A New Formalization of Subtyping to Match Subclasses to Subtypes

Hyunik Na and *Sukyoung Ryu*

Programming Language Research Group
KAIST

June 6, 2014

This Paper

A New Formalization of Subtyping to Match Subclasses to Subtypes

Goal

A New Formalization of Subtyping to Match Subclasses to Subtypes

- Subclassing
  - `class ColorPoint extends Point`
  - explicit by "declarations"
  - easy for programmers
- Subtyping
  - $\mathrm{ColorPoint} <: \mathrm{Point}$
  - implicit by "inference"
  - easy for type checkers

Goal

A New Formalization of Subtyping to Match Subclasses to Subtypes

- Subclassing
  - `class ColorPoint extends Point`
  - explicit by "declarations"
  - easy for programmers
- Subtyping
  - ColorPoint <: Point
  - implicit by "inference"
  - easy for type checkers

Goal

A New Formalization of Subtyping to Match Subclasses to Subtypes

- Subclassing
  - `class ColorPoint extends Point`
  - explicit by "declarations"
  - easy for programmers
- Subtyping
  - $\mathrm{ColorPoint} <: \mathrm{Point}$
  - implicit by "inference"
  - easy for type checkers

Solution

A New Formalization of Subtyping to Match Subclasses to Subtypes

Motivation

The self-type idiom in Fortress:

```
trait Equality⟦Self extends Equality⟦Self⟧⟧
    abstract opr =(self, other : Self) : Boolean
end
```

This type

Motivation

The self-type idiom in Fortress:

```
trait Equality⟦Self extends Equality⟦Self⟧⟧
    abstract opr =(self, other : Self) : Boolean
end
```

This type

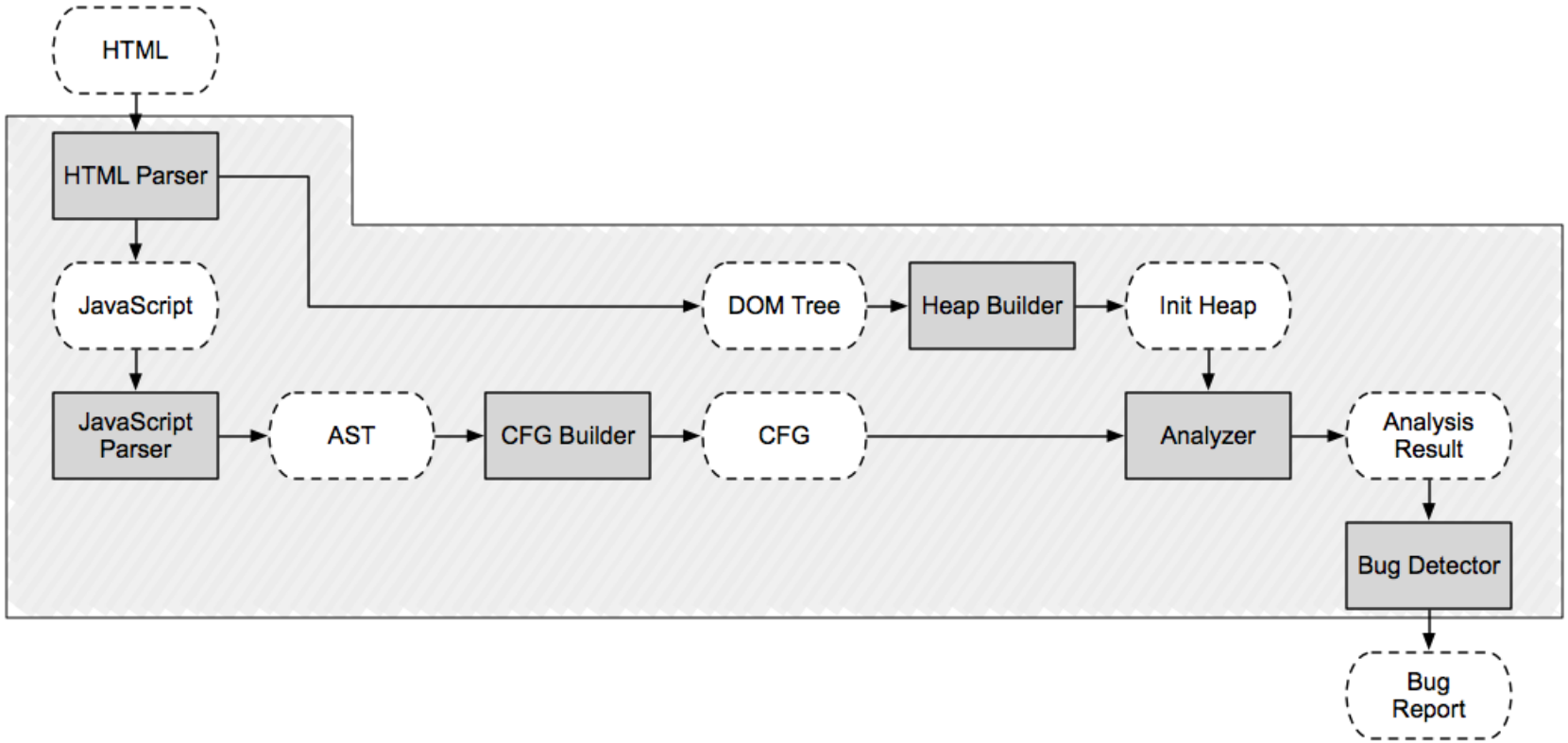
This **Type in Practice**

- Abstract domains for analyzing JavaScript programs

This **Type in Practice**

- Abstract domains for analyzing JavaScript programs

  SAFE: Scalable Analysis Framework for ECMAScript



This **Type in Practice**

- Abstract domains for analyzing JavaScript programs

$$\hat{v} \in \widehat{\text{Value}} = \widehat{\text{PValue}} \times \wp(\widehat{\text{Loc}})$$

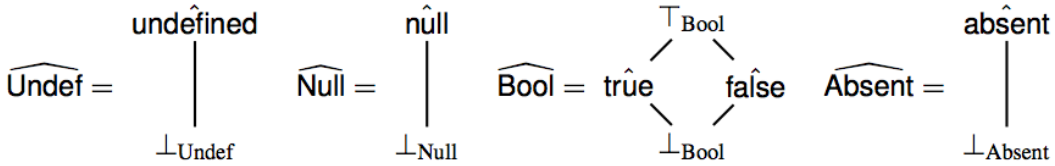
$$\hat{pv} \in \widehat{\text{PValue}} = \widehat{\text{Undef}} \times \widehat{\text{Null}} \times \widehat{\text{Bool}} \times \widehat{\text{Number}} \times \widehat{\text{String}}$$

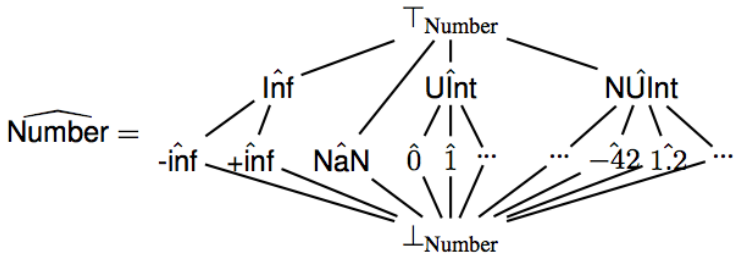
This **Type in Practice**

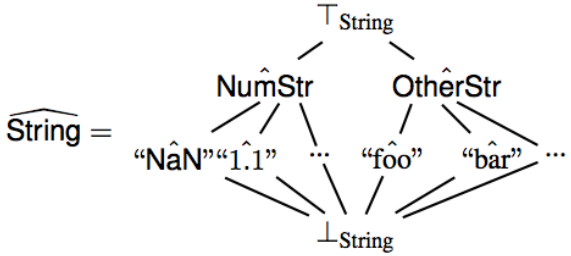
- Abstract domains for analyzing JavaScript programs

$$\hat{v} \in \widehat{\text{Value}} = \widehat{\text{PValue}} \times \wp(\widehat{\text{Loc}})$$

$$\hat{pv} \in \widehat{\text{PValue}} = \widehat{\text{Undef}} \times \widehat{\text{Null}} \times \widehat{\text{Bool}} \times \widehat{\text{Number}} \times \widehat{\text{String}}$$







This **Type in Practice**

- Abstract domains for analyzing JavaScript programs

```
abstract class AbsBase {
  def isTop(): Boolean
  def isBottom(): Boolean
  def isConcrete(): Boolean
  def toAbsString(): AbsString
}
```

This **Type in Practice**

- Abstract domains for analyzing JavaScript programs

```
sealed abstract class AbsNull extends AbsBase {
  /* partial order */
  def <= (that: AbsNull) = { ... }

  /* join */
  def + (that: AbsNull) = { ... }

  /* meet */
  def <> (that: AbsNull) = { ... }

  ...
}
```

This **Type in Practice**

- Abstract domains for analyzing JavaScript programs

```
sealed abstract class AbsString extends AbsBase {
  /* partial order */
  def <= (that: AbsString) = { ... }

  /* join */
  def + (that: AbsString) = { ... }

  /* meet */
  def <> (that: AbsString) = { ... }

  ...
}
```

This **Type in Practice**

- Abstract domains for analyzing JavaScript programs

```
abstract class AbsBase {
  /* partial order */
  def <= (that: ThisType): Boolean

  /* join */
  def + (that: ThisType): ThisType

  /* meet */
  def <> (that: ThisType): ThisType
  ...
}
```

This **Type in Practice**

- Abstract domains for analyzing JavaScript programs
  - Signatures of required methods in `AbsBase`
  - Exact class matches at run time

This **Type in Practice**

- Abstract domains for analyzing JavaScript programs
  - Signatures of required methods in `AbsBase`
  - Exact class matches at run time
- Bruno Oliveira's solution

```
abstract class AbsBase {
  type ThisType <: AbsBase
  def + (that: ThisType): ThisType
  ...
}
case class AbsString extends AbsBase {
  type ThisType = AbsString
  override def + (that : ThisType): ThisType =
    ... new AbsString() ...
}
```

This **Type in Practice**

- Abstract domains for analyzing JavaScript programs
  - Signatures of required methods in `AbsBase`
  - Exact class matches at run time
  - Multiple implementations for one abstract domain

```
abstract class AbsDomain { ... }
abstract class AbsBase[A] extends AbsDomain { ... }
class AbsString extends AbsBase[String] { ... }
class AbsStringSet extends AbsString { ... }
class AbsStringAutomata extends AbsString { ... }
```

This **Type in Practice**

- Abstract domains for analyzing JavaScript programs
  - Signatures of required methods in `AbsBase`
  - Exact class matches at run time
  - Multiple implementations for one abstract domain

```
abstract class AbsDomain { ... }

abstract class AbsBase[A] extends AbsDomain { ... }

class AbsString extends AbsBase[String] { ... }

class AbsStringSet extends AbsString { ... }

class AbsStringAutomata extends AbsString { ... }
```

This **Type in Practice**

12/03/2013 12:17 pm	Sooncheol Won	[AbsDomain] Changed string domain to set domain.
12/02/2013 05:48 pm	Sooncheol Won	[AbsDomain] 1) Renamed AbsStringSimple to AbsStringSet Added an AbsStringAutomata prototype class.
12/02/2013 04:45 pm	Sooncheol Won	[AbsDomain] Separated common domain elements(StrTop from AbsStringSimple.
12/02/2013 01:48 pm	Sooncheol Won	[AbsDomain] Removed useless parenthesises.
12/02/2013 12:01 pm	Sukyoung Ryu	[AbsDomain] Gave up with ThisType to support multiple implementations for AbsString. Refactored AbsString to Abs and AbsStringSimple.
12/02/2013 12:04 am	Sora Bae	Merge branch 'master' of ssh://plrg.kaist.ac.kr/var/git/saf
12/02/2013 12:03 am	Sora Bae	[Concolic] Create input objects for integer properties.
11/29/2013 10:13 pm	Sukyoung Ryu	[Refactoring] Refactored PropValue(Value(_))
11/29/2013 03:41 pm	Sooncheol Won	[Domain] Fixed the meet operator bug.
11/29/2013 12:51 pm	Sukyoung Ryu	[AbsNumber] Removed the getSingleValue method.
11/29/2013 09:34 am	Sukyoung Ryu	[AbsString] Moved one method from the AbsString object t AbsString class.
11/28/2013 11:02 am	Sukyoung Ryu	[AbsDomain] Removed uses of abstract value constructors
11/21/2013 09:59 am	Sukyoung Ryu	[AbsDomain] Revised the AbsUndef domain and added the AbsCas classes for pattern matching of abstract values.
11/20/2013 10:21 am	Sukyoung Ryu	[AbsNull] Revised the AbsNull domain.
11/20/2013 07:52 am	Sukyoung Ryu	[AbstractDomain] Made AbsBase covariantly parameterized by the of its cocrete values.
11/14/2013 07:31 pm	Sooncheol Won	[Domain] Changed the string domain.

This **Type in Theory**

This-typed methods:
owner types in their parameter types or return types

```
abstract class AbsBase {
  /* partial order */
  def <= (that: ThisType): Boolean

  /* join */
  def + (that: ThisType): ThisType

  /* meet */
  def <> (that: ThisType): ThisType
  ...
}
```

This **Type in Theory**

This-typed methods:
owner types in their parameter types or return types

- Traditional `This` type
  - "declared" type of a receiver
  - inexact compile-time type
- Our `This` type
  - "run-time" type of a receiver
  - exact run-time type
  - not available but sayable at compile time

This **Type in Theory**

This-typed methods:
owner types in their parameter types or return types

- Traditional `This` type
  - "declared" type of a receiver
  - inexact compile-time type
- Our `This` type
  - "run-time" type of a receiver
  - exact run-time type
  - not available but sayable at compile time

This **Type in Calculus (TLDI'12)**

CoreThisJava, a formal core calculus to support the This type:

- new typing features
  - *exact class types* of the form `#C` for a class `C`
  - `This` type variable
  - named wildcards of the form `</X/>` to describe more equality relationships between exact types
  - exact type inference to lessen the programmers' burden of using explicit annotations
- new language constructs
  - virtual constructors to describe methods with `ThisTyped` results
  - `classesmatch` to compare run-time types

This **Type in Java (APLAS'13)**

`ThisJava`, an open-source implementation using JastAddJ:

`http://plrg.kaist.ac.kr/research/software`

- backward compatible
  compilation to Java bytecode
- practical
  interactions with existing Java features

This **Type in Type System (FLOPS'14)**

- Traditional subtyping cannot support subtyping by inheritance.

```
class Point {
  int x;
  Point(int i) { this.x = i; }
  boolean equals(This other) { return this.x == other.x; }
}

class ColorPoint extends Point {
  int color;
  ColorPoint(int i, int c) { super(i); this.color = c; }
  boolean equals(This other) {
    return this.x == other.x && this.color == other.color;
  }
}
```

This **Type in Type System (FLOPS'14)**

- Traditional subtyping cannot derive the following:

$$\vdash \mu \mathtt{t}.\{\mathtt{x}\colon \mathtt{int}, \mathtt{c}\colon \mathtt{int}, \mathtt{eq}\colon \mathtt{t} \to \mathtt{bool}\} \leq \mu \mathtt{t}.\{\mathtt{x}\colon \mathtt{int}, \mathtt{eq}\colon \mathtt{t} \to \mathtt{bool}\}$$

- New subtyping can derive the following:

$$\emptyset \vdash \exists \mathtt{s} <: \mu \mathtt{t}.\{\mathtt{x}\colon \mathtt{int}, \mathtt{c}\colon \mathtt{int}, \mathtt{eq}\colon \mathtt{t} \to \mathtt{bool}\}.\mathtt{s} <: \exists \mathtt{s} <: \mu \mathtt{t}.\{\mathtt{x}\colon \mathtt{int}, \mathtt{eq}\colon \mathtt{t} \to \mathtt{bool}\}.\mathtt{s}$$

by distinguishing record types $\mu t.\{l_i\colon \tau_i\ ^{i \in 1..n}\}$
and some-record types $\exists s <: \mu t.\{l_i\colon \tau_i\ ^{i \in 1..n}\}.s$ explicitly

This **Type in Type System (FLOPS'14)**

- Traditional subtyping cannot derive the following:

$$\vdash \mu \mathtt{t}.\{\mathtt{x}\colon \mathtt{int}, \mathtt{c}\colon \mathtt{int}, \mathtt{eq}\colon \mathtt{t} \to \mathtt{bool}\} \leq \mu \mathtt{t}.\{\mathtt{x}\colon \mathtt{int}, \mathtt{eq}\colon \mathtt{t} \to \mathtt{bool}\}$$

- New subtyping can derive the following:

$$\emptyset \vdash \exists \mathtt{s} <: \mu \mathtt{t}.\{\mathtt{x}\colon \mathtt{int}, \mathtt{c}\colon \mathtt{int}, \mathtt{eq}\colon \mathtt{t} \to \mathtt{bool}\}.\mathtt{s} <: \exists \mathtt{s} <: \mu \mathtt{t}.\{\mathtt{x}\colon \mathtt{int}, \mathtt{eq}\colon \mathtt{t} \to \mathtt{bool}\}.\mathtt{s}$$

by distinguishing record types $\mu t.\{l_i\colon \tau_i\ ^{i \in 1..n}\}$
and some-record types $\exists s <: \mu t.\{l_i\colon \tau_i\ ^{i \in 1..n}\}.s$ explicitly

This **Type in Type System (FLOPS'14)**

- Traditional subtyping cannot derive the following:

$$\vdash \mu \mathtt{t}.\{\mathtt{x}\colon \mathtt{int}, \mathtt{c}\colon \mathtt{int}, \mathtt{eq}\colon \mathtt{t} \to \mathtt{bool}\} \leq \mu \mathtt{t}.\{\mathtt{x}\colon \mathtt{int}, \mathtt{eq}\colon \mathtt{t} \to \mathtt{bool}\}$$

- New subtyping can derive the following:

$$\emptyset \vdash \exists \mathtt{s} <_! \mu \mathtt{t}.\{\mathtt{x}\colon \mathtt{int}, \mathtt{c}\colon \mathtt{int}, \mathtt{eq}\colon \mathtt{t} \to \mathtt{bool}\}.\mathtt{s} <: \exists \mathtt{s} <_! \mu \mathtt{t}.\{\mathtt{x}\colon \mathtt{int}, \mathtt{eq}\colon \mathtt{t} \to \mathtt{bool}\}.\mathtt{s}$$

by distinguishing record types $\mu t.\{l_i\colon \tau_i\ ^{i \in 1..n}\}$
and some-record types $\exists s <_! \mu t.\{l_i\colon \tau_i\ ^{i \in 1..n}\}.s$ explicitly

This **Type in Type System (FLOPS'14)**

- Record types
  - $\mu t.\{l_i : \tau_i \ ^{i \in 1..n}\}$
  - types of "run-time" objects that have exactly those members l_1, l_2, ..., l_n
  - exact run-time types
- Some-record types
  - $\exists s <: \mu t.\{l_i : \tau_i \ ^{i \in 1..n}\}.s$
  - types of "compile-time" expressions that may evaluate to objects of record types specializing $\mu t.\{l_i : \tau_i \ ^{i \in 1..n}\}$
  - inexact compile-time types

This **Type in Type System (FLOPS'14)**

- Record types
  - $\mu t.\{l_i : \tau_i \ ^{i \in 1..n}\}$
  - types of "run-time" objects that have exactly those members $l_1, l_2, \ldots, l_n$
  - exact run-time types
- Some-record types
  - $\exists s <: \mu t.\{l_i : \tau_i \ ^{i \in 1..n}\}.s$
  - types of "compile-time" expressions that may evaluate to objects of record types specializing $\mu t.\{l_i : \tau_i \ ^{i \in 1..n}\}$
  - inexact compile-time types

This **Type in Type System (FLOPS'14)**

$$\begin{array}{rcll} \tau, \upsilon & ::= & t \mid \rho \mid \alpha \mid \gamma \mid \tau \to \tau & \text{type (revised)} \\ \alpha, \beta & ::= & \mu t.\{l_i : \tau_i \ ^{i \in 1..n}\} & \text{record type (revised)} \\ \gamma & ::= & \exists s <_! \alpha.s & \text{some-record type} \end{array}$$

Specializing: $\Delta \vdash \alpha <_! \beta$ where $\Delta ::= \{t_i \ ^{i \in 1..n}\}$

[Specializing]

$$\frac{n \geq 0,\ m \geq 0 \qquad t \notin \Delta \qquad \forall i \in 1..n:\ \Delta \cup \{t\} \vdash \tau_i <: \upsilon_i}{\Delta \vdash \mu t.\{l_i : \tau_i \ ^{i \in 1..n+m}\} <_! \mu t.\{l_i : \upsilon_i \ ^{i \in 1..n}\}}$$

This **Type in Type System (FLOPS'14)**

$$
\begin{array}{rcll}
\tau, \upsilon & ::= & t \mid \rho \mid \alpha \mid \gamma \mid \tau \to \tau & \text{type (revised)} \\
\alpha, \beta & ::= & \mu t.\{l_i : \tau_i \ ^{i \in 1..n}\} & \text{record type (revised)} \\
\gamma & ::= & \exists s <_! \alpha.s & \text{some-record type}
\end{array}
$$

Specializing: $\boxed{\Delta \vdash \alpha <_! \beta \quad \text{where} \quad \Delta ::= \{t_i \ ^{i \in 1..n}\}}$

[Specializing]

$$
\frac{n \geq 0,\ m \geq 0 \qquad t \notin \Delta \qquad \forall i \in 1..n:\ \Delta \cup \{t\} \vdash \tau_i <: \upsilon_i}{\Delta \vdash \mu t.\{l_i : \tau_i \ ^{i \in 1..n+m}\} <_! \mu t.\{l_i : \upsilon_i \ ^{i \in 1..n}\}}
$$

This **Type in Type System (FLOPS'14)**

$$
\begin{array}{rcll}
\tau, \upsilon & ::= & t \mid \rho \mid \alpha \mid \gamma \mid \tau \to \tau & \text{type (revised)} \\
\alpha, \beta & ::= & \mu t.\{l_i : \tau_i \ ^{i \in 1..n}\} & \text{record type (revised)} \\
\gamma & ::= & \exists s <_{\mathsf{I}} \alpha.s & \text{some-record type}
\end{array}
$$

Revised subtyping: $\boxed{\Delta \vdash \tau <: \upsilon \quad \text{where} \quad \Delta ::= \{t_i \ ^{i \in 1..n}\}}$

$$\text{[RS-Prim]} \quad \frac{}{\Delta \vdash \rho <: \rho}$$

$$\text{[RS-Func]} \quad \frac{\Delta \vdash \upsilon <: \tau \qquad \Delta \vdash \tau' <: \upsilon'}{\Delta \vdash \tau \to \tau' \ <: \ \upsilon \to \upsilon'}$$

$$\text{[RS-TVar]} \quad \frac{t \in \Delta}{\Delta \vdash t <: t}$$

$$\text{[RS-RtoR]} \quad \frac{}{\Delta \vdash \alpha <: \alpha}$$

$$\text{[RS-RtoS]} \quad \frac{\Delta \vdash \alpha <_{\mathsf{I}} \beta}{\Delta \vdash \alpha \ <: \ \exists s <_{\mathsf{I}} \beta.s}$$

$$\text{[RS-StoS]} \quad \frac{\Delta \vdash \alpha <_{\mathsf{I}} \beta}{\Delta \vdash \exists s <_{\mathsf{I}} \alpha.s \ <: \ \exists s <_{\mathsf{I}} \beta.s}$$

This **Type in Type System (FLOPS'14)**

- Record types $\mu t.\{l_i : \tau_i \ ^{i \in 1..n}\}$
- Some-record types $\exists s <_! \mu t.\{l_i : \tau_i \ ^{i \in 1..n}\}.s$
- Revised subtyping

$$\text{[RS-RToS]} \quad \frac{\Delta \vdash \alpha <_! \beta}{\Delta \vdash \alpha \ <: \ \exists s <_! \beta.s} \qquad \text{[RS-SToS]} \quad \frac{\Delta \vdash \alpha <_! \beta}{\Delta \vdash \exists s <_! \alpha.s \ <: \ \exists s <_! \beta.s}$$

Conclusion

- Supporting `ThisTyped` methods is an important real-world problem.
- Adding typing features and language constructs can support more `ThisTyped` methods.
- An open-source prototype implementation is available:

  `http://plrg.kaist.ac.kr/research/software`

- Inheritance can be subtyping.

$$\emptyset \vdash \quad \exists \mathtt{s} <_! \mu \mathtt{t}.\{\mathtt{x}\colon \mathtt{int}, \mathtt{c}\colon \mathtt{int}, \mathtt{eq}\colon \mathtt{t} \to \mathtt{bool}\}.\mathtt{s} \;<\colon \exists \mathtt{s} <_! \mu \mathtt{t}.\{\mathtt{x}\colon \mathtt{int}, \mathtt{eq}\colon \mathtt{t} \to \mathtt{bool}\}.\mathtt{s}$$