$\vdash (x{:}^{\exists}\{x \mid x = 0\}) \to \{y \mid y = 0\}^{\forall\exists}$$
$$<: (x{:}^{\forall}\{x \mid x = 0\}) \to \{y \mid y = x\}^{\forall\exists}$$

Overview: Our Type System

- **Extends dependent refinement types with:**
  - Qualified types $\tau^{Q_1 Q_2}$
    to express universal/existential branching behaviors
    and partial/total correctness
  - Qualified bindings $x{:}^{Q}\tau$
    to cope with non-determinism from program inputs
  - Gödel encoding of function-type values
    & guarded intersection types
    to achieve relative completeness

Gödel Encoding of Function-Type Values

- Enables predicates of the underlying logic T (e.g., second-order arithmetic) to depend on function-type arguments encoded as T-objects

- $(f :^{\forall} \mathbf{int} \rightarrow \mathbf{int}^{\forall\exists}) \rightarrow (g :^{\forall} \mathbf{int} \rightarrow \mathbf{int}^{\forall\exists}) \rightarrow \{u \mid f \sim g\}^{\forall\forall}$
  Functions that, given two terminating functions f and g, always diverge if f is not observationally equivalent to g

Guarded Intersection Types

$$\bigwedge_i \left(\phi_i \rhd \tau_i^{Q_i Q_i'} \right)$$

- Collectively express different behaviors of functions depending on the arguments

- $(x :^{\forall} \mathbf{int}) \to (x > 0 \rhd \mathbf{int}^{\forall\exists}) \wedge (x < 0 \rhd \{y \mid \bot\}^{\forall\forall}) \wedge (x = 0 \rhd \mathbf{int}^{\exists\exists}) \wedge (x = 0 \rhd \{y \mid \bot\}^{\exists\forall})$

  Functions that, given the argument x :
  - always terminate if $x > 0$,
  - always diverge if $x < 0$, and otherwise,
  - non-deterministically terminate or diverge

Our Contributions

- Novel dependent refinement type system that can:
  - uniformly express and verify universal and existential branching properties of call-by-value, higher-order, and non-deterministic programs:
    - (cond.) safety, non-safety, termination, and non-termination
  - seamlessly combine universal and existential reasoning
    - e.g., Prove non-safety via termination
    - e.g., Prove non-termination via safety
    - e.g., Prove termination and non-termination simultaneously
- Meta-theoretic properties of the type system:
  - Closure of types under complement
  - Soundness
  - Relative completeness

Complement Types $\neg\sigma$

- Thanks to having both modes of non-determinism, the type complement operator $\neg$ can be defined:

$$\neg\left(\tau^{Q_1 Q_2}\right) \triangleq (\neg\tau)^{(\neg Q_1)(\neg Q_2)}$$

$$\neg\{x : \mathbf{int} \mid \phi\} \triangleq \{x : \mathbf{int} \mid \neg\phi\}$$

$$\neg\left(\left(x :^{Q} \tau\right) \rightarrow \sigma\right) \triangleq \left(x :^{\neg Q} \tau\right) \rightarrow \neg\sigma$$

$$\neg\forall \triangleq \exists \qquad \neg\exists \triangleq \forall$$

Example: Complement Types $\neg\sigma$

- $\sigma \triangleq (x :^{\forall} \mathbf{int}) \rightarrow \{u : \mathbf{int} \mid u = x\}^{\forall\forall}$
  functions that, for any integer x and for any run with the argument x, diverge or return an integer $u = x$

- $\neg\sigma = (x :^{\exists} \mathbf{int}) \rightarrow \{u : \mathbf{int} \mid u \neq x\}^{\exists\exists}$
  functions that, for some integer x and for some run with the argument x, terminate and return an integer $u \neq x$

Example: Combined Reasoning (Non-safety via Termination)

Goal: prove that

$$\mathbf{let\ rec}\ f\ x\ y =$$
$$\quad \mathbf{if}\ x = y\ \mathbf{then}\ 0$$
$$\quad \mathbf{else}\ f\ (x-1)\ y$$
$$\mathbf{let}\ r = f\ 10^9\ 0\ \mathbf{in}$$
$$\mathbf{let}\ z = *\ \mathbf{in}\ z + r$$

violates

$$\{u \mid u = 0\}^{\forall\forall}$$

Example: Combined Reasoning (Non-safety via Termination)

Goal: prove that

$$\mathbf{let\ rec}\ f\ x\ y =$$
$$\quad \mathbf{if}\ x = y\ \mathbf{then}\ 0$$
$$\quad \mathbf{else}\ f\ (x - 1)\ y$$
$$\mathbf{let}\ r = f\ 10^9\ 0\ \mathbf{in}$$
$$\mathbf{let}\ z = *\ \mathbf{in}\ z + r$$

satisfies

$$\neg(\{u \mid u = 0\}^{\forall\forall})$$

Example: Combined Reasoning (Non-safety via Termination)

Goal: prove that

$$\textbf{let rec } f\ x\ y = \\ \quad \textbf{if } x = y \textbf{ then } 0 \\ \quad \textbf{else } f\ (x-1)\ y \\ \textbf{let } r = f\ 10^9\ 0 \textbf{ in} \\ \textbf{let } z = * \textbf{ in } z + r$$

satisfies

$$\{u \mid u \neq 0\}^{\exists\exists}$$

1. Show that f is conditionally terminating:
   The well-founded relation
   $\lambda((x,y),(x',y')).\ x > x' \land y = y' \land x \geq y$
   witnesses that f has the type:
   $(x : \textbf{int}) \to (y : \{y \mid y \leq x\}) \to \textbf{int}^{\forall\exists}$
2. Show that the actual call $f\ 10^9\ 0$ always terminates by checking $\models 0 \leq 10^9$
3. Show that, for any integer r, we can choose an integer z such that $z + r \neq 0$

Q.E.D.

Our Contributions

- Novel dependent refinement type system that can:
  - uniformly express and verify universal and existential branching properties of call-by-value, higher-order, and non-deterministic programs:
    - (cond.) safety, non-safety, termination, and non-termination
  - seamlessly combine universal and existential reasoning
    - e.g., Prove non-safety via termination
    - e.g., Prove non-termination via safety
    - e.g., Prove termination and non-termination simultaneously
- Meta-theoretic properties of the type system:
  - Closure of types under complement
  - Soundness
  - Relative completeness

Closure of Types under Complement

For any type σ refining a simple type S,

- $[\![\sigma]\!] \cap [\![\neg\sigma]\!] = \emptyset$ and
- $[\![\sigma]\!] \cup [\![\neg\sigma]\!] = [\![S]\!]$

where

- $[\![\sigma]\!]$: the set of expressions that behave according to σ
- $[\![S]\!]$: the set of expressions of the type S

Soundness

$\Gamma \vdash e : \sigma$ **implies** $e \in [\![\Gamma \vdash \sigma]\!]$

the set of expressions that behave according to σ under any valuation conforming to Γ

Relative Completeness

$$e \in [\![\Gamma \vdash \sigma]\!] \textbf{ implies } \Gamma \vdash e : \sigma$$

under the assumption that the underlying logic is sufficiently expressible

- **to Gödel encode arbitrary functions definable in the target programming language**
- **to represent well-founded relations witnessing the termination of the definable functions**

Summary

- Novel dependent refinement type system that can:
  - uniformly express and verify universal and existential branching properties of call-by-value, higher-order, and non-deterministic programs:
    - (cond.) safety, non-safety, termination, and non-termination
  - seamlessly combine universal and existential reasoning
    - e.g., Prove non-safety via termination
    - e.g., Prove non-termination via safety
    - e.g., Prove termination and non-termination simultaneously
- Meta-theoretic properties of the type system:
  - Closure of types under complement
  - Soundness
  - Relative completeness

Future Work

- Extensions with temporal specifications:
  - Temporal trace specs. (e.g., LTL)
  - Branching temporal specs. (e.g., CTL, modal-μ)
- Extensions with language features:
  - Recursive data structures
  - Linked data structures
  - Call-by-name evaluation
  - Probabilistic choice
- Automation of type checking and inference

Towards Automation (Ongoing)

- Type Checking
  - How to leverage off-the-shelf SMT solvers?
  - ➢ Abstraction and counterexample guided refinement for the encoding of function-type values
- Type Inference
  - How to synthesize inductive invariants, well-founded relations, and Skolemization predicates?
  - ➢ Reduction to existentially-quantified Horn clause and well-foundedness constraints
  - How to achieve scalable inference?
  - ➢ Combination of universal and existential reasoning