Undecidability of $D_{<:}$ and Its Decidable Fragments

Jason Z.S. Hu
University of Waterloo $\longrightarrow$
McGill University
zhong.s.hu@mail.mcgill.ca

Ondřej Lhoták
University of Waterloo
olhotak@uwaterloo.ca

- Scala was first released in 2004.

- Scala was first released in 2004.
- Formalization of Scala is a long running process (Odersky et al., 2003; Cremet et al., 2006; Moors et al., 2008; Amin et al., 2012; Rompf and Amin, 2016; Amin et al., 2016; Rapoport et al., 2017).

- Scala was first released in 2004.
- Formalization of Scala is a long running process (Odersky et al., 2003; Cremet et al., 2006; Moors et al., 2008; Amin et al., 2012; Rompf and Amin, 2016; Amin et al., 2016; Rapoport et al., 2017).
- How do type soundness proofs help to implement the compiler directly?

- Scala was first released in 2004.
- Formalization of Scala is a long running process (Odersky et al., 2003; Cremet et al., 2006; Moors et al., 2008; Amin et al., 2012; Rompf and Amin, 2016; Amin et al., 2016; Rapoport et al., 2017).
- How do type soundness proofs help to implement the compiler directly?

We consider the decidability of path dependent types,
and this theoretical result also benefits the implementation.

Path Dependent Types: An Example

Trait Definitions


```
trait Account

trait Bank { self =>
  type A <: Account
  def createAccount(initialBalance : Long = 0) : A
  def transfer(amount : Long, from : self.A,
               toBank : Bank, to : toBank.A) : Unit
}
```

Path Dependent Types: An Example

Trait Definitions


```
trait Account

trait Bank { self =>
  type A <: Account
  def createAccount(initialBalance : Long = 0) : A
  def transfer(amount : Long, from : self.A,
               toBank : Bank, to : toBank.A) : Unit
}
```

`toBank.A` depends on a previous parameter.

Path Dependent Types: An Example

A Tiny Program


```
def transfer(amount : Long, from : self.A,
             toBank : Bank, to : toBank.A) : Unit
```

```
object BankOfWaterloo extends Bank { /* ... */ }
object McGillBank extends Bank { /* ... */ }
val david : BankOfWaterloo.A = BankOfWaterloo.createAccount(200)
val elly  : McGillBank.A     = McGillBank.createAccount(300)
```

Path Dependent Types: An Example

A Tiny Program


```
def transfer(amount : Long, from : self.A,
             toBank : Bank, to : toBank.A) : Unit
```

```
object BankOfWaterloo extends Bank { /* ... */ }
object McGillBank extends Bank { /* ... */ }
val david : BankOfWaterloo.A = BankOfWaterloo.createAccount(200)
val elly  : McGillBank.A     = McGillBank.createAccount(300)

BankOfWaterloo.transfer(10, david, McGillBank,     elly)
```

This program works and transfers 10 dollars from David to Elly.

Path Dependent Types: An Example

A Tiny Program


```
def transfer(amount : Long, from : self.A,
             toBank : Bank, to : toBank.A) : Unit
```

```
object BankOfWaterloo extends Bank { /* ... */ }
object McGillBank extends Bank { /* ... */ }
val david : BankOfWaterloo.A = BankOfWaterloo.createAccount(200)
val elly  : McGillBank.A     = McGillBank.createAccount(300)
```

```
BankOfWaterloo.transfer(10, david, McGillBank,     elly)
```

```
BankOfWaterloo.transfer(10, david, BankOfWaterloo, elly)
```

What about this program?

```
def transfer(amount : Long, from : self.A,
             toBank : Bank, to : toBank.A) : Unit
```

```
object BankOfWaterloo extends Bank { /* ... */ }
object McGillBank extends Bank { /* ... */ }
val david : BankOfWaterloo.A = BankOfWaterloo.createAccount(200)
val elly  : McGillBank.A     = McGillBank.createAccount(300)

BankOfWaterloo.transfer(10, david, McGillBank,     elly)

BankOfWaterloo.transfer(10, david, BankOfWaterloo, elly)
```

found: `McGillBank.A`

expect: `BankOfWaterloo.A`

Research Questions

We can see that path dependent types are very expressive, but ...

Research Questions

We can see that path dependent types are very expressive, but ...

- Is type checking decidable with path dependent types?

Research Questions

We can see that path dependent types are very expressive, but ...

- Is type checking decidable with path dependent types?
- Is subtyping decidable with path dependent types?

Definition of $D_{<:}$ (Amin et al., 2016)

$$\frac{}{\Gamma \vdash_{D_{<:}} T <: \top}\ \textsc{Top} \qquad \frac{}{\Gamma \vdash_{D_{<:}} \bot <: T}\ \textsc{Bot} \qquad \frac{}{\Gamma \vdash_{D_{<:}} T <: T}\ \textsc{Refl}$$

$$\frac{}{\Gamma \vdash_{D_{<:}} T <: \top}\ \textsc{Top} \qquad \frac{}{\Gamma \vdash_{D_{<:}} \bot <: T}\ \textsc{Bot} \qquad \frac{}{\Gamma \vdash_{D_{<:}} T <: T}\ \textsc{Refl}$$

$$\boxed{\frac{\Gamma \vdash_{D_{<:}} S_2 <: S_1 \qquad \Gamma \vdash_{D_{<:}} U_1 <: U_2}{\Gamma \vdash_{D_{<:}} \{A : S_1..U_1\} <: \{A : S_2..U_2\}}\ \textsc{Bnd}}$$

$$\frac{}{\Gamma \vdash_{D_{<:}} T <: \top}\ \textsc{Top} \qquad \frac{}{\Gamma \vdash_{D_{<:}} \bot <: T}\ \textsc{Bot} \qquad \frac{}{\Gamma \vdash_{D_{<:}} T <: T}\ \textsc{Refl}$$

$$\frac{\Gamma \vdash_{D_{<:}} S_2 <: S_1 \quad \Gamma \vdash_{D_{<:}} U_1 <: U_2}{\Gamma \vdash_{D_{<:}} \{A : S_1..U_1\} <: \{A : S_2..U_2\}}\ \textsc{Bnd}$$

$$\boxed{\frac{\Gamma \vdash_{D_{<:}} S_2 <: S_1 \quad \Gamma; x : S_2 \vdash_{D_{<:}} U_1 <: U_2}{\Gamma \vdash_{D_{<:}} \forall(x : S_1)U_1 <: \forall(x : S_2)U_2}\ \textsc{All}}$$

Definition of $D_{<:}$ (Amin et al., 2016)

Path Dependent Types

$$\frac{}{\Gamma \vdash_{D_{<:}} T <: \top}\ \textsc{Top} \qquad \frac{}{\Gamma \vdash_{D_{<:}} \bot <: T}\ \textsc{Bot} \qquad \frac{}{\Gamma \vdash_{D_{<:}} T <: T}\ \textsc{Refl}$$

$$\frac{\Gamma \vdash_{D_{<:}} S_2 <: S_1 \qquad \Gamma \vdash_{D_{<:}} U_1 <: U_2}{\Gamma \vdash_{D_{<:}} \{A : S_1..U_1\} <: \{A : S_2..U_2\}}\ \textsc{Bnd} \qquad \frac{\Gamma \vdash_{D_{<:}} S_2 <: S_1 \qquad \Gamma; x : S_2 \vdash_{D_{<:}} U_1 <: U_2}{\Gamma \vdash_{D_{<:}} \forall(x : S_1)U_1 <: \forall(x : S_2)U_2}\ \textsc{All}$$

$$\boxed{\frac{\Gamma \vdash_{D_{<:}} \Gamma(x) <: \{A : S..\top\}}{\Gamma \vdash_{D_{<:}} S <: x.A}\ \textsc{Sel1'}} \qquad \frac{\Gamma \vdash_{D_{<:}} \Gamma(x) <: \{A : \bot..U\}}{\Gamma \vdash_{D_{<:}} x.A <: U}\ \textsc{Sel2'}$$

Definition of $D_{<:}$ (Amin et al., 2016)

Path Dependent Types

$$\dfrac{}{\Gamma \vdash_{D_{<:}} T <: \top}\ \textsc{Top} \qquad \dfrac{}{\Gamma \vdash_{D_{<:}} \bot <: T}\ \textsc{Bot} \qquad \dfrac{}{\Gamma \vdash_{D_{<:}} T <: T}\ \textsc{Refl}$$

$$\dfrac{\Gamma \vdash_{D_{<:}} S_2 <: S_1 \quad \Gamma \vdash_{D_{<:}} U_1 <: U_2}{\Gamma \vdash_{D_{<:}} \{A : S_1..U_1\} <: \{A : S_2..U_2\}}\ \textsc{Bnd} \qquad \dfrac{\Gamma \vdash_{D_{<:}} S_2 <: S_1 \quad \Gamma; x : S_2 \vdash_{D_{<:}} U_1 <: U_2}{\Gamma \vdash_{D_{<:}} \forall(x : S_1)U_1 <: \forall(x : S_2)U_2}\ \textsc{All}$$

$$\dfrac{\Gamma \vdash_{D_{<:}} \Gamma(x) <: \{A : S..\top\}}{\Gamma \vdash_{D_{<:}} S <: x.A}\ \textsc{Sel1'} \qquad \boxed{\dfrac{\Gamma \vdash_{D_{<:}} \Gamma(x) <: \{A : \bot..U\}}{\Gamma \vdash_{D_{<:}} x.A <: U}\ \textsc{Sel2'}}$$

Definition of $D_{<:}$ (Amin et al., 2016)

Path Dependent Types

$$\frac{}{\Gamma \vdash_{D_{<:}} T <: \top}\ \textsc{Top} \qquad \frac{}{\Gamma \vdash_{D_{<:}} \bot <: T}\ \textsc{Bot} \qquad \frac{}{\Gamma \vdash_{D_{<:}} T <: T}\ \textsc{Refl}$$

$$\frac{\begin{array}{c}\Gamma \vdash_{D_{<:}} S_2 <: S_1 \\ \Gamma \vdash_{D_{<:}} U_1 <: U_2\end{array}}{\Gamma \vdash_{D_{<:}} \{A : S_1..U_1\} <: \{A : S_2..U_2\}}\ \textsc{Bnd} \qquad \frac{\begin{array}{c}\Gamma \vdash_{D_{<:}} S_2 <: S_1 \\ \Gamma; x : S_2 \vdash_{D_{<:}} U_1 <: U_2\end{array}}{\Gamma \vdash_{D_{<:}} \forall(x : S_1)U_1 <: \forall(x : S_2)U_2}\ \textsc{All}$$

$$\frac{\Gamma \vdash_{D_{<:}} \Gamma(x) <: \{A : S..\top\}}{\Gamma \vdash_{D_{<:}} S <: x.A}\ \textsc{Sel1'} \qquad \frac{\Gamma \vdash_{D_{<:}} \Gamma(x) <: \{A : \bot..U\}}{\Gamma \vdash_{D_{<:}} x.A <: U}\ \textsc{Sel2'}$$

$$\boxed{\frac{\Gamma \vdash_{D_{<:}} S <: T \qquad \Gamma \vdash_{D_{<:}} T <: U}{\Gamma \vdash_{D_{<:}} S <: U}\ \textsc{Trans}}$$

Outline of Our Undecidability Proof

The actual proof is quite tricky, e.g. the TRANS rule doesn't provide strong enough inductive hypothesis.

Outline of Our Undecidability Proof

The actual proof is quite tricky, e.g. the TRANS rule doesn't provide strong enough inductive hypothesis.

To establish the proof, we

Outline of Our Undecidability Proof

The actual proof is quite tricky, e.g. the TRANS rule doesn't provide strong enough inductive hypothesis.

To establish the proof, we

1. find a suitable undecidable problem to reduce from,

Outline of Our Undecidability Proof

The actual proof is quite tricky, e.g. the TRANS rule doesn't provide strong enough inductive hypothesis.

To establish the proof, we

1. find a suitable undecidable problem to reduce from,
2. define $D_{<:}$ normal form by restricting the TRANS rule,

Outline of Our Undecidability Proof

The actual proof is quite tricky, e.g. the TRANS rule doesn't provide strong enough inductive hypothesis.

To establish the proof, we

1. find a suitable undecidable problem to reduce from,
2. define $D_{<:}$ normal form by restricting the TRANS rule,
3. show the equivalence between $D_{<:}$ and $D_{<:}$ normal form,

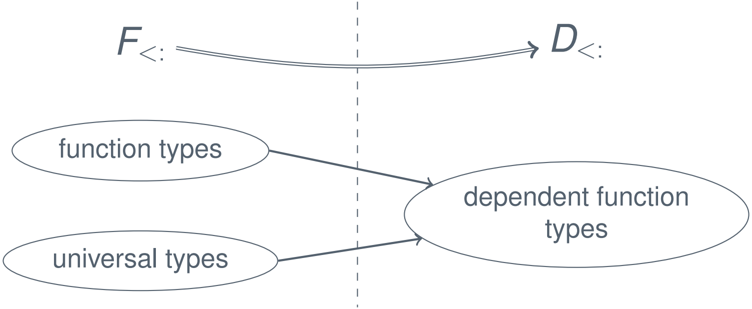
Outline of Our Undecidability Proof

The actual proof is quite tricky, e.g. the TRANS rule doesn't provide strong enough inductive hypothesis.

To establish the proof, we

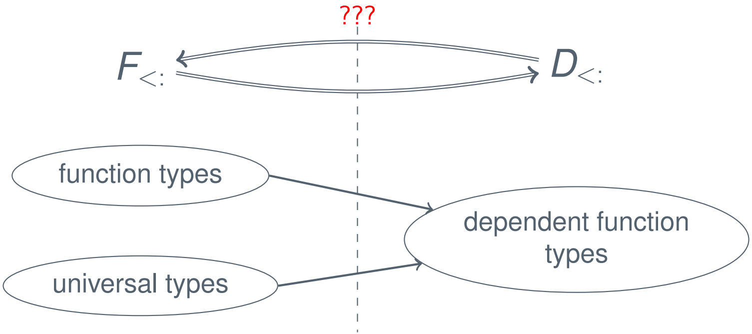
1. find a suitable undecidable problem to reduce from,
2. define $D_{<:}$ normal form by restricting the TRANS rule,
3. show the equivalence between $D_{<:}$ and $D_{<:}$ normal form,
4. conclude undecidability of $D_{<:}$.

Finding An Undecidable Problem
Step 1



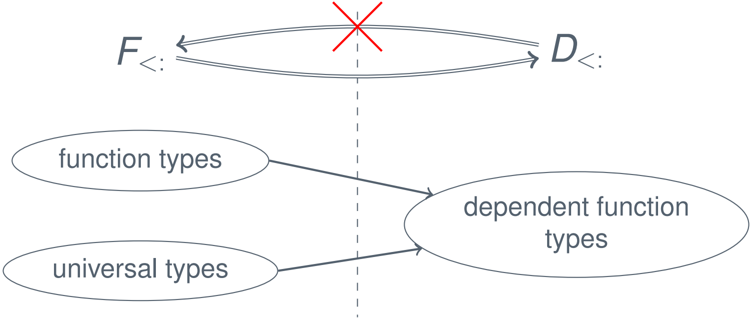
Amin et al. (2016) presents an attempt.

Finding An Undecidable Problem
Step 1



Amin et al. (2016) presents an attempt.

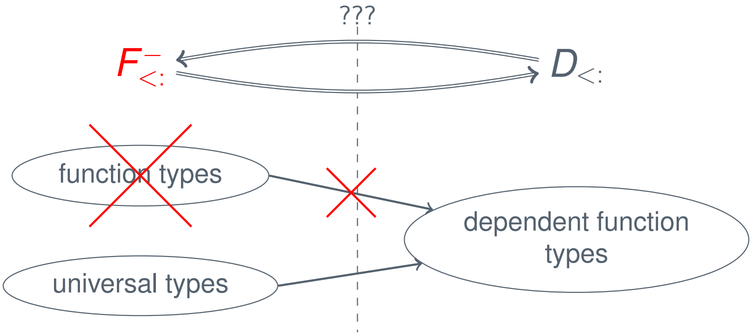
Finding An Undecidable Problem
Step 1



Amin et al. (2016) presents an attempt.

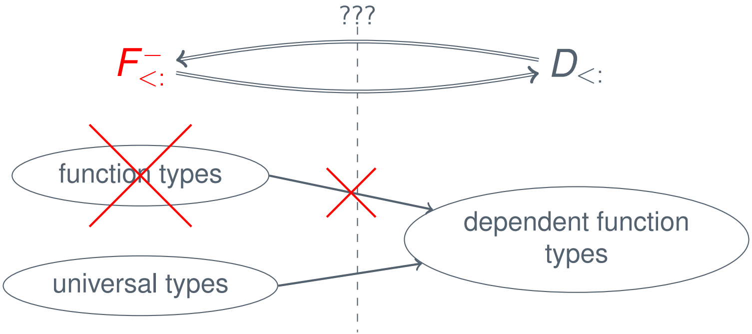
Finding An Undecidable Problem

Step 1



Finding An Undecidable Problem

Step 1



Theorem

Subtyping of $F^{-}_{<:}$ is undecidable.

Transitivity and Subtyping Reflection

Step 2

The Trans rule induces an unexpected phenomenon:

assume $\Gamma(x) = \{A : S..U\}$

$$\dfrac{\dfrac{\Gamma \vdash_{D_{<:}} \Gamma(x) <: \{A : S..\top\}}{\Gamma \vdash_{D_{<:}} S <: x.A}\ \textsc{Sel1'} \qquad \dfrac{\Gamma \vdash_{D_{<:}} \Gamma(x) <: \{A : \bot..U\}}{\Gamma \vdash_{D_{<:}} x.A <: U}\ \textsc{Sel2'}}{\Gamma \vdash_{D_{<:}} S <: U}\ \textsc{Trans}$$

Transitivity and Subtyping Reflection

Step 2

The Trans rule induces an unexpected phenomenon:

$$\text{assume } \Gamma(x) = \{A : S..U\}$$

$$\dfrac{\dfrac{\Gamma \vdash_{D_{<:}} \Gamma(x) <: \{A : S..\top\}}{\Gamma \vdash_{D_{<:}} S <: x.A}\ \textsc{Sel1'} \qquad \dfrac{\Gamma \vdash_{D_{<:}} \Gamma(x) <: \{A : \bot..\color{red}{U}\}}{\Gamma \vdash_{D_{<:}} x.A <: \color{red}{U}}\ \color{red}{\textsc{Sel2'}}}{\Gamma \vdash_{D_{<:}} S <: \color{red}{U}}\ \textsc{Trans}$$

Transitivity and Subtyping Reflection

Step 2

The TRANS rule induces an unexpected phenomenon:

$$\text{assume } \Gamma(x) = \{A : S..U\}$$

$$\dfrac{\dfrac{\Gamma \vdash_{D_{<:}} \Gamma(x) <: \{A : S..\top\}}{\Gamma \vdash_{D_{<:}} S <: x.A}\ \textsc{Sel1'} \qquad \dfrac{\Gamma \vdash_{D_{<:}} \Gamma(x) <: \{A : \bot..U\}}{\Gamma \vdash_{D_{<:}} x.A <: U}\ \textsc{Sel2'}}{\Gamma \vdash_{D_{<:}} S <: U}\ \textsc{Trans}$$

Type declarations reflect bounds into the subtyping relation.

Transitivity and Subtyping Reflection

Step 2

The TRANS rule induces an unexpected phenomenon:

$$\text{assume } \Gamma(x) = \{A : S..U\}$$

$$\dfrac{\dfrac{\Gamma \vdash_{D_{<:}} \Gamma(x) <: \{A : S..\top\}}{\Gamma \vdash_{D_{<:}} S <: x.A}\ \textsc{Sel1'} \qquad \dfrac{\Gamma \vdash_{D_{<:}} \Gamma(x) <: \{A : \bot..U\}}{\Gamma \vdash_{D_{<:}} x.A <: U}\ \textsc{Sel2'}}{\Gamma \vdash_{D_{<:}} S <: U}\ \textsc{Trans}$$

Type declarations reflect bounds into the subtyping relation.
This phenomenon is called "subtyping reflection" (or "bad bounds" in the previous literature).

Subtyping reflection is captured by the following rule:

$$\frac{\Gamma \vdash_{D_{<:}} \Gamma(x) <: \{A : S..\top\} \qquad \Gamma \vdash_{D_{<:}} \Gamma(x) <: \{A : \bot..U\} \qquad \text{(for some } x\text{)}}{\Gamma \vdash_{D_{<:}} S <: U} \text{ SR}$$

$D_{<:}$ Normal Form
Step 2

Subtyping reflection is captured by the following rule:

$$\frac{\Gamma \vdash_{D_{<:}} \Gamma(x) <: \{A : S..\top\} \qquad \Gamma \vdash_{D_{<:}} \Gamma(x) <: \{A : \bot..U\} \qquad \text{(for some } x\text{)}}{\Gamma \vdash_{D_{<:}} S <: U} \text{ SR}$$

We replace the TRANS rule with this rule.

The resulting calculus is called $D_{<:}$ normal form.

Properties of $D_{<:}$ Normal Form

Step 3

Theorem

$D_{<:}$ normal form admits transitivity.

Theorem

$D_{<:}$ normal form admits transitivity.

Theorem

Subtyping in the original $D_{<:}$ definition and in $D_{<:}$ normal form is equivalent.

Undecidability of $D_{<:}$ Subtyping

Step 4

$$
\begin{tikzcd}
F^{-}_{<:} \arrow[r, bend right, Rightarrow, "\text{easy}"'] & D_{<:} \text{ normal form} \arrow[l, bend right, Rightarrow, "???"'] \arrow[r, Leftrightarrow, "\text{equivalence}"] & D_{<:}
\end{tikzcd}
$$

$$\begin{tikzcd} F^{-}_{<:} \arrow[r, Rightarrow, bend right, "\text{easy}"'] & D_{<:}\ \text{normal form} \arrow[l, Rightarrow, bend right] \arrow[r, Leftrightarrow, "\text{equivalence}"] & D_{<:} \end{tikzcd}$$

Theorem

Subtyping in $D_{<:}$ normal form is undecidable.

Undecidability of $D_{<:}$ Subtyping

Step 4

$$
\begin{tikzcd}
F_{<:}^{-} \arrow[r, bend right, Rightarrow, "\text{easy}"'] & D_{<:}\ \text{normal form} \arrow[l, bend right, Rightarrow] \arrow[r, Leftrightarrow, "\text{equivalence}"] & D_{<:}
\end{tikzcd}
$$

Theorem

Subtyping in $D_{<:}$ normal form is undecidable.

Theorem

$D_{<:}$ subtyping is undecidable.

A Thought about $D_{<:}$

Subtyping reflection and transitivity are two sides of the same coin.

Step toward Decidable Fragments

Capturing subtyping reflection inspires us to a straightforward study of decidable fragments of $D_{<:}$.

Kernel $D_{<:}$

Consider the following rules from $D_{<:}$ normal form:

$$\frac{\Gamma \vdash_{D_{<:}} S_2 <: S_1 \qquad \Gamma; x : S_2 \vdash_{D_{<:}} U_1 <: U_2}{\Gamma \vdash_{D_{<:}} \forall(x : S_1)U_1 <: \forall(x : S_2)U_2} \text{ ALL}$$

$$\frac{\Gamma \vdash_{D_{<:}} \Gamma(x) <: \{A : S..\top\} \qquad \Gamma \vdash_{D_{<:}} \Gamma(x) <: \{A : \bot..U\} \qquad \text{(for some } x\text{)}}{\Gamma \vdash_{D_{<:}} S <: U} \text{ SR}$$

Kernel $D_{<:}$

Consider the following rules from $D_{<:}$ normal form:

$$\frac{\Gamma \vdash_{D_{<:}} S_2 <: S_1 \qquad \Gamma; x : S_2 \vdash_{D_{<:}} U_1 <: U_2}{\Gamma \vdash_{D_{<:}} \forall(x : S_1)U_1 <: \forall(x : S_2)U_2} \text{ All}$$

$$\frac{\Gamma \vdash_{D_{<:}} \Gamma(x) <: \{A : S..\top\} \qquad \Gamma \vdash_{D_{<:}} \Gamma(x) <: \{A : \bot..U\} \qquad \text{(for some } x\text{)}}{\Gamma \vdash_{D_{<:}} S <: U} \text{ SR}$$

Kernel $D_{<:}$

Consider the following rules from $D_{<:}$ normal form:

$$\frac{\Gamma; \color{red}{x : S} \vdash_{D_{<:}K} U_1 <: U_2}{\Gamma \vdash_{D_{<:}K} \forall(x : \color{red}{S})U_1 <: \forall(x : \color{red}{S})U_2} \text{ K-ALL}$$

$$\cancel{\frac{\Gamma \vdash_{D_{<:}} \Gamma(x) <: \{A : S..\top\} \qquad \Gamma \vdash_{D_{<:}} \Gamma(x) <: \{A : \bot..U\} \qquad \text{(for some } x\text{)}}{\Gamma \vdash_{D_{<:}} S <: U}} \text{ SR}$$

Kernel $D_{<:}$

Consider the following rules from $D_{<:}$ normal form:

$$\frac{\Gamma; x : S \vdash_{D_{<:}K} U_1 <: U_2}{\Gamma \vdash_{D_{<:}K} \forall(x : S)U_1 <: \forall(x : S)U_2} \text{ K-ALL}$$

$$\cancel{\frac{\Gamma \vdash_{D_{<:}} \Gamma(x) <: \{A : S..\top\} \qquad \Gamma \vdash_{D_{<:}} \Gamma(x) <: \{A : \bot..U\} \qquad \text{(for some } x\text{)}}{\Gamma \vdash_{D_{<:}} S <: U} \text{ SR}}$$

These modifications define kernel $D_{<:}$.

Theorem

Kernel $D_{<:}$ is decidable.

Proof.

The decision procedure is step subtyping designed by Nieto (2017). □

A Limitation of Kernel $D_{<:}$

$$x : \{A : \top..\top\} \vdash_{D_{<:}} \forall(y : x.A)\top <: \forall(y : \top)\top$$

is rejected by kernel $D_{<:}$.

A Limitation of Kernel $D_{<:}$

$$x : \{A : \top..\top\} \vdash_{D_{<:}} \forall(y : x.A)\top <: \forall(y : \top)\top$$

is rejected by kernel $D_{<:}$.

Asymmetry and Symmetry

We want to lift the previous limitation.

Asymmetry and Symmetry

We want to lift the previous limitation.

The undecidability proof indicates the problem being the asymmetry of the parameter types of dependent function types.

Asymmetry and Symmetry

We want to lift the previous limitation.

The undecidability proof indicates the problem being the asymmetry of the parameter types of dependent function types.

The idea is to recover the symmetry by operating on **two contexts** at the same time.

Strong Kernel $D_{<:}$

$\Gamma \vdash_{D_{<:}K} S <: U$	$\Rightarrow$	$(\Gamma_1 \vdash S) <: (U \dashv \Gamma_2)$
Kernel $D_{<:}$		Strong kernel $D_{<:}$

Strong Kernel $D_{<:}$

$\Gamma \vdash_{D_{<:}K} S <: U$ $\Rightarrow$ $(\Gamma_1 \vdash S) <: (U \dashv \Gamma_2)$

Kernel $D_{<:}$ Strong kernel $D_{<:}$

In $(\Gamma_1 \vdash S) <: (U \dashv \Gamma_2)$, a type only concerns the context on its side:

$$\frac{(\Gamma_1 \vdash \Gamma_1(x)) <: (\{A : \bot..U\} \dashv \Gamma_2)}{(\Gamma_1 \vdash x.A) <: (U \dashv \Gamma_2)} \text{ SK-SEL2}$$

Strong Kernel $D_{<:}$

$\Gamma \vdash_{D_{<:}K} S <: U$	$\Rightarrow$	$(\Gamma_1 \vdash S) <: (U \dashv \Gamma_2)$
Kernel $D_{<:}$		Strong kernel $D_{<:}$

In $(\Gamma_1 \vdash S) <: (U \dashv \Gamma_2)$, a type only concerns the context on its side:

$$\frac{(\Gamma_1 \vdash \Gamma_1(x)) <: (\{A : \bot..U\} \dashv \Gamma_2)}{(\Gamma_1 \vdash x.A) <: (U \dashv \Gamma_2)} \textsc{Sk-Sel2}$$

$$\frac{{\color{red}(\Gamma_2 \vdash S_2) <: (S_1 \dashv \Gamma_1)} \qquad (\Gamma_1; x : S_1 \vdash U_1) <: (U_2 \dashv \Gamma_2; x : S_2)}{(\Gamma_1 \vdash \forall(x : {\color{red}S_1})U_1) <: (\forall(x : {\color{red}S_2})U_2 \dashv \Gamma_2)} \textsc{Sk-All}$$

Strong Kernel $D_{<:}$

$\Gamma \vdash_{D_{<:}K} S <: U$ $\Rightarrow$ $(\Gamma_1 \vdash S) <: (U \dashv \Gamma_2)$

Kernel $D_{<:}$ — Strong kernel $D_{<:}$

In $(\Gamma_1 \vdash S) <: (U \dashv \Gamma_2)$, a type only concerns the context on its side:

$$\frac{(\Gamma_1 \vdash \Gamma_1(x)) <: (\{A : \bot..U\} \dashv \Gamma_2)}{(\Gamma_1 \vdash x.A) <: (U \dashv \Gamma_2)} \text{ SK-SEL2}$$

$$\frac{(\Gamma_2 \vdash S_2) <: (S_1 \dashv \Gamma_1) \qquad (\Gamma_1 ; x : S_1 \vdash U_1) <: (U_2 \dashv \Gamma_2 ; x : S_2)}{(\Gamma_1 \vdash \forall(x : S_1)U_1) <: (\forall(x : S_2)U_2 \dashv \Gamma_2)} \text{ SK-ALL}$$

Theorem

Strong kernel $D_{<:}$ is decidable.

Proof.

The decision procedure is stare-at subtyping (defined in the paper). $\square$

Theorem

Strong kernel $D_{<:}$ is strictly stronger than kernel $D_{<:}$.

$$x : \{A : \top..\top\} \vdash_{D_{<:}} \forall(y : x.A)\top <: \forall(y : \top)\top$$

becomes admissible.

Theorem

Strong kernel $D_{<:}$ is strictly stronger than kernel $D_{<:}$.

$$x : \{A : \top..\top\} \vdash_{D_{<:}} \forall(y : x.A)\top <: \forall(y : \top)\top$$

becomes admissible.
let $\Gamma = x : \{A : \top..\top\}$

$$\frac{(\Gamma \vdash \top) <: (x.A \dashv \Gamma) \qquad (\Gamma; y : x.A \vdash \top) <: (\top \dashv \Gamma; y : \top)}{(\Gamma \vdash \forall(y : x.A)\top) <: (\forall(y : \top)\top \dashv \Gamma)} \text{ SK-ALL}$$

Summary

- For theorists: we present a systematic way of investigating (un)decidability!
- For practitioners: we develop algorithms for path dependent types!

References

Nada Amin, Samuel Grütter, Martin Odersky, Tiark Rompf, and Sandro Stucki. 2016. The Essence of Dependent Object Types. In A List of Successes That Can Change the World - Essays Dedicated to Philip Wadler on the Occasion of His 60th Birthday (Lecture Notes in Computer Science), Sam Lindley, Conor McBride, Philip W. Trinder, and Donald Sannella (Eds.), Vol. 9600. Springer, 249–272. https://doi.org/10.1007/978-3-319-30936-1_14

Nada Amin, Adriaan Moors, and Martin Odersky. 2012. Dependent object types. In 19th International Workshop on Foundations of Object-Oriented Languages.

Vincent Cremet, François Garillot, Sergueï Lenglet, and Martin Odersky. 2006. A Core Calculus for Scala Type Checking. In Mathematical Foundations of Computer Science 2006, Rastislav Královič and Paweł Urzyczyn (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 1–23.

Adriaan Moors, Frank Piessens, and Martin Odersky. 2008. Safe type-level abstraction in Scala. In Proceedings of the International Workshop on Foundations of Object-Oriented Languages (FOOL 2008). 1–13.

Abel Nieto. 2017. Towards Algorithmic Typing for DOT (Short Paper). In Proceedings of the 8th ACM SIGPLAN International Symposium on Scala (SCALA 2017). ACM, New York, NY, USA, 2–7. https://doi.org/10.1145/3136000.3136003

Martin Odersky, Vincent Cremet, Christine Röckl, and Matthias Zenger. 2003. A Nominal Theory of Objects with Dependent Types. In ECOOP 2003 - Object-Oriented Programming, 17th European Conference, Darmstadt, Germany, July 21-25, 2003, Proceedings (Lecture Notes in Computer Science), Luca Cardelli (Ed.), Vol. 2743. Springer, 201–224. https://doi.org/10.1007/978-3-540-45070-2_10

Marianna Rapoport, Ifaz Kabir, Paul He, and Ondřej Lhoták. 2017. A Simple Soundness Proof for Dependent Object Types. Proc. ACM Program. Lang. 1, OOPSLA, Article 46 (Oct. 2017), 27 pages. https://doi.org/10.1145/3133870

Tiark Rompf and Nada Amin. 2016. Type Soundness for Dependent Object Types (DOT). In Proceedings of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2016). ACM, New York, NY, USA, 624–641. https://doi.org/10.1145/2983990.2984008

Examples

Works for $D_{<:}$ but not strong kernel

$$\frac{\vdash_{D_{<:}} \{A : \bot..\bot\} <: \{A : \bot..\top\} \qquad x : \{A : \bot..\bot\} \vdash_{D_{<:}} x.A <: \bot}{\vdash_{D_{<:}} \forall(x : \{A : \bot..\top\})x.A <: \forall(x : \{A : \bot..\bot\})\bot} \textsc{All}$$

This judgment is not admissible in strong kernel, because when comparing the return types, the following judgment is required:

$$(x : \{A : \bot..\top\} \vdash x.A) <: (?\bot \dashv x : \{A : \bot..\bot\})$$

Notice that on the left only $x.A <: \top$ is known so it is not admissible.

Definition of $D_{<:}$ Normal Form

$$\frac{}{\Gamma \vdash_{D_{<:}} T <: \top}\ \textsc{Top} \qquad \frac{}{\Gamma \vdash_{D_{<:}} \bot <: T}\ \textsc{Bot} \qquad \frac{}{\Gamma \vdash_{D_{<:}} T <: T}\ \textsc{Refl}$$

$$\frac{\begin{array}{c}\Gamma \vdash_{D_{<:}} S_2 <: S_1 \\ \Gamma \vdash_{D_{<:}} U_1 <: U_2\end{array}}{\Gamma \vdash_{D_{<:}} \{A : S_1..U_1\} <: \{A : S_2..U_2\}}\ \textsc{Bnd} \qquad \frac{\begin{array}{c}\Gamma \vdash_{D_{<:}} S_2 <: S_1 \\ \Gamma ; x : S_2 \vdash_{D_{<:}} U_1 <: U_2\end{array}}{\Gamma \vdash_{D_{<:}} \forall(x : S_1)U_1 <: \forall(x : S_2)U_2}\ \textsc{All}$$

$$\frac{\Gamma \vdash_{D_{<:}} \Gamma(x) <: \{A : S..\top\}}{\Gamma \vdash_{D_{<:}} S <: x.A}\ \textsc{Sel1'} \qquad \frac{\Gamma \vdash_{D_{<:}} \Gamma(x) <: \{A : \bot..U\}}{\Gamma \vdash_{D_{<:}} x.A <: U}\ \textsc{Sel2'}$$

$$\frac{\Gamma \vdash_{D_{<:}} \Gamma(x) <: \{A : S..\top\} \qquad \Gamma \vdash_{D_{<:}} \Gamma(x) <: \{A : \bot..U\} \qquad \text{(for some } x\text{)}}{\Gamma \vdash_{D_{<:}} S <: U}\ \text{SR}$$

Summary Table

Name	the ALL rule	the SR rule	Decidability
$D_{<:}$ and $D_{<:}$ normal form	full ALL	✓	undecidable
	full ALL	×	undecidable
Strong kernel $D_{<:}$	SK-ALL	×	decidable
Kernel $D_{<:}$	K-ALL	×	decidable
	K-ALL or SK-ALL	✓	unknown

One future work is to check whether kernel $D_{<:}$ + SR is decidable or not.
We don't really understand much about subtyping reflection.