(A Quick Intro to)
A Technique for Proving Subtyping Completeness, with an Application to Iso-recursive Types

Jay Ligatti

Joint work with:

Jeremy Blackburn   Ivory Hernandez   Michael Nachtigal

UNIVERSITY OF SOUTH FLORIDA

Suppose you’re defining a subtyping relation for a type-safe PL

What should your basic goals be?

Subtyping Relation Goals

- Soundness

$\tau_1 \leq \tau_2 \Rightarrow$ τ_1-type terms can always safely stand in for τ_2-type terms

- Completeness

τ_1-type terms can always safely stand in for τ_2-type terms $\Rightarrow \tau_1 \leq \tau_2$

Subtyping Relation Goals

trivially complete $\forall\tau_1,\tau_2 : \tau_1\leq\tau_2$

precise = sound and complete

trivially sound $\tau_1\leq\tau_2$ iff $\tau_1=\tau_2$

- *Preciseness* is a standard goal when defining ≤

Idea: ≤ is as complete as possible without sacrificing soundness

Proving Preciseness

- Soundness of ≤ can be proved with standard type-safety proofs
  - An unsound definition of ≤ would break type safety

Proving Preciseness

- Soundness of ≤ can be proved with standard type-safety proofs
  - An unsound definition of ≤ would break type safety

- Completeness of ≤ can be proved with

Problem

To Fill in the Blank,

- Need to state completeness property formally
- Then hopefully we can figure out how to prove it

To Fill in the Blank,

- Need to state completeness property formally
- Then hopefully we can figure out how to prove it

- Actually, let’s try to state the preciseness property formally…

Preciseness

- Intuition:
  $\tau_1 \leq \tau_2$ iff any term of type τ_2 could be replaced by any term of type τ_1 without breaking type safety

Preciseness

- Intuition:

  $\tau_1 \leq \tau_2$ iff any term of type τ_2 could be replaced by any term of type τ_1 without breaking type safety

- In other words:

  $\tau_1 \leq \tau_2$ iff τ_2-type expressions can—in any context—be replaced by τ_1-type expressions without causing well-typed programs to “get stuck”

Preciseness

$\tau_1 \le \tau_2$ iff τ_2-type expressions can—in any context—be replaced by τ_1-type expressions without causing well-typed programs to “get stuck”

Definition: A subtyping relation ≤ is precise wrt type safety when for all τ_1, τ_2:

$$\tau_1 \le \tau_2 \Leftrightarrow \begin{array}{l} \neg\exists\ e, E, \tau, e': \\ \underbrace{E[\tau_2]:\tau} \ \wedge\ e:\tau_1 \ \wedge\ E[e] \to^* e' \ \wedge\ \mathrm{stuck}(e') \end{array}$$

Filling evaluation context E’s hole with a τ_2-type expression produces a well-typed program

Soundness

$\tau_1 \le \tau_2$ iff τ_2-type expressions can—in any context—be replaced by τ_1-type expressions without causing well-typed programs to “get stuck”

Definition: A subtyping relation ≤ is sound wrt type safety when for all τ_1, τ_2:

$$\tau_1 \le \tau_2 \Rightarrow \quad \begin{array}{l} \neg\exists\ e, E, \tau, e': \\ \underbrace{E[\tau_2]:\tau} \ \wedge\ e:\tau_1 \ \wedge\ E[e] \rightarrow^* e' \ \wedge\ \text{stuck}(e') \end{array}$$

Filling evaluation context E’s hole with a τ_2 -type expression produces a well-typed program

Completeness

$\tau_1 \leq \tau_2$ iff τ_2-type expressions can—in any context—be replaced by τ_1-type expressions without causing well-typed programs to "get stuck"

Definition: A subtyping relation ≤ is complete wrt type safety when for all τ_1, τ_2:

$$\tau_1 \leq \tau_2 \Leftarrow \neg\exists\, e, E, \tau, e': \underbrace{E[\tau_2]:\tau}_{} \wedge\ e:\tau_1 \wedge\ E[e] \rightarrow^* e' \wedge\ \text{stuck}(e')$$

Filling evaluation context E's hole with a τ_2 -type expression produces a well-typed program

Soundness of ≤ is a Corollary of Type Safety

$$\tau_1 \le \tau_2 \Rightarrow \quad \begin{array}{l} \neg\exists\ e, E, \tau, e': \\ E[\tau_2]:\tau \ \wedge\ e:\tau_1 \ \wedge\ E[e] \to^* e' \ \wedge\ \text{stuck}(e') \end{array}$$

- Proof idea:

$$\text{Assume } \underbrace{\tau_1 \le \tau_2,\ E[\tau_2]:\tau,\ e:\tau_1,}\ \underbrace{E[e] \to^* e',\ \text{and stuck}(e')}$$

By subsumption and the definition of well-typed contexts, $E[e]:\tau$

But $E[e]:\tau$, $E[e] \to^* e'$, and $\text{stuck}(e')$ combine to contradict type safety

Proving Completeness

$$\neg\exists\ e, E, \tau, e': \quad E[\tau_2]:\tau \ \wedge\ e:\tau_1 \ \wedge\ E[e]\to^* e' \ \wedge\ \text{stuck}(e') \ \Rightarrow\ \tau_1 \leq \tau_2$$

Proving Completeness

$$\neg\exists\ e, E, \tau, e':$$

$$E[\tau_2]:\tau \ \land\ e:\tau_1 \ \land\ E[e]\rightarrow^{*}e' \ \land\ \mathrm{stuck}(e') \quad \Rightarrow \tau_1 \le \tau_2$$

hmm…

Proving Completeness

(contrapositive)

$\tau_1 \not\leq \tau_2 \Rightarrow$ $\exists$ e, E, τ, e':
E[τ_2]:τ $\wedge$ e:τ_1 $\wedge$ E[e]$\rightarrow$*e' $\wedge$ stuck(e')

- Approach: Define the subtyping relation in an algorithmic deductive system
  - i.e., the inference rules are deterministic, and all "attempted" derivations of $\tau_1 \leq \tau_2$ succeed/fail at a finite height

Proving Completeness

(contrapositive)

$\tau_1 \not\leq \tau_2 \Rightarrow$ $\exists$ e, E, τ, e’:
E[τ_2]:τ $\wedge$ e:τ_1 $\wedge$ E[e]$\rightarrow^*$e’ $\wedge$ stuck(e’)

- Approach: Define the subtyping relation in an algorithmic deductive system
  - i.e., the inference rules are deterministic, and all “attempted” derivations of $\tau_1 \leq \tau_2$ succeed/fail at a finite height

- Hence, because $\tau_1 \not\leq \tau_2$, there exists a unique, finite “failing derivation” of $\tau_1 \leq \tau_2$

Example Failing Derivation

$$\dfrac{\dfrac{}{\text{real} \le \text{real}}\checkmark \quad \dfrac{}{\text{int} \le \text{real}}\checkmark}{\text{real}\to\text{int} \le \text{real}\to\text{real}}\checkmark \qquad \dfrac{\dfrac{}{\text{real} \le \text{int}}\times \quad \dfrac{}{\text{real} \le \text{real}}\checkmark}{\text{int}\to\text{real} \le \text{real}\to\text{real}}\times}{(\text{real}\to\text{real})\to(\text{int}\to\text{real}) \;\le\; (\text{real}\to\text{int})\to(\text{real}\to\text{real})}\times$$

Types $\tau ::= \text{int} \mid \text{real} \mid \tau_1 \to \tau_2$

$$\boxed{\tau_1 \le \tau_2} \qquad \dfrac{}{\text{int} \le \text{int}} \qquad \dfrac{}{\text{real} \le \text{real}}$$

$$\dfrac{}{\text{int} \le \text{real}} \qquad \dfrac{\tau_3 \le \tau_1 \qquad \tau_2 \le \tau_4}{\tau_1 \to \tau_2 \;\le\; \tau_3 \to \tau_4}$$

Proving Completeness

$\tau_1 \not\leq \tau_2 \Rightarrow$ $\exists$ e, E, τ, e':

$E[\tau_2]:\tau \;\wedge\; e:\tau_1 \;\wedge\; E[e] \rightarrow^{*} e' \;\wedge\; \mathrm{stuck}(e')$

By induction on the unique, finite, failing derivation of $\tau_1 \leq \tau_2$

Proving Completeness

$$\tau_1 \not\leq \tau_2 \Rightarrow \quad \begin{array}{l} \exists\ e, E, \tau, e': \\ E[\tau_2]:\tau \ \wedge\ e:\tau_1 \ \wedge\ E[e] \to^{*} e' \ \wedge\ \text{stuck}(e') \end{array}$$

We'll trace the failure from a leaf to the root of the failing derivation tree, showing that completeness holds on each failing judgment along the way

$$\dfrac{\dfrac{\dfrac{}{\text{real} \leq \text{real}} \qquad \dfrac{}{\text{int} \leq \text{real}}}{\text{real}\to\text{int} \leq \text{real}\to\text{real}} \qquad \dfrac{\dfrac{}{\text{real} \leq \text{int}}^{\times} \qquad \dfrac{}{\text{real} \leq \text{real}}}{\text{int}\to\text{real} \leq \text{real}\to\text{real}}}{(\text{real}\to\text{real})\to(\text{int}\to\text{real}) \ \leq\ (\text{real}\to\text{int})\to(\text{real}\to\text{real})}$$

Proving Completeness

$$\tau_1 \not\leq \tau_2 \Rightarrow \quad \begin{array}{l} \exists\ e, E, \tau, e': \\ E[\tau_2]:\tau \ \wedge\ e:\tau_1 \ \wedge\ E[e] \rightarrow^{*} e' \ \wedge\ \mathrm{stuck}(e') \end{array}$$

We'll trace the failure from a leaf to the root of the failing derivation tree, showing that completeness holds on each failing judgment along the way

$$\dfrac{\dfrac{\dfrac{}{\mathrm{real} \leq \mathrm{real}} \qquad \dfrac{}{\mathrm{int} \leq \mathrm{real}}}{\mathrm{real} \rightarrow \mathrm{int} \leq \mathrm{real} \rightarrow \mathrm{real}} \qquad \dfrac{\dfrac{}{\mathrm{real} \leq \mathrm{int}}\text{✖} \qquad \dfrac{}{\mathrm{real} \leq \mathrm{real}}}{\mathrm{int} \rightarrow \mathrm{real} \leq \mathrm{real} \rightarrow \mathrm{real}}\text{✖}}{(\mathrm{real} \rightarrow \mathrm{real}) \rightarrow (\mathrm{int} \rightarrow \mathrm{real}) \ \leq\ (\mathrm{real} \rightarrow \mathrm{int}) \rightarrow (\mathrm{real} \rightarrow \mathrm{real})}$$

Proving Completeness

$$\tau_1 \not\leq \tau_2 \Rightarrow \quad \exists\, e, E, \tau, e': \quad E[\tau_2]:\tau \;\wedge\; e:\tau_1 \;\wedge\; E[e] \rightarrow^{*} e' \;\wedge\; \text{stuck}(e')$$

We'll trace the failure from a leaf to the root of the failing derivation tree, showing that completeness holds on each failing judgment along the way

$$\dfrac{\dfrac{\dfrac{}{\text{real} \le \text{real}} \qquad \dfrac{}{\text{int} \le \text{real}}}{\text{real}\rightarrow\text{int} \le \text{real}\rightarrow\text{real}} \qquad \dfrac{\dfrac{}{\text{real} \le \text{int}}\,\text{✖} \qquad \dfrac{}{\text{real} \le \text{real}}}{\text{int}\rightarrow\text{real} \le \text{real}\rightarrow\text{real}}\,\text{✖}}{(\text{real}\rightarrow\text{real})\rightarrow(\text{int}\rightarrow\text{real}) \;\le\; (\text{real}\rightarrow\text{int})\rightarrow(\text{real}\rightarrow\text{real})}\,\text{✖}$$

Base Cases of Completeness Proof

- 5 possible failing leaf judgments here:
  1. real≤int
  2. real $\leq \tau_3 \rightarrow \tau_4$
  3. int $\leq \tau_3 \rightarrow \tau_4$
  4. $\tau_3 \rightarrow \tau_4 \leq$ real
  5. $\tau_3 \rightarrow \tau_4 \leq$ int

Base Cases of Completeness Proof

- 5 possible failing leaf judgments here:
  1. real≤int
  2. real ≤ $\tau_3 \rightarrow \tau_4$
  3. int ≤ $\tau_3 \rightarrow \tau_4$
  4. $\tau_3 \rightarrow \tau_4$ ≤ real
  5. $\tau_3 \rightarrow \tau_4$ ≤ int

- In every case, an e, E, τ, e’ can be constructed such that $E[\tau_2]:\tau$, $e:\tau_1$, $E[e] \rightarrow^* e'$, and stuck(e’)

Inductive Step of Completeness Proof

- One case here: $$\frac{\tau_3 \leq \tau_1 \qquad \tau_2 \leq \tau_4}{\tau_1 \rightarrow \tau_2 \;\leq\; \tau_3 \rightarrow \tau_4}$$
- Assuming the completeness property holds on some failing premise, prove that it also holds on the failing conclusion

Inductive Step of Completeness Proof

- One case here: $\dfrac{\tau_3 \leq \tau_1 \qquad \tau_2 \leq \tau_4}{\tau_1 \rightarrow \tau_2 \ \leq \ \tau_3 \rightarrow \tau_4}$
- Assuming the completeness property holds on some failing premise, prove that it also holds on the failing conclusion
- Again, it can be done; please see tech report for details

Another Interesting Problem

- Let’s apply these techniques (for proving subtyping preciseness) to the problem of subtyping iso-recursive types

Quick Refresher on Recursive Types

- Are fundamental for typing aggregate data structures

- Heavily used in functional and object-oriented PLs
  - datatype list = Empty of unit | Node of int * list
  - class Integer {… public void add(Integer i) …}

Quick Refresher on Recursive Types

- There are 2 primary varieties of recursive types:
  - Iso-recursive systems require programmers to manually roll & unroll the recursion
    - ML and Haskell support iso-recursive types
  - Equi-recursive systems rely on type checkers to roll and unroll as needed, so programmers don’t have to
    - Modula-3 supports equi-recursive types

Amber Rules [Cardelli, 1986]

$$\frac{S\ U\ \{ t_1 \le t_2 \} \vdash \tau_1 \le \tau_2}{S \vdash \mu t_1.\tau_1 \le \mu t_2.\tau_2} \qquad \frac{}{S\ U\ \{ t_1 \le t_2 \} \vdash t_1 \le t_2}$$

- Standard, textbook rules for subtyping iso-recursive types
- These rules are elegant and sound

Incompleteness of the Amber Rules

- Define:

  $\tau_1 \equiv \mu L.\{add:(\mu i.\{add:i \rightarrow unit\}) \rightarrow unit,\ min:unit \rightarrow int\}$

  $\tau_2 \equiv \mu i'.\{add:i' \rightarrow unit\}$

Incompleteness of the Amber Rules

- Define:

  $\tau_1 \equiv \mu L.\{add:(\mu i.\{add:i \rightarrow unit\}) \rightarrow unit,\ min:unit \rightarrow int\}$

  $\tau_2 \equiv \mu i'.\{add:i' \rightarrow unit\}$

- τ_1 and τ_2 are natural encodings of class types

```
class GreatInteger extends Integer {
  …
  public void add(Integer i) {…}
  public int min() {…}
  …
}
```

```
class Integer {
  …
  public void add(Integer i) {…}
  …
}
```

Incompleteness of the Amber Rules

- Define:

  $\tau_1 \equiv \mu L.\{add:(\mu i.\{add:i \rightarrow unit\}) \rightarrow unit,\ min:unit \rightarrow int\}$

  $\tau_2 \equiv \mu i'.\{add:i' \rightarrow unit\}$

- τ_1 and τ_2 are natural encodings of class types

```
class GreatInteger extends Integer {
  …
  public void add(Integer i) {…}
  public int min() {…}
  …
}
```

```
class Integer {
  …
  public void add(Integer i) {…}
  …
}
```

- GreatInteger (τ_1) is a subclass of Integer (τ_2)
  $\Rightarrow$ we should be able to derive $\tau_1 \leq \tau_2$

Incompleteness of the Amber Rules

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\{L \le i'\} \vdash i' \le \mu i.\{\text{add}:i \to \text{unit}\}}\ \text{✖} \qquad \dfrac{}{\{L \le i'\} \vdash \text{unit} \le \text{unit}}\ \checkmark}{\{L \le i'\} \vdash (\mu i.\{\text{add}:i \to \text{unit}\}) \to \text{unit} \ \le \ i' \to \text{unit}}}{\{L \le i'\} \vdash \{\text{add}:(\mu i.\{\text{add}:i \to \text{unit}\}) \to \text{unit},\ \text{min}:\text{unit} \to \text{int}\} \le \{\text{add}:i' \to \text{unit}\}}}{\emptyset \vdash \mu L.\{\text{add}:(\mu i.\{\text{add}:i \to \text{unit}\}) \to \text{unit},\ \text{min}:\text{unit} \to \text{int}\} \le \mu i'.\{\text{add}:i' \to \text{unit}\}}$$

Incompleteness of the Amber Rules

$$\dfrac{\dfrac{\dfrac{\overline{\{L \le i'\} \vdash i' \le \mu i.\{add{:}i \to unit\}}\ \times \qquad \overline{\{L \le i'\} \vdash unit \le unit}\ \checkmark}{\{L \le i'\} \vdash (\mu i.\{add{:}i \to unit\}) \to unit \ \le\ i' \to unit}}{\{L \le i'\} \vdash \{add{:}(\mu i.\{add{:}i \to unit\}) \to unit,\ min{:}unit \to int\} \le \{add{:}i' \to unit\}}}{\emptyset \vdash \mu L.\{add{:}(\mu i.\{add{:}i \to unit\}) \to unit,\ min{:}unit \to int\} \le \mu i'.\{add{:}i' \to unit\}}$$

Problem: Amber rules don’t unroll recursive types in their premises, so their conclusions aren’t based on how iso-recursive types actually get used (i.e., eliminated)

New Iso-recursive Subtyping Rules

$$\frac{\mu t_1.\tau_1 \;\leq\; \mu t_1.\tau_1 \notin S \qquad S \cup \{\mu t_1.\tau_1 \;\leq\; \mu t_2.\tau_2\} \vdash [\mu t_1.\tau_1/t_1]\tau_1 \leq [\mu t_2.\tau_2/t_2]\tau_2}{S \vdash \mu t_1.\tau_1 \;\leq\; \mu t_2.\tau_2}$$

$$\frac{}{S \cup \{\mu t_1.\tau_1 \;\leq\; \mu t_2.\tau_2\} \vdash \mu t_1.\tau_1 \;\leq\; \mu t_2.\tau_2}$$

New Rules Enable Desired Derivation

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\{\mathbf{L}\le\mathbf{I'}, \mathbf{I'}\le\mathbf{I}, \mathbf{I}\le\mathbf{I'}\}\vdash\mathbf{I'}\le\mathbf{I}}\checkmark \qquad \dfrac{}{\{\mathbf{L}\le\mathbf{I'}, \mathbf{I'}\le\mathbf{I}, \mathbf{I}\le\mathbf{I'}\}\vdash\text{unit}\le\text{unit}}\checkmark}{\{\mathbf{L}\le\mathbf{I'}, \mathbf{I'}\le\mathbf{I}, \mathbf{I}\le\mathbf{I'}\} \vdash \mathbf{I}\to\text{unit} \le \mathbf{I'}\to\text{unit}}}{\{\mathbf{L}\le\mathbf{I'},\mathbf{I'}\le\mathbf{I},\mathbf{I}\le\mathbf{I'}\}\vdash\{\text{add:}\mathbf{I}\to\text{unit}\}\le\{\text{add:}\mathbf{I'}\to\text{unit}\}}}{\{\mathbf{L} \le \mathbf{I'}, \mathbf{I'} \le \mathbf{I}\} \vdash \mathbf{I} \le \mathbf{I'}} \qquad \dfrac{}{\{\mathbf{L} \le \mathbf{I'}, \mathbf{I'} \le \mathbf{I}\} \vdash \text{unit} \le \text{unit}}\checkmark}{\{\mathbf{L} \le \mathbf{I'}, \mathbf{I'} \le \mathbf{I}\} \vdash \mathbf{I'}\to\text{unit} \le \mathbf{I}\to\text{unit}}}{\{\mathbf{L} \le \mathbf{I'}, \mathbf{I'} \le \mathbf{I}\} \vdash \{\text{add:}\mathbf{I'}\to\text{unit}\} \le \{\text{add:}\mathbf{I}\to\text{unit}\}}}{\{\mathbf{L} \le \mathbf{I'}\} \vdash \mathbf{I'} \le \mathbf{I}} \qquad \dfrac{}{\{\mathbf{L} \le \mathbf{I'}\} \vdash\text{unit} \le \text{unit}}\checkmark}{\{\mathbf{L} \le \mathbf{I'}\} \vdash \mathbf{I}\to\text{unit} \le \mathbf{I'}\to\text{unit}}}{\{\mathbf{L} \le \mathbf{I'}\} \vdash \{\text{add:}\mathbf{I}\to\text{unit, min:unit}\to\text{int}\} \le \{\text{add:}\mathbf{I'}\to\text{unit}\}}}{\emptyset \vdash \underbrace{\mu\text{L.}\{\text{add:}(\underbrace{\mu\text{i.}\{\text{add:i}\to\text{unit}\}}_{\mathbf{I}})\to\text{unit, min:unit}\to\text{int}\}}_{\mathbf{L}} \le \underbrace{\mu\text{i}'.\{\text{add:i}'\to\text{unit}\}}_{\mathbf{I'}}}$$

New Rules are Precise

- Proof uses the techniques described earlier
- Proof also shows that the standard subtyping rules for function and (binary) sum and product types are precise as well

More Information

- Technical report:

  “Completely Subtyping Iso-recursive Types”

- Project webpage:

  http://www.cse.usf.edu/~ligatti/projects/completeness/