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

JASON Z.S. HU, McGill University, Canada
ONDŘEJ LHOTÁK, University of Waterloo, Canada

Dependent Object Types (DOT) is a calculus with path dependent types, intersection types, and object self-references, which serves as the core calculus of Scala 3. Although the calculus has been proven sound, it remains open whether type checking in DOT is decidable. In this paper, we establish undecidability proofs of type checking and subtyping of $D_{<:}$, a syntactic subset of DOT. It turns out that even for $D_{<:}$, undecidability is surprisingly difficult to show, as evidenced by counterexamples for past attempts. To prove undecidability, we discover an equivalent definition of the $D_{<:}$ subtyping rules in normal form. Besides being easier to reason about, this definition makes the phenomenon of subtyping reflection explicit as a single inference rule. After removing this rule, we discover two decidable fragments of $D_{<:}$ subtyping and identify algorithms to decide them. We prove soundness and completeness of the algorithms with respect to the fragments, and we prove that the algorithms terminate. Our proofs are mechanized in a combination of Coq and Agda.

CCS Concepts: • **Theory of computation → Type theory**; • **Software and its engineering → Functional languages**.

Additional Key Words and Phrases: $D_{<:}$, Dependent Object Types, Undecidability, Algorithmic Typing

ACM Reference Format:
Jason Z.S. Hu and Ondřej Lhoták. 2020. Undecidability of $D_{<:}$ and Its Decidable Fragments. *Proc. ACM Program. Lang.* 4, POPL, Article 9 (January 2020), 30 pages. https://doi.org/10.1145/3371077

1 INTRODUCTION

The Dependent Object Types (DOT) calculus has received attention as a model for the Scala type system [Amin et al. 2016; Rapoport et al. 2017; Rompf and Amin 2016, etc.]. The calculus features objects with abstract type members with upper and lower bounds, and path-dependent types to select those type members. It also supports object self-references, intersection types, and dependent function types.

To implement any type system in a compiler requires a type checking *algorithm.* If type checking is undecidable, a compiler writer needs either at least a semi-algorithm or an algorithm for a decidable variant of the type system.

Type checking DOT has been conjectured to be undecidable because bounded quantification is undecidable in $F_{<:}$ [Pierce 1992]. However, such informal reasoning about DOT can understandably be incorrect, as we show with a simple example in §4.2. Formally determining decidability of DOT turns out to be surprisingly challenging. It is challenging even for $D_{<:}$, a restriction of DOT that removes self-references and intersection types, leaving type members and path-dependent types that select them [Amin et al. 2016; Amin and Rompf 2017]. In this paper, our focus is entirely on decidability of $D_{<:}$ and its variants.

Authors' addresses: Jason Z.S. Hu, School of Computer Science, McGill University, 3480 University St. Montréal, Quebec, H3A 0E9, Canada, zhong.s.hu@mail.mcgill.ca; Ondřej Lhoták, Cheriton School of Computer Science, University of Waterloo, 200 University Avenue West, Waterloo, Ontario, N2L 3G1, Canada, olhotak@uwaterloo.ca.

2475-1421/2020/1-ART9
https://doi.org/10.1145/3371077

A general technique to prove a decision problem P undecidable is *reduction* from a known undecidable problem Q. This requires ① defining a mapping M from instances of Q to instances of P and proving that p is yes-instance of P if ② and only if ③ q is a yes-instance of Q. Amin et al. [2016] does ① and ② for a reduction from $F_{<:}$ to $D_{<:}$. However, in §4.2, we identify a counterexample to ③. This means that the proposed mapping ① cannot be used to prove $D_{<:}$ undecidable.

Based on the counterexample, we define $F^{-}_{<:}$, an undecidable fragment of $F_{<:}$ that is better suited for reduction to $D_{<:}$. However, reduction is still thwarted by subtyping transitivity, which is posed as an explicit inference rule in $D_{<:}$. In $D_{<:}$, all reasoning about any subtyping relationship $S <: U$ must consider the possibility that it arose due to transitivity $S <: T <: U$ involving some arbitrary and unknown type T.

In previous work on DOT and $D_{<:}$, a recurring challenge has been the concept of *subtyping reflection*, or bad bounds in the previous literature. In the presence of a type member declaration $x : \{A : S..U\}$ with upper and lower bounds, the defining subtyping relationships $S <: x.A$ and $x.A <: U$ conspire with transitivity to induce the possibly unexpected and undesirable subtyping relationship $S <: U$ between the bounds.

For $F_{<:}$, there is a normal form of the subtyping rules that achieves transitivity without an explicit rule [Curien and Ghelli 1990; Pierce 1992]. We discover an analogous normal form for $D_{<:}$ in §4.6. In particular, we show that to achieve transitivity in $D_{<:}$ normal form, it is both necessary and sufficient to express the subtyping reflection concept as an explicit rule (SR), and add it to the obvious fundamental rules that define the meaning of each form of type. $D_{<:}$ normal form turns out to have convenient properties and becomes the core concept underlying all of our developments.

We prove undecidability of $D_{<:}$ by a reduction from $F^{-}_{<:}$ to $D_{<:}$ normal form.

In $D_{<:}$ normal form, undecidability is crisply characterized by two specific subtyping rules. The first is the All rule that compares function types, which is well known from $F_{<:}$ as the root cause of its undecidability. In $F_{<:}$, this rule can be restricted to a kernel version that applies only to functions with equal parameter types to make the resulting *kernel* $F_{<:}$ decidable. The second is the SR rule that models subtyping reflection. If the SR rule is removed from $D_{<:}$ and the All rule is replaced with the kernel version, the resulting kernel $D_{<:}$ becomes decidable.

Moreover, we show that kernel $D_{<:}$ is exactly the fragment of (full) $D_{<:}$ that can be typed by the partial typing algorithm of Nieto [2017]. Nieto identified a counterexample demonstrating that the subtyping relation implemented by the Scala compiler violates transitivity. The violation corresponds directly to the SR rule of kernel $D_{<:}$. The implementation of subtyping in the compiler does not implement this rule. This observation motivates dropping this problematic rule from practical, decidable variants of $D_{<:}$ normal form and DOT (when a normal form for DOT is found).

The kernel restriction of the All rule seriously limits expressiveness in $D_{<:}$ because it prevents comparison between parameter types of functions. This disables the case in which the parameter types are type aliases of each other. For example, in the scope of a type member declaration $x : \{A : T..T\}$, the types $x.A$ and T should be considered equivalent. To address this limitation, we define a *strong kernel* variant of the All rule that allows comparison between parameter types. The expressiveness of strong kernel $D_{<:}$ is strictly between kernel $D_{<:}$ and full $D_{<:}$, but unlike full $D_{<:}$, strong kernel $D_{<:}$ is decidable. Finally, we provide stare-at subtyping, an algorithm to decide subtyping in strong kernel $D_{<:}$.

To summarize, our contributions are:

(1) a counterexample to the previously proposed reduction from $F_{<:}$ to $D_{<:}$,
(2) $D_{<:}$ normal form and its equivalence to $D_{<:}$,
(3) undecidability of $D_{<:}$ by reduction from $F^{-}_{<:}$,

Table 1. Summary of $D_{<:}$ variants

Name	ALL rule	SR rule	Decidability
$D_{<:}$ and $D_{<:}$ normal form	full ALL	✓	undecidable (§4.6)
	full ALL	×	undecidable (§4.6)
Strong kernel $D_{<:}$	SK-ALL	×	decidable by Stare-at subtyping (§6.3)
Kernel $D_{<:}$	kernel K-ALL	×	decidable by Step subtyping (§5.2)
	K-ALL or SK-ALL	✓	unknown

(4) equivalence of kernel $D_{<:}$ and the fragment of $D_{<:}$ typeable by Nieto's algorithm,
(5) strong kernel $D_{<:}$,
(6) the stare-at algorithm for deciding subtyping of strong kernel $D_{<:}$, and
(7) strong kernel and stare-at subtyping for $F_{<:}$.

We have verified the proofs of our lemmas and theorems using proof assistants. The proofs of undecidability are mechanized in Agda. The proofs of equivalence of the decidable variants of the calculi and their decision algorithms are mechanized in Coq. There are no dependencies between the Agda and Coq formalizations.

The properties of the variants of $D_{<:}$ are summarized in Table 1.

2 PRELIMINARIES

We adopt the following conventions throughout the paper.

Throughout this paper, we consider two types or terms the same if they are equivalent up to α-conversion [Barendregt 1985]. We use subscripts to emphasize free occurrences of a variable. For example, T_x means x may have free occurrences in T.

We use semicolons (;) to denote context concatenation instead of commas (,).

DEFINITION 1. *A type T is closed w.r.t. a context Γ, if $fv(T) \subseteq dom(\Gamma)$.*

DEFINITION 2. *Well-formedness of a context is inductively defined as follows.*

(1) The empty context $\bullet$ is well-formed.
(2) If Γ is well-formed, T is closed w.r.t. Γ and $x \notin dom(\Gamma)$, then $\Gamma; x : T$ is well-formed.

Unless explicitly mentioned, all lemmas and theorems require and preserve that types are closed and contexts are well-formed. This is proven explicitly in the mechanized proofs.

3 DEFINITIONS OF $F_{<:}$ AND $D_{<:}$

$F_{<:}$ is introduced by Curien and Ghelli [1990] as an extension of system F with upper bounded quantification to study its coherence property.

DEFINITION 3. *$F_{<:}$ is defined in Figure 1.*

Universal types in $F_{<:}$ combine polymorphism and subtyping. Universal types can be compared by the F-ALL rule. The F-TRANS rule indicates that the system has transitivity. It turns out that $F_{<:}$ can be defined in a way such that transitivity does not appear as an inference rule but rather a provable property.

DEFINITION 4. *$F_{<:}$ normal form is defined in Figure 2. The rules that differ are shaded.*

We call this alternative definition "normal form", following the convention in Pierce [1992]. The two definitions are equivalent:

X, Y, Z	**Type variable**
$S, T, U ::=$	**Type**
$\top$	top type
X	type variable
$S \to U$	function
$\forall X <: S.U_X$	universal type

$$\frac{}{\Gamma \vdash_{F_{<:}} T <: \top}\ \text{F-Top} \qquad \frac{}{\Gamma \vdash_{F_{<:}} T <: T}\ \text{F-Refl}$$

$$\frac{X <: T \in \Gamma}{\Gamma \vdash_{F_{<:}} X <: T}\ \text{F-TVar}$$

$$\frac{\Gamma \vdash_{F_{<:}} S' <: S \qquad \Gamma \vdash_{F_{<:}} U <: U'}{\Gamma \vdash_{F_{<:}} S \to U <: S' \to U'}\ \text{F-Fun}$$

$$\frac{\Gamma \vdash_{F_{<:}} S' <: S \qquad \Gamma ; X <: S' \vdash_{F_{<:}} U <: U'}{\Gamma \vdash_{F_{<:}} (\forall X <: S.U) <: (\forall X <: S'.U')}\ \text{F-All} \qquad \frac{\Gamma \vdash_{F_{<:}} S <: T \qquad \Gamma \vdash_{F_{<:}} T <: U}{\Gamma \vdash_{F_{<:}} S <: U}\ \text{F-Trans}$$

Fig. 1. Definition of subtyping in $F_{<:}$ [Pierce 2002, Figure 26-2]

$$\frac{}{\Gamma \vdash_{F_{<:}} T <: \top}\ \text{F-Top} \qquad \boxed{\frac{}{\Gamma \vdash_{F_{<:}} X <: X}\ \text{F-VarRefl}} \qquad \boxed{\frac{X <: T \in \Gamma \qquad \Gamma \vdash_{F_{<:}} T <: U}{\Gamma \vdash_{F_{<:}} X <: U}\ \text{F-TVar'}}$$

$$\frac{\Gamma \vdash_{F_{<:}} S' <: S \qquad \Gamma \vdash_{F_{<:}} U <: U'}{\Gamma \vdash_{F_{<:}} S \to U <: S' \to U'}\ \text{F-Fun} \qquad \frac{\Gamma \vdash_{F_{<:}} S' <: S \qquad \Gamma ; X <: S' \vdash_{F_{<:}} U <: U'}{\Gamma \vdash_{F_{<:}} (\forall X <: S.U) <: (\forall X <: S'.U')}\ \text{F-All}$$

Fig. 2. Definition of $F_{<:}$ normal form

Theorem 1. *[Curien and Ghelli 1990] $F_{<:}$ subtyping is equivalent to $F_{<:}$ normal form. Namely $\Gamma \vdash_{F_{<:}} S <: U$ holds in non-normal form, iff it holds in normal form.*

$D_{<:}$ is a richer calculus than $F_{<:}$. It adds a form of dependent types, called path types, each of which has both upper bounds and lower bounds, so it is more general than $F_{<:}$.

Definition 5. *$D_{<:}$ is defined in Figure 3.*

$D_{<:}$ has the following types: the top type $\top$, the bottom type $\bot$, type declarations, path types, and dependent function types. In $D_{<:}$, a path type has the form $x.A$ where the type label A is fixed. That is, in $D_{<:}$, there is only one type label and it is A. A term in $D_{<:}$ can be a variable, a type tag, a lambda abstraction, an application, or a let binding.

In the typing rules, the Var, Sub and Let rules are standard. The All-I rule says a lambda is typed by pushing its declared parameter type to the context. Note that the return type is allowed to depend on the parameter, which makes the system dependently typed. The All-E rule types a function application. Since U may depend on its parameter, the overall type may refer to y. The Typ-I rule assigns a type declaration with equal bounds to a type tag.

In the subtyping rules, the Top, Bot, Refl and Trans rules are standard. In the Bnd rule, type declarations are compared by comparing their corresponding components. Notice that the lower bounds are in contravariant position and hence they are compared in reversed order. Similarly, the All rule also compares parameter types in reversed order. The return types are compared with the context extended with S_2. The Sel1 and Sel2 rules are used to access the bounds of a path type.

x, y, z	**Variable**	$v ::=$	**Value**
$S, T, U ::=$	**Type**	$\{A = T\}$	type tag
$\top$	top type	$\lambda(x : T)t_x$	lambda
$\bot$	bottom type	$s, t, u ::=$	**Term**
$\{A : S..U\}$	type declaration	x	variable
$x.A$	path type	v	value
$\forall(x : S)U_x$	function	$x\ y$	application
		$\text{let } x = t \text{ in } u_x$	let binding

Type Assignment

$$\frac{}{\Gamma \vdash_{D_{<:}} x : \Gamma(x)}\ \textsc{Var} \qquad \frac{\Gamma \vdash_{D_{<:}} t : S \qquad \Gamma \vdash_{D_{<:}} S <: U}{\Gamma \vdash_{D_{<:}} t : U}\ \textsc{Sub} \qquad \frac{\Gamma; x : S \vdash_{D_{<:}} t : U_x}{\Gamma \vdash_{D_{<:}} \lambda(x : S)t : \forall(x : S)U_x}\ \textsc{All-I}$$

$$\frac{\Gamma \vdash_{D_{<:}} x : \forall(z : S)U_z \quad \Gamma \vdash_{D_{<:}} y : S}{\Gamma \vdash_{D_{<:}} x\ y : U_y}\ \textsc{All-E} \qquad \frac{}{\Gamma \vdash_{D_{<:}} \{A = T\} : \{A : T..T\}}\ \textsc{Typ-I} \qquad \frac{\Gamma \vdash_{D_{<:}} t : S \qquad x \notin fv(U) \quad \Gamma; x : S \vdash_{D_{<:}} u : U}{\Gamma \vdash_{D_{<:}} \text{let } x = t \text{ in } u : U}\ \textsc{Let}$$

Subtyping

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

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

$$\frac{\Gamma \vdash_{D_{<:}} x : \{A : S..U\}}{\Gamma \vdash_{D_{<:}} S <: x.A}\ \textsc{Sel1} \qquad \frac{\Gamma \vdash_{D_{<:}} x : \{A : S..U\}}{\Gamma \vdash_{D_{<:}} x.A <: U}\ \textsc{Sel2} \qquad \frac{\Gamma \vdash_{D_{<:}} S <: T \qquad \Gamma \vdash_{D_{<:}} T <: U}{\Gamma \vdash_{D_{<:}} S <: U}\ \textsc{Trans}$$

Fig. 3. Definition of $D_{<:}$ [Amin et al. 2016]

Notice that the typing and subtyping rules in $D_{<:}$ are mutually dependent. This is because the Sub rule uses subtyping and the Sel1 and Sel2 rules use typing in their premises. This mutual dependency makes $D_{<:}$ harder to reason about. Nonetheless, this mutual dependency can be eliminated due to the following lemma.

Lemma 2. *(unravelling of $D_{<:}$ subtyping) Sel1 and Sel2 can be changed to the following rules, and the resulting subtyping relation is equivalent to the original one.*

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

Proof. This follows directly from the fact that the only typing rules that apply to variables are the Var and Sub rules. ◻

This new definition of subtyping with the Sel1' and Sel2' rules no longer depends on typing. We will use this definition in the rest of the paper.

4 UNDECIDABILITY OF $D_{<:}$ (SUB)TYPING

4.1 Definition of Undecidability

A common method for proving a decision problem undecidable is by *reduction* from some other known undecidable problem.

Definition 6. *[Martin 1997, Definition 12.1a] If Q and P are decision problems, we say Q is reducible to P ($Q \leq P$) if there is an algorithmic procedure F that allows us, given an arbitrary instance I_1 of Q, to find an instance $F(I_1)$ of P so that for every I_1, I_1 is a yes-instance of Q if and only if $F(I_1)$ is a yes-instance of P.*

Notice that reducibility requires an if and only if proof: for our choice of F, we must show that I_1 is a yes-instance of Q *if and only if* $F(I_1)$ is a yes-instance of P.

Reduction can be understood intuitively as an adversarial game. Consider a target decision problem P. Merlin is a wizard who claims to have access to true magic, and therefore be able to decide P. He is so confident that he would also offer a complete proof accompanying each yes answer he gives. Sherlock is a skeptical detective. He questions the Merlin's ability, and comes up with the following scheme in order to disprove Merlin's claim.

$$\begin{tikzcd}
\text{an instance of Q} \arrow[r, "\text{Step 1}"] & \text{an instance of P} \arrow[dr] & \\
& & \text{Merlin} \arrow[dl] \\
\text{a proof of Q} & \text{a proof of P} \arrow[l, "\text{Step 2}"'] &
\end{tikzcd}$$

Sherlock selects some undecidable problem Q. As Step 1, Sherlock devises a mapping from instances of Q to instances of P that preserves yes-instances: every yes-instance of Q maps to some yes-instance of P. As Step 2, Sherlock devises a mapping from (yes-)proofs of P to (yes-)proofs of Q. Then, if Merlin could really decide P, then Sherlock could use this setup to decide Q, which is impossible. For any instance of Q, Sherlock would map it to an instance of P and give it to Merlin to decide. If the instance of P is a no-instance, then so was the instance of Q. If the instance of P is a yes-instance, then Sherlock could map Merlin's proof into a proof that the instance of Q is also a yes-instance. P is undecidable if and only if Sherlock achieves both steps and therefore proves Merlin is wrong.

4.2 The Partial Undecidability Proof of Amin et al. [2016]

Subtyping in $F_{<:}$ is known to be undecidable [Pierce 1992]. Amin et al. [2016] defined the following total mappings from types and contexts in $F_{<:}$ to types and contexts in $D_{<:}$:

Definition 7. *[Amin et al. 2016] The mappings $[\![\cdot]\!]$ and $\langle\!\langle\cdot\rangle\!\rangle$ are defined as follows:*

$$\begin{aligned}
[\![\top]\!] &= \top \\
[\![X]\!] &= x_X.A \\
[\![S \to U]\!] &= \forall(x : [\![S]\!])[\![U]\!] && \text{(function case)} \\
[\![\forall X <: S.U]\!] &= \forall(x_X : \{A : \bot..[\![S]\!]\})[\![U]\!]
\end{aligned}$$

$$\begin{aligned}
\langle\!\langle\bullet\rangle\!\rangle &= \bullet \\
\langle\!\langle\Gamma; X <: T\rangle\!\rangle &= \langle\!\langle\Gamma\rangle\!\rangle; x_X : \{A : \bot..[\![T]\!]\}
\end{aligned}$$

In the mapping, a correspondence between type variables in $F_{<:}$ and variables in $D_{<:}$ is assumed, as indicated by the notation x_X. Amin et al. also proved that given a yes-instance of subtyping in $F_{<:}$, its image under the mapping is also a yes-instance of subtyping in $D_{<:}$:

Theorem 3. *[Amin et al. 2016, Theorem 1] If* $\Gamma \vdash_{F_{<:}} S <: U$, *then* $\langle\!\langle \Gamma \rangle\!\rangle \vdash_{D_{<:}} [\![S]\!] <: [\![U]\!]$.

According to Definition 6, to show subtyping in $D_{<:}$ undecidable, it remains to show the other direction:

Conjecture 4. *If* $\langle\!\langle \Gamma \rangle\!\rangle \vdash_{D_{<:}} [\![S]\!] <: [\![U]\!]$, *then* $\Gamma \vdash_{F_{<:}} S <: U$.

To see why this step is essential, consider what would happen if we defined a new calculus $D^{+}_{<:}$ by extending $D_{<:}$ subtyping with a rule that makes every type S a subtype of every type U:

$$\frac{}{\Gamma \vdash_{D^{+}_{<:}} S <: U}\ \textsc{Trivial}$$

The mapping in Definition 7 and Theorem 3 continue to hold even for $D^{+}_{<:}$. But subtyping in $D^{+}_{<:}$ is obviously decidable because every instance is a yes-instance. If Theorem 3 were sufficient to prove undecidability of $D_{<:}$, then it would also be sufficient to "prove" undecidability of the obviously decidable $D^{+}_{<:}$. Thus, Conjecture 4 is *essential* to complete the proof of undecidability of $D_{<:}$ subtyping.

Unfortunately, Conjecture 4 is *false*. As a counterexample, consider the following subtyping query in $F_{<:}$:

$$\vdash_{F_{<:}} \top \to \top <: (\forall X <: \top.\top)$$

This subtyping relationship is false: in $F_{<:}$, function types and universal types are not related by subtyping. The image of this subtyping relationship under the mapping is:

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

In $D_{<:}$, this subtyping relationship is true, as witnessed by the following derivation tree.

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

The counterexample shows that reduction from $F_{<:}$ via the mapping in Definition 7 *cannot* be used to prove undecidability of $D_{<:}$ subtyping.

4.3 $F^{-}_{<:}$

The counterexample suggests that the problem is that the mapping permits interference between function types and universal types in $F_{<:}$ because it maps both of them to dependent function types in $D_{<:}$. Reviewing Pierce [1992], we notice that the undecidability proof of $F_{<:}$ does not make use of function types. Therefore, we can remove function types from $F_{<:}$ to obtain a simpler calculus that is better suited for undecidability reductions.

Definition 8. *$F^{-}_{<:}$ is obtained from $F_{<:}$ defined in Figure 2 by removing function types ($\to$) and the F-Fun rule.*

Theorem 5. *$F^{-}_{<:}$ subtyping is undecidable.*

Proof. The $F_{<:}$ undecidability proof of Pierce [1992] does not depend on function types. □

The mappings from Definition 7 can be applied to types and contexts in $F^{-}_{<:}$. The function case can be removed from the mapping since $F^{-}_{<:}$ does not have function types.

4.4 Subtyping Reflection

Since $F^{-}_{<:}$ invalidates the counterexample to Conjecture 4, we can attempt to prove the conjecture for $F^{-}_{<:}$. When we try to invert the premise of the conjecture, $\langle\!\langle \Gamma \rangle\!\rangle \vdash_{D_{<:}} [\![S]\!] <: [\![U]\!]$, the first problem we encounter is *subtyping reflection*. The pattern of subtyping reflection is discussed and known as *bad bounds* in Rapoport et al. [2017]; Rompf and Amin [2016]. Subtyping reflection is an unintended consequence of the combination of the SEL1', SEL2' and TRANS rules. In certain typing contexts, subtyping reflection makes it possible to prove subtyping between *any* types S and U. Consider the following derivation tree:

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

$$\dfrac{\dfrac{\dfrac{}{\Gamma \vdash_{D_{<:}} \{A : S..U\} : \{A : S..\top\}}\ \textsc{Bnd}}{\Gamma \vdash_{D_{<:}} S <: x.A}\ \textsc{Sel1'} \qquad \dfrac{\text{similar to the left}}{\Gamma \vdash_{D_{<:}} x.A <: U}\ \textsc{Sel2'}}{\Gamma \vdash_{D_{<:}} S <: U}\ \textsc{Trans}$$

This derivation uses transitivity to connect the lower and upper bounds of the path type $x.A$. The types S and U can be any types at all, as long as they appear in the type of x in the typing context Γ.

In $F_{<:}$, on the other hand, it is easy to show that, for example, a supertype of $\top$ must be $\top$. Properties like this are called *inversion properties*. These properties do not hold in general in $D_{<:}$ due to subtyping reflection. Fortunately, we can prove similar properties in $D_{<:}$ if we restrict the typing context Γ according to the following definition:

DEFINITION 9. *(invertible context) A context Γ in $D_{<:}$ is invertible if all of the following hold.*

(1) No variable is assigned the type $\bot$,
(2) No variable is assigned a type of the form $\{A : S..\bot\}$ for any S,
(3) No variable is assigned a type of the form $\{A : T..\{A : S..U\}\}$ for any T, S and U, and
(4) If a variable is assigned a type of the form $\{A : S..U\}$, then $S = \bot$.

The contexts in the range of the mapping from Definition 7 are all invertible:

Lemma 6. *Given an $F^{-}_{<:}$ context Γ, $\langle\!\langle \Gamma \rangle\!\rangle$ is invertible.*

In invertible contexts, we can prove many useful inversion properties:

Lemma 7. *(supertypes in invertible contexts) If a context Γ is invertible, then all of the following hold.*

(1) If $\Gamma \vdash_{D_{<:}} \top <: T$, then $T = \top$.
(2) If $\Gamma \vdash_{D_{<:}} \{A : S..U\} <: T$, then $T = \top$ or T has the form $\{A : S'..U'\}$.
(3) If $\Gamma \vdash_{D_{<:}} \forall(x : S)U <: T$, then $T = \top$ or T has the form $\forall(x : S')U'$.

Lemma 8. *(subtypes in invertible contexts) If a context Γ is invertible, then all of the following hold.*

(1) If $\Gamma \vdash_{D_{<:}} T <: \bot$, then $T = \bot$.
(2) If $\Gamma \vdash_{D_{<:}} T <: \{A : S..U\}$, then $T = \bot$ or T has the form $\{A : S'..U'\}$.
(3) If $\Gamma \vdash_{D_{<:}} T <: \forall(x : S)U$, then $T = \bot$ or T is some path $y.A$, or T has the form $\forall(x : S')U'$.

Lemma 9. *(subtyping inversion) If a context Γ is invertible, then the following hold.*

(1) If $\Gamma \vdash_{D_{<:}} \{A : S_1..U_1\} <: \{A : S_2..U_2\}$, then $\Gamma \vdash_{D_{<:}} S_2 <: S_1$ and $\Gamma \vdash_{D_{<:}} U_1 <: U_2$.
(2) If $\Gamma \vdash_{D_{<:}} \forall(x : S_1)U_1 <: \forall(x : S_2)U_2$, then $\Gamma \vdash_{D_{<:}} S_2 <: S_1$ and $\Gamma; x : S_2 \vdash_{D_{<:}} U_1 <: U_2$.

These lemmas show that $D_{<:}$ is getting much closer to $F^{-}_{<:}$ in invertible contexts (hence in the contexts in the range of $\langle\!\langle \cdot \rangle\!\rangle$) and suggest that we are just one step away from proving undecidability of $D_{<:}$. Unfortunately, there is one more problem.

4.5 The Trans Rule

Recall the conjecture that we are trying to prove: if $\langle\!\langle \Gamma \rangle\!\rangle \vdash_{D_{<:}} [\![S]\!] <: [\![U]\!]$, then $\Gamma \vdash_{F^-_{<:}} S <: U$. When we perform induction on the premise, in the case of the Trans rule, we have the following antecedents in the proof context:

(1) For some T, $\langle\!\langle \Gamma \rangle\!\rangle \vdash_{D_{<:}} [\![S]\!] <: T$,
(2) $\langle\!\langle \Gamma \rangle\!\rangle \vdash_{D_{<:}} T <: [\![U]\!]$,
(3) Inductive hypothesis: if $\langle\!\langle \Gamma \rangle\!\rangle \vdash_{D_{<:}} [\![T_1]\!] <: [\![T_2]\!]$, then $\Gamma \vdash_{F^-_{<:}} T_1 <: T_2$.

The problem is that T is not necessarily in the range of $[\![\ ]\!]$.

A counterexample for the Trans case: Define $x <: T$ as syntactic sugar for $x : \{A : \bot..T\}$. The following is a proof that Merlin gives us to prove that $\vdash_{D_{<:}} \forall(x <: \top)\top <: \top$:

$$\dfrac{\dfrac{\mathcal{D}}{\vdash_{D_{<:}} \forall(x <: \top)\top <: \forall(x : \{A : \top..\bot\})x.A} \qquad \dfrac{}{\vdash_{D_{<:}} \forall(x : \{A : \top..\bot\})x.A <: \top}\ \textsc{Top}}{\vdash_{D_{<:}} \forall(x <: \top)\top <: \top}\ \textsc{Trans}$$

In the above, the subderivation $\mathcal{D}$ is as shown below.

$$\dfrac{\dfrac{\text{straightforward}}{\vdash_{D_{<:}} \{A : \top..\bot\} <: \{A : \bot..\top\}}\ \textsc{Bnd} \qquad \dfrac{\text{straightforward}}{x : \{A : \top..\bot\} \vdash_{D_{<:}} \top <: x.A}\ \textsc{Sel1}}{\vdash_{D_{<:}} \forall(x <: \top)\top <: \forall(x : \{A : \top..\bot\})x.A}\ \textsc{All}$$

In this example, the proof of $\vdash_{D_{<:}} \forall(x <: \top)\top <: \top$ is concluded by transitivity on $\forall(x : \{A : \top..\bot\})x.A$. An inspection shows that both $\forall(x <: \top)\top$ and $\top$ are in the image of $[\![\cdot]\!]$, but $\forall(x : \{A : \top..\bot\})x.A$ is not, as it would require at the very least the lower bound in the type declaration to be $\bot$. Therefore, the target theorem cannot be proven by induction, because the induction hypothesis can be applied only to types in the range of $[\![\cdot]\!]$. To resolve this issue, we need to reformulate $D_{<:}$ so that it does not use the Trans rule.

4.6 $D_{<:}$ Normal Form

Although $F_{<:}$ also has a F-Trans rule, it does not cause any problems for the undecidability proof of Pierce [1992]. The reason is that the paper begins with $F_{<:}$ normal form, a formulation that defines the same calculus but does not use the F-Trans rule. Therefore, it is interesting to ask whether there is $D_{<:}$ normal form. We first define what we mean by a normal form.

Definition 10. *A subtyping definition is in normal form if it satisfies the* subformula property, *which requires that the premises of every rule are defined in terms of syntactic subterms of the conclusion.*

The subterms of the conclusion $\Gamma \vdash S <: U$ include subterms of both S and U, as well as of the context Γ. Consider the following rule in $F_{<:}$ normal form.

$$\dfrac{X <: T \in \Gamma \qquad \Gamma \vdash_{F_{<:}} T <: U}{\Gamma \vdash_{F_{<:}} X <: U}\ \text{F-}\textsc{Tvar'}$$

Although T is not found in X or U, notice that T is a result of context lookup of X and is therefore a subterm of the context Γ.

Consider the F-Trans rule in $F_{<:}$.

$$\dfrac{\Gamma \vdash_{F_{<:}} S <: T \qquad \Gamma \vdash_{F_{<:}} T <: U}{\Gamma \vdash_{F_{<:}} S <: U}\ \text{F-}\textsc{Trans}$$

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

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

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

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

Fig. 4. Definition of subtyping of $D_{<:}$ normal form

In this rule, T could be arbitrary and is unrelated to the inputs. Therefore, a definition in normal form should not contain rules like this.

We have discovered a reformulation of the $D_{<:}$ subtyping relation that is in normal form. The normal form subtyping rules are shown in Figure 4. The difference from the original $D_{<:}$ rules is that the Trans rule is removed and replaced by the new SR rule. By inspecting the rules one by one, we can see that they are indeed in normal form. We must also check that the normal form rules define the same subtyping relation as the original $D_{<:}$ subtyping rules. As a first step, we will show that $D_{<:}$ normal form satisfies transitivity.

Proving transitivity of $D_{<:}$ normal form is quite tricky. First, transitivity is interdependent with narrowing, so we will need to prove the two together. Second, the proof of transitivity requires reasoning about types of the following form:

$$\{A : T_1..\{A : T_2..\{A : T_3.. \cdots \{A : T_n..T\} \cdots \}\}\}$$

We define such types formally as follows:

Definition 11. *A type declaration hierarchy is a type $tdh(l, T)$ defined by a list of types l and another type T inductively as follows.*

$$tdh(l, T) = \begin{cases} T, \text{ if } l = nil, \text{ or} \\ \{A : T'..tdh(l', T)\}, \text{ if } l = T' :: l' \end{cases}$$

With that definition, we can now state and prove the full transitivity and narrowing theorem:

Theorem 10. *For any type T and two subtyping derivations $\mathcal{D}_1$ and $\mathcal{D}_2$, the following hold:*

(1) *(transitivity) If $\mathcal{D}_1$ concludes $\Gamma \vdash_{D_{<:}} S <: T$ and $\mathcal{D}_2$ concludes $\Gamma \vdash_{D_{<:}} T <: U$, then $\Gamma \vdash_{D_{<:}} S <: U$.*
(2) *(narrowing) If $\mathcal{D}_1$ concludes $\Gamma \vdash_{D_{<:}} S <: T$ and $\mathcal{D}_2$ concludes $\Gamma; x : T; \Gamma' \vdash_{D_{<:}} S' <: U'$, then $\Gamma; x : S; \Gamma' \vdash_{D_{<:}} S' <: U'$.*
(3) *If $\mathcal{D}_1$ concludes $\Gamma \vdash_{D_{<:}} T' <: tdh(l, \{A : S'..T\})$ and $\mathcal{D}_2$ concludes $\Gamma \vdash_{D_{<:}} T <: U$, then $\Gamma \vdash_{D_{<:}} T' <: tdh(l, \{A : S'..U\})$.*
(4) *If $\mathcal{D}_1$ concludes $\Gamma \vdash_{D_{<:}} S <: T$ and $\mathcal{D}_2$ concludes $\Gamma \vdash_{D_{<:}} T' <: tdh(l, \{A : T..U'\})$, then $\Gamma \vdash_{D_{<:}} T' <: tdh(l, \{A : S..U'\})$.*

All derivations involved are in $D_{<:}$ normal form.

Proof. The proof is done by induction on the lexicographical order of the structure of the triple $(T, \mathcal{D}_1, \mathcal{D}_2)$. That is, the inductive hypotheses of the theorem are:

(a) If T^* is a strict syntactic subterm of T, then the theorem holds for T^* and any other two subtyping derivations $\mathcal{D}_1'$ and $\mathcal{D}_2'$.
(b) If $\mathcal{D}_1^*$ is a strict subderivation of $\mathcal{D}_1$, then the theorem holds for the same type T, the subderivation $\mathcal{D}_1^*$ and any subtyping derivation $\mathcal{D}_2'$.
(c) If $\mathcal{D}_2^*$ is a strict subderivation of $\mathcal{D}_2$, then the theorem holds for the same type T, the same derivation $\mathcal{D}_1$ and the subderivation $\mathcal{D}_2^*$.

This form of induction is motivated by the dependencies between the four clauses of the theorem and can be found in other literature [Pfenning 2000, Theorem 5 (Cut)]. Specifically, (a) addresses that transitivity (1) and narrowing (2) are mutually dependent, but when transitivity uses narrowing, T is replaced with a syntactic subterm T^*. Similarly, (b) addresses that transitivity (1) and (3) are mutually dependent, but in each dependence cycle, $\mathcal{D}_1$ is replaced with a subderivation $\mathcal{D}_1^*$. Finally, (c) addresses that transitivity (1) and (4) are mutually dependent, but in each dependence cycle, $\mathcal{D}_2$ is replaced with a subderivation $\mathcal{D}_2^*$.

In proving transitivity (1), we consider the cases by which $\Gamma \vdash_{D_{<:}} S <: T$ and $\Gamma \vdash_{D_{<:}} T <: U$ are derived. We consider three cases in detail:

All-All case: In this case, S, T and U are all dependent function types. Let $S = \forall(x : S_1)U_1$, $T = \forall(x : S_2)U_2$ and $U = \forall(x : S_3)U_3$. The antecedents are:

i. $\Gamma \vdash_{D_{<:}} S_2 <: S_1$,
ii. $\Gamma \vdash_{D_{<:}} S_3 <: S_2$,
iii. $\Gamma; x : S_2 \vdash_{D_{<:}} U_1 <: U_2$, and
iv. $\Gamma; x : S_3 \vdash_{D_{<:}} U_2 <: U_3$.

The goal is to show $\Gamma \vdash_{D_{<:}} \forall(x : S_1)U_1 <: \forall(x : S_3)U_3$ by All, which requires $\Gamma \vdash_{D_{<:}} S_3 <: S_1$ and $\Gamma; x : S_3 \vdash_{D_{<:}} U_1 <: U_3$. Applying inductive hypothesis (a) to ii. and i., we obtain $\Gamma \vdash_{D_{<:}} S_3 <: S_1$ via transitivity (1). Again applying inductive hypothesis (a) to ii. and iii., we obtain $\Gamma; x : S_3 \vdash_{D_{<:}} U_1 <: U_2$ via narrowing (2). Together with inductive hypothesis (a) and iv., $\Gamma; x : S_3 \vdash_{D_{<:}} U_1 <: U_3$ is concluded via transitivity (1) and the goal is also concluded.

Sel1'-Sel2' case: In this case, we know $T = x.A$ for some x. The antecedents are:

i. $\Gamma \vdash_{D_{<:}} \Gamma(x) <: \{A : S..\top\}$ and
ii. $\Gamma \vdash_{D_{<:}} \Gamma(x) <: \{A : \bot..U\}$.

By the SR rule, we can show the conclusion $\Gamma \vdash_{D_{<:}} S <: U$. That is, the SR rule is a restricted form of transitivity for the case when the middle type is a path type $x.A$.

Sel2'-*any* case: When $\Gamma \vdash_{D_{<:}} S <: T$ is derived by Sel2', we know $S = y.A$ for some y. The antecedents are:

i. $\Gamma \vdash_{D_{<:}} \Gamma(y) <: \{A : \bot..T\}$, and
ii. $\Gamma \vdash_{D_{<:}} T <: U$.

The intention is to show that $\Gamma \vdash_{D_{<:}} \Gamma(y) <: \{A : \bot..U\}$ holds and hence conclude $\Gamma \vdash_{D_{<:}} y.A <: U$ by Sel2'. To derive this conclusion, we need to apply the induction hypothesis (b) with $\Gamma \vdash_{D_{<:}} \Gamma(y) <: \{A : \bot..T\}$ as the subderivation $\mathcal{D}_1^*$. The induction hypothesis (b) provides the necessary $\Gamma \vdash_{D_{<:}} \Gamma(y) <: \{A : \bot..U\}$ via clause (3), and hence $\Gamma \vdash_{D_{<:}} y.A <: U$. The SR-*any* case can be proved in the same way. The *any*-Sel1' and *any*-SR cases can be proved in a symmetric way, by invoking inductive hypothesis (c) instead of inductive hypothesis (b) in the corresponding places.

Narrowing (2) is proved by case analysis on the derivation of $\Gamma; x : T; \Gamma' \vdash_{D_{<:}} S' <: U'$. Several cases require transitivity, which is obtained by applying the induction hypothesis (c).

Clause (3) of the theorem is proved by case analysis on $\mathcal{D}_1$, the derivation of $\Gamma \vdash_{D_{<:}} T' <: \text{tdh}(l, \{A : S'..T\})$, and then by an inner induction on the list l. We discuss two interesting cases.

Bnd-nil case: $\text{tdh}(nil, \{A : S'..T\}) = \{A : S'..T\}$ and $\Gamma \vdash_{D_{<:}} T' <: \text{tdh}(nil, \{A : S'..T\})$ is constructed by Bnd. From the Bnd rule, we know that $T' = \{A : S_0..U_0\}$ and have the following antecedents:

i. $\Gamma \vdash_{D_{<:}} S' <: S_0$, and
ii. $\Gamma \vdash_{D_{<:}} U_0 <: T$, and
iii. $\Gamma \vdash_{D_{<:}} T <: U$.

We wish to apply transitivity (1) to antecedents ii. and iii. to obtain $\Gamma \vdash_{D_{<:}} U_0 <: U$. We can do this by invoking the induction hypothesis (b) with the antecedent ii. $\Gamma \vdash_{D_{<:}} U_0 <: T$ as $\mathcal{D}_1^*$. After applying transitivity, we can apply Bnd to $\Gamma \vdash_{D_{<:}} S' <: S_0$ and $\Gamma \vdash_{D_{<:}} U_0 <: U$ to obtain $\Gamma \vdash_{D_{<:}} \{A : S_0..U_0\} <: \{A : S'..U\}$ as required. This case shows the mutual dependence between clause (3) and transitivity (1).

Sel2'-*any* case: In this case, we know that $T' = z.A$ for some z and have the following antecedents:

i. $\Gamma \vdash_{D_{<:}} \Gamma(z) <: \{A : \bot..\text{tdh}(l, \{A : S'..T\})\}$, and
ii. $\Gamma \vdash_{D_{<:}} T <: U$.

We apply the induction hypothesis (b) with the subderivation $\Gamma \vdash_{D_{<:}} \Gamma(z) <: \{A : \bot..\text{tdh}(l, \{A : S'..T\})\}$ as $\mathcal{D}_1^*$. Notice that $\{A : \bot..\text{tdh}(l, \{A : S'..T\})\}$ can be rewritten as $\text{tdh}(\bot :: l, \{A : S'..T\})$, so the induction hypothesis of (3) applies to yield $\Gamma \vdash_{D_{<:}} \Gamma(z) <: \text{tdh}(\bot :: l, \{A : S'..\,U\,\})$, which can be rewritten as $\Gamma \vdash_{D_{<:}} \Gamma(z) <: \{A : \bot..\text{tdh}(l, \{A : S'..U\})\}$. Finally, by Sel2', $\Gamma \vdash_{D_{<:}} z.A <: \text{tdh}(l, \{A : S'..U\})$ as required. Since the list $\bot :: l$ is longer than l, this case shows why clause (3) needs to be defined on type declaration hierarchies of non-empty lists.

Clause (4) of the theorem is dual to clause (3) and is proven in a symmetric way. Instead of the inductive hypothesis (b), clause (4) uses the inductive hypothesis (c). □

Once transitivity is proved, we can show that the two definitions of $D_{<:}$ subtyping are equivalent.

Theorem 11. *Subtyping in $D_{<:}$ normal form is equivalent to the original $D_{<:}$.*

Proof. The if direction is immediate. In the only if direction, the Trans case can be discharged by transitivity of $D_{<:}$ normal form. □

Now that we have $D_{<:}$ normal form, we can finally prove Conjecture 4 for $F^-_{<:}$, which leads to the proof of undecidability of $D_{<:}$ subtyping.

Theorem 12. *If $\langle\!\langle \Gamma \rangle\!\rangle \vdash_{D_{<:}} [\![S]\!] <: [\![U]\!]$, then $\Gamma \vdash_{F^-_{<:}} S <: U$.*

Proof. The proof is by induction on the subtyping derivation in $D_{<:}$ normal form, which no longer has the problem with the Trans rule discussed in §4.5. Most of the cases are proved by straightforward application of the induction hypothesis. The Sel1' and SR cases require the following argument. In both cases, we have the antecedent:

$$\text{for some } x, \langle\!\langle \Gamma \rangle\!\rangle \vdash_{D_{<:}} \langle\!\langle \Gamma \rangle\!\rangle(x) <: \{A : [\![S]\!]..\top\}$$

By inspecting $\langle\!\langle \cdot \rangle\!\rangle$, we know that $\langle\!\langle \Gamma \rangle\!\rangle(x)$ must be $\{A : \bot..T\}$ for some T, and therefore the antecedent becomes

$$\langle\!\langle \Gamma \rangle\!\rangle \vdash_{D_{<:}} \{A : \bot..T\} <: \{A : [\![S]\!]..\top\}$$

Recall that $\langle\!\langle \Gamma \rangle\!\rangle$ is invertible. By Lemma 9, we know

$$\langle\!\langle \Gamma \rangle\!\rangle \vdash_{D_{<:}} [\![S]\!] <: \bot$$

Furthermore, by Lemma 8, we know $[\![S]\!] = \bot$. By inspecting $[\![\cdot]\!]$, we see that $\bot$ is not in the image, and therefore both Sel1' and SR cases are discharged by contradiction. ◻

Theorem 13. *Subtyping in $D_{<:}$ is undecidable.*

Proof. The proof is by reduction from $F^-_{<:}$ using the mapping from Definition 7 but without the function case. For the if direction, Theorem 3 applies since the $F^-_{<:}$ subtyping rules are a subset of the $F_{<:}$ subtyping rules. The only if direction is proved by the previous theorem. ◻

As we have seen, the only change in the normal form rules of $D_{<:}$ subtyping is that the Trans rule is removed and replaced with the SR rule. In other words, the only thing that transitivity really contributes to $D_{<:}$ is the phenomenon of subtyping reflection. Conversely, if we exclude subtyping reflection from $D_{<:}$, then it no longer has transitivity of subtyping.

The undecidability proof relies only on the common features of $F^-_{<:}$ and $D_{<:}$, and in particular, it does not depend on the SR rule. If we remove this rule from $D_{<:}$, subtyping in the resulting variant is still undecidable.

Theorem 14. *Subtyping in $D_{<:}$ normal form without the SR rule is undecidable.*

Proof. The proof is the same as Theorem 13, but without the SR case. ◻

4.7 Undecidability of Typing

In most calculi, undecidability of typing usually follows by some simple reduction from undecidability of subtyping in the same calculus. For example, for $D_{<:}$, we might map the subtyping problem $\Gamma \vdash_{D_{<:}} S <: U$ to the typing problem:

$$\Gamma \vdash_{D_{<:}} \{A = S\} : \{A : \bot..U\}$$

and conjecture that the two problems are equivalent. In $D_{<:}$, however, we have to be careful because of the possibility of subtyping reflection. Indeed, it turns out that the two problems are *not* equivalent. As a counterexample, note that if $\Gamma(w) = \{A : \{A : S..S\}..\{A : \bot..U\}\}$, then the typing problem is true (since $\Gamma \vdash_{D_{<:}} \{A = S\} : \{A : S..S\}$ and $\Gamma \vdash_{D_{<:}} \{A : S..S\} <: \{A : \bot..U\}$) even if S and U are chosen so that the subtyping problem $\Gamma \vdash_{D_{<:}} S <: U$ is false.

In general, the approach to proving undecidability of typing using undecidability of subtyping depends on inversion properties, which do not always hold in $D_{<:}$ due to subtyping reflection, so this approach does not work for $D_{<:}$. Nevertheless, $D_{<:}$ typing still turns out to be undecidable, but to prove it, we must reduce not from $D_{<:}$ subtyping, but from $F^-_{<:}$ subtyping, which does obey inversion properties.

Theorem 15. *For all Γ, S and U in $F^-_{<:}$,*

$$\textit{if } \langle\!\langle\Gamma\rangle\!\rangle \vdash_{D_{<:}} \{A = [\![S]\!]\} : \{A : \bot..[\![U]\!]\}, \textit{ then } \Gamma \vdash_{F^-_{<:}} S <: U.$$

Proof. The only typing rules that apply to $\{A = [\![S]\!]\}$ are Typ-I and Sub. Therefore, the premise implies that $\langle\!\langle\Gamma\rangle\!\rangle \vdash_{D_{<:}} \{A : [\![S]\!]..[\![S]\!]\} <: \{A : \bot..[\![U]\!]\}$. Since $\langle\!\langle\Gamma\rangle\!\rangle$ is invertible, Lemma 9 implies $\langle\!\langle\Gamma\rangle\!\rangle \vdash_{D_{<:}} [\![S]\!] <: [\![U]\!]$ and Theorem 12 implies $\Gamma \vdash_{F^-_{<:}} S <: U$. ◻

Theorem 16. *$D_{<:}$ typing is undecidable.*

Proof. By reduction from $F^-_{<:}$ subtyping, mapping the $F^-_{<:}$ subtyping problem $\Gamma \vdash_{F^-_{<:}} S <: U$ to the $D_{<:}$ typing problem $\langle\!\langle\Gamma\rangle\!\rangle \vdash_{D_{<:}} \{A = [\![S]\!]\} : \{A : \bot..[\![U]\!]\}$. The if direction is immediate and the only if direction is proved by the previous theorem. ◻

$$\frac{}{\Gamma \vdash_{D_{<:}K} T <: \top}\ \text{K-Top} \qquad \frac{}{\Gamma \vdash_{D_{<:}K} \bot <: T}\ \text{K-Bot} \qquad \frac{}{\Gamma \vdash_{D_{<:}K} x.A <: x.A}\ \text{K-VRefl}$$

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

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

Fig. 5. Definition of kernel $D_{<:}$

5 KERNEL $D_{<:}$

5.1 Motivation and Definition

In the previous section, we showed that both typing and subtyping in $D_{<:}$ are undecidable. A natural question to ask is *what fragments of* $D_{<:}$ *are decidable?* In this section, we consider one such fragment.

We base our adjustments to $D_{<:}$ on its normal form. The first adjustment is inspired by $F_{<:}$, which becomes decidable if its F-All rule is restricted to a *kernel* rule that requires the parameter types of both universal types to be identical [Cardelli and Wegner 1985]. We apply the same restriction to the $D_{<:}$ All rule.

The second adjustment is to remove the SR rule. There are several reasons for that:

(1) Subtyping reflection is a consequence of unintended interactions among the Sel1', Sel2' and Trans rules.
(2) Nieto [2017] observed that the implementation of subtyping in the Scala compiler violates transitivity in some cases, and these cases correspond exactly to the SR rule. That is, the Scala compiler does not implement this rule.
(3) We conjecture that a calculus with subtyping reflection is undecidable.

The calculus after these two changes is shown in Figure 5. We will see that this calculus is decidable, so we call it *kernel* $D_{<:}$, following the convention in [Pierce 2002].

We can show that kernel $D_{<:}$ is sound with respect to the original (full) $D_{<:}$:

Theorem 17. *If* $\Gamma \vdash_{D_{<:}K} S <: U$*, then* $\Gamma \vdash_{D_{<:}} S <: U$.

If kernel $D_{<:}$ is decidable, it cannot also be complete for full $D_{<:}$. For example, it does not admit the following subtyping judgment that is admitted by full $D_{<:}$.

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

Kernel $D_{<:}$ rejects it because $x.A$ and $\top$ are not syntactically identical.

Moreover, kernel $D_{<:}$ rejects conclusions that can only be drawn from subtyping reflection, such as:

$$x : \{A : \top..\bot\} \vdash_{D_{<:}} \top <: \bot$$

This judgment can only be achieved by invoking Trans or SR, but both of these rules are absent from kernel $D_{<:}$.

5.2 Step subtyping

Nieto [2017] defined step subtyping, a partial algorithm for deciding a fragment of $D_{<:}$ subtyping based on ideas developed for subtyping in $F_{<:}$ by Pierce [2002]. We briefly review the step subtyping

$$\frac{}{\Gamma \vdash_{D_{<:}S} T <: \top}\ \textsc{S-Top} \qquad \frac{}{\Gamma \vdash_{D_{<:}S} \bot <: T}\ \textsc{S-Bot} \qquad \frac{}{\Gamma \vdash_{D_{<:}S} x.A <: x.A}\ \textsc{S-VRefl}$$

$$\frac{\Gamma \vdash_{D_{<:}S} S' <: S \qquad \Gamma \vdash_{D_{<:}S} U <: U'}{\Gamma \vdash_{D_{<:}S} \{A : S..U\} <: \{A : S'..U'\}}\ \textsc{S-Bnd} \qquad \frac{\Gamma; x : S \vdash_{D_{<:}S} U <: U'}{\Gamma \vdash_{D_{<:}S} \forall(x : S)U <: \forall(x : S)U'}\ \textsc{S-All}$$

$$\frac{\Gamma \vdash x.A \searrow S \qquad \Gamma \vdash_{D_{<:}S} T <: S}{\Gamma \vdash_{D_{<:}S} T <: x.A}\ \textsc{S-Sel1} \qquad \frac{\Gamma \vdash x.A \nearrow U \qquad \Gamma \vdash_{D_{<:}S} U <: T}{\Gamma \vdash_{D_{<:}S} x.A <: T}\ \textsc{S-Sel2}$$

Fig. 6. Definition of step subtyping operation [Nieto 2017]

Exposure

$$\frac{T \text{ is not a path}}{\Gamma \vdash T \Uparrow T}\ \textsc{Exp-Stop} \qquad \frac{}{\Gamma \vdash T \Uparrow \top}\ \textsc{Exp-Top}^* \qquad \frac{\Gamma_1 \vdash T \Uparrow \bot}{\Gamma_1; x : T; \Gamma_2 \vdash x.A \Uparrow \bot}\ \textsc{Exp-Bot}$$

$$\frac{\Gamma_1 \vdash T \Uparrow \{A : S..U\} \qquad \Gamma_1 \vdash U \Uparrow U'}{\Gamma_1; x : T; \Gamma_2 \vdash x.A \Uparrow U'}\ \textsc{Exp-Bnd}$$

Upcast/ Downcast

$$\frac{}{\Gamma \vdash x.A \nearrow \top}\ \textsc{Uc-Top}^* \qquad \frac{\Gamma_1 \vdash T \Uparrow \bot}{\Gamma_1; x : T; \Gamma_2 \vdash x.A \nearrow \bot}\ \textsc{Uc-Bot} \qquad \frac{\Gamma_1 \vdash T \Uparrow \{A : S..U\}}{\Gamma_1; x : T; \Gamma_2 \vdash x.A \nearrow U}\ \textsc{Uc-Bnd}$$

$$\frac{}{\Gamma \vdash x.A \searrow \bot}\ \textsc{Dc-bot}^* \qquad \frac{\Gamma_1 \vdash T \Uparrow \bot}{\Gamma_1; x : T; \Gamma_2 \vdash x.A \searrow \top}\ \textsc{Dc-Top} \qquad \frac{\Gamma_1 \vdash T \Uparrow \{A : S..U\}}{\Gamma_1; x : T; \Gamma_2 \vdash x.A \searrow S}\ \textsc{Dc-Bnd}$$

Fig. 7. Definitions of **Exposure** and **Upcast** / **Downcast** operations [Nieto 2017]

algorithm here. In the next section, we will observe that the fragment of $D_{<:}$ subtyping decided by the algorithm turns out to be exactly the kernel $D_{<:}$ that we defined in the previous section. We made some adjustments to the presentation to set up a framework, so the definition is not identical to Nieto's, but the adjustments are minor and have no impact on expressiveness.

Definition 12. *Step subtyping is defined using the inference rules in Figure 6. The algorithm searches for a derivation using these rules, backtracking if necessary. Backtracking eventually terminates by Theorem 19.*

The definitions of kernel $D_{<:}$ and step subtyping look similar. The differences are the cases related to path types. For these types, step subtyping uses three additional operations, **Exposure** ($\Uparrow$), **Upcast** ($\nearrow$), and **Downcast** ($\searrow$). The purpose of **Upcast** (**Downcast**) is, given a path type $x.A$, to look up x in the typing context to a type member declaration $\{A : S..U\}$ and read off the upper bound U (lower bound S, respectively). A complication, however, is that the typing context could assign to x another path type. Therefore, **Upcast** and **Downcast** use **Exposure**, whose purpose is to convert a type that could be a path type to a supertype that is guaranteed to not be a path type. **Exposure** maps every non-path type to itself, and it maps a path type $x.A$ to its supertype U in a similar way as **Upcast**. However, U could itself be a path type, so, unlike **Upcast**, **Exposure** calls itself recursively on U. This guarantees that the type returned from **Exposure** is never a path type.

The definitions of these operations are shown in Figure 7. The Exp-Top, Uc-Top and Dc-bot rules are defined to make the operations total functions. We mark them with asterisks to indicate that they apply only when no other rules do, and therefore each of the three operations has exactly one rule to apply for any given input.

Upcast and **Downcast** are shallow wrappers over **Exposure**. Notice that **Upcast** and **Downcast** are not even recursive. When handling a path type $x.A$, they use **Exposure** to find a non-path supertype of $\Gamma(x)$ and simply return bounds in the right directions. It is possible that **Upcast** and **Downcast** return other path types.

Notice that in the Exp-Bot and Exp-Bnd rules, the recursive calls continue with Γ_1, the context preceding x. This ensures termination of **Exposure**. As long as the original context $\Gamma_1; x : T; \Gamma_2$ is well-formed, T is closed in the truncated context Γ_1.

Nieto showed that step subtyping is a sound and terminating algorithm.

Theorem 18. *[Nieto 2017] Step subtyping as an algorithm is sound w.r.t. full $D_{<:}$.*

$$\textit{If } \Gamma \vdash_{D_{<:S}} S <: U, \textit{ then } \Gamma \vdash_{D_{<:}} S <: U$$

Theorem 19. *[Nieto 2017] Step subtyping as an algorithm terminates.*

5.3 Soundness and Completeness of Step Subtyping

In this section, we will show that the subset of $D_{<:}$ subtyping relationships that step subtyping discovers turns out to be exactly the relation defined by the declarative kernel $D_{<:}$ rules. We begin by proving some basic properties of kernel $D_{<:}$.

Although the kernel $D_{<:}$ subtyping reflexivity rule K-VRefl applies only to path types, subtyping is actually reflexive for all types:

Lemma 20. *Kernel $D_{<:}$ subtyping is reflexive.*

$$\Gamma \vdash_{D_{<:K}} T <: T$$

Since kernel $D_{<:}$ does not have the SR or Trans rules, transitivity no longer holds in general, but it does hold on $\top$ and $\bot$:

Lemma 21. *If $\Gamma \vdash_{D_{<:K}} \top <: U$, then $\Gamma \vdash_{D_{<:K}} S <: U$.*

Lemma 22. *If $\Gamma \vdash_{D_{<:K}} S <: \bot$, then $\Gamma \vdash_{D_{<:K}} S <: U$.*

Comparing step subtyping with kernel $D_{<:}$, we will show soundness of step subtyping first and completeness second. In step subtyping, the operations are separated into two layers. The first is the subtyping algorithm itself and the second is **Exposure**, which handles path types. The proof needs to go from the reverse direction by connecting **Exposure** with kernel $D_{<:}$ first.

Lemma 23. *If $\Gamma \vdash S \Uparrow T$ and $\Gamma \vdash_{D_{<:K}} T <: U$, then $\Gamma \vdash_{D_{<:K}} S <: U$.*

Proof. By induction on the derivation of **Exposure**. □

We can then show that step subtyping is sound.

Theorem 24. *(soundness of step subtyping w.r.t. kernel $D_{<:}$) If $\Gamma \vdash_{D_{<:S}} S <: U$, then $\Gamma \vdash_{D_{<:K}} S <: U$.*

Proof. By induction on step subtyping. From the rules, we can see that kernel $D_{<:}$ and step subtyping are almost identical, except for the S-Sel1 and S-Sel2 cases. These cases can be discharged by expanding **Upcast** and **Downcast** and then applying Lemma 23. □

Now we proceed to the opposite direction, proving completeness of step subtyping.

Theorem 25. *(completeness of step subtyping w.r.t. kernel $D_{<:}$)*
If $\Gamma \vdash_{D_{<:}K} S <: U$, then $\Gamma \vdash_{D_{<:}S} S <: U$.

Proof. The proof requires an intricate strengthening of the statement to obtain a strong enough inductive hypothesis: if $\Gamma \vdash_{D_{<:}K} S <: U$ and this derivation contains n steps, then $\Gamma \vdash_{D_{<:}S} S <: U$, and if U is of the form $\{A : T_1..T_2\}$, then $\Gamma \vdash S \Uparrow S'$ for some S', and either

(1) $S' = \bot$ or
(2) $S' = \{A : T_1'..T_2'\}$ for some T_1' and T_2' such that
  (a) $\Gamma \vdash_{D_{<:}S} T_1 <: T_1'$ and
  (b) $\Gamma \vdash_{D_{<:}K} T_2' <: T_2$, and the number of steps in the derivation of $\Gamma \vdash_{D_{<:}K} T_2' <: T_2$ is less than or equal to n.

The proof is by strong induction on n.

To prove $\Gamma \vdash_{D_{<:}S} S <: U$, the non-trivial cases are K-Sel1 and K-Sel2 cases; we discuss the latter. The antecedent is $\Gamma \vdash_{D_{<:}K} \Gamma(x) <: \{A : \bot..U\}$. This case requires the strengthened induction hypothesis, since the original would only imply that $\Gamma \vdash_{D_{<:}S} \Gamma(x) <: \{A : \bot..U\}$, which is insufficient to establish $\Gamma \vdash_{D_{<:}S} x.A <: U$. To establish this conclusion, we wish to apply the S-Sel2 rule. The strengthened induction hypothesis is designed specifically to provide the necessary premises of this rule.

It remains to prove the properties that the strengthened statement of the theorem requires in the case that U is of the form $\{A : T_1..T_2\}$. The type U can have this form in the conclusions of three rules: K-Bot, K-Bnd and K-Sel2. Only the K-Sel2 case is interesting. The conclusion of this rule forces $S = y.A$ for some y, and the antecedent is $\Gamma \vdash_{D_{<:}K} \Gamma(y) <: \{A : \bot..\{A : T_1..T_2\}\}$. Notice that this supertype $\{A : \bot..\{A : T_1..T_2\}\}$ is a type declaration for which the strengthened statement of the theorem ensures that $\Gamma \vdash \Gamma(y) \Uparrow S'$ for some S' with additional properties. Specifically, applying the induction hypothesis to this antecedent leads to two cases:

(1) When $\Gamma \vdash \Gamma(y) \Uparrow \bot$, the goal $\Gamma \vdash y.A \Uparrow \bot$ follows by Exp-Bot.
(2) Otherwise, for some T_1' and T_2', we obtain additional antecedents:
  (a) $\Gamma \vdash \Gamma(y) \Uparrow \{A : T_1'..T_2'\}$,
  (b) $\Gamma \vdash_{D_{<:}S} \bot <: T_1'$, and
  (c) $\Gamma \vdash_{D_{<:}K} T_2' <: \{A : T_1..T_2\}$ by a derivation with strictly fewer that n steps.
  The intention is to apply the Exp-Bnd rule, but this rule requires an **Exposure** on T_2' as well. This can be achieved by applying the inductive hypothesis to the third antecedent again. This yields $\Gamma \vdash T_2' \Uparrow T_2''$ for some T_2'' and this case is concluded, so we can apply Exp-Bnd to obtain $\Gamma \vdash y.A \Uparrow T_2''$, where T_2'' satisfies the properties that the strengthened induction hypothesis requires of S'.

□

Hence, we have shown that the subrelation of $D_{<:}$ subtyping induced by the step subtyping algorithm is exactly the kernel $D_{<:}$ subtyping relation.

6 STRONG KERNEL $D_{<:}$

6.1 Motivation and Definition

In the previous section, we defined a decidable fragment of $D_{<:}$, kernel $D_{<:}$. Notwithstanding its decidability, it comes with obvious disadvantages. One example is the judgment we presented in §5.1:

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

This judgment is admitted in full $D_{<:}$ but not kernel $D_{<:}$. The latter rejects this judgment because it requires the parameter types to be syntactically identical. However, we can see that here $x.A$ and $\top$ are in a special situation: $x.A$ is defined with $\top$ as both its lower and upper bounds, which makes $x.A$ an *alias* for $\top$. In Scala, we would like to be able to use aliased types interchangeably. The kernel requirement of syntactically identical parameter types significantly restricts the usefulness of type aliases. Hence, the aim of this section is to (at least) lift this restriction while maintaining decidability.

The inspiration for the new calculus comes from writing out the typing context *twice* in a subtyping derivation. For example, the All rule is:

$$\frac{\Gamma \vdash_{D_{<:}} S' <: S \qquad \Gamma; x : S' \vdash_{D_{<:}} U_x <: U'_x}{\Gamma \vdash_{D_{<:}} \forall(x : S)U <: \forall(x : S')U'} \text{ All}$$

Let us write the contexts twice for this rule:

$$\frac{\Gamma \vdash_{D_{<:}} S' <: S \dashv \Gamma \qquad \Gamma; x : S' \vdash_{D_{<:}} U_x <: U'_x \dashv \Gamma; x : S'}{\Gamma \vdash_{D_{<:}} \forall(x : S)U <: \forall(x : S')U' \dashv \Gamma} \text{ All-TwoContexts}$$

Now do the same for the kernel version too:

$$\frac{\Gamma; x : S \vdash_{D_{<:}} U_x <: U'_x \dashv \Gamma; x : S}{\Gamma \vdash_{D_{<:}} \forall(x : S)U <: \forall(x : S)U' \dashv \Gamma} \text{ K-All-TwoContexts}$$

So far, both copies of the context have been the same, so the second copy is redundant. However, comparing these two rules for a moment, we start to see some potential for improvement. In the premise comparing $U_x <: U'_x$, the only difference are the primes on S in the typing contexts: the first rule uses S' on both sides, while the second rule uses S on both sides. Since U_x comes from a universal type where x has type S, and U'_x from one where x has type S', what if we took the middle ground between the two rules, and added S to the left context and S' to the right context?

$$\frac{\Gamma \vdash_{D_{<:}} S' <: S \dashv \Gamma \qquad \Gamma; x : S \vdash_{D_{<:}} U_x <: U'_x \dashv \Gamma; x : S'}{\Gamma \vdash_{D_{<:}} \forall(x : S)U <: \forall(x : S')U' \dashv \Gamma} \text{ All-AsymmetricContexts}$$

The new rule enables the contexts to be different, so it justifies maintaining both contexts. But how will a calculus with this hybrid rule behave? Will it be strictly in between the decidable kernel $D_{<:}$ and the undecidable full $D_{<:}$ in expressiveness? Will it be decidable? We will show that the answer to both questions is yes. The new hybrid rule allows comparison of function types with *different* parameter types, and the return types are compared in two different contexts. In particular, it admits the example judgment with the aliased parameter types with which we began this section.

We call this new calculus *strong kernel* $D_{<:}$, and define its subtyping rules in Figure 8. In strong kernel $D_{<:}$, the free variables of types on the left (right) are bound and looked up in the context on the left (right), respectively. This can be seen in the Sk-Sel1 and Sk-Sel2 rules. The Sk-All rule is the only rule that enables the two contexts to diverge. All of the other rules simply copy both contexts unchanged to the premises.

Since the typing judgment remains the same, we omit it here. From the typing judgment, the subsumption rule uses strong kernel $D_{<:}$ with the same context on both sides:

$$\frac{\Gamma \vdash_{D_{<:}} t : S \qquad \colorbox{lightgray}{Γ} \vdash_{D_{<:}SK} S <: U \dashv \colorbox{lightgray}{Γ}}{\Gamma \vdash_{D_{<:}} t : U} \text{ Sk-Sub}$$

$$\frac{}{\Gamma_1 \vdash_{D_{<:}SK} T <: \top \dashv \Gamma_2}\ \textsc{Sk-Top} \qquad \frac{}{\Gamma_1 \vdash_{D_{<:}SK} \bot <: T \dashv \Gamma_2}\ \textsc{Sk-Bot}$$

$$\frac{}{\Gamma_1 \vdash_{D_{<:}SK} x.A <: x.A \dashv \Gamma_2}\ \textsc{Sk-VRefl} \qquad \frac{\Gamma_1 \vdash_{D_{<:}SK} S_1 >: S_2 \dashv \Gamma_2 \qquad \Gamma_1 \vdash_{D_{<:}SK} U_1 <: U_2 \dashv \Gamma_2}{\Gamma_1 \vdash_{D_{<:}SK} \{A : S_1..U_1\} <: \{A : S_2..U_2\} \dashv \Gamma_2}\ \textsc{Sk-Bnd}$$

$$\frac{\Gamma_1 \vdash_{D_{<:}SK} S_1 >: S_2 \dashv \Gamma_2 \qquad \Gamma_1; x : S_1 \vdash_{D_{<:}SK} U_1 <: U_2 \dashv \Gamma_2; x : S_2}{\Gamma_1 \vdash_{D_{<:}SK} \forall(x : S_1)U_1 <: \forall(x : S_2)U_2 \dashv \Gamma_2}\ \textsc{Sk-All}$$

$$\frac{\Gamma_1 \vdash_{D_{<:}SK} \{A : S..\top\} >: \Gamma_2(x) \dashv \Gamma_2}{\Gamma_1 \vdash_{D_{<:}SK} S <: x.A \dashv \Gamma_2}\ \textsc{Sk-Sel1} \qquad \frac{\Gamma_1 \vdash_{D_{<:}SK} \Gamma_1(x) <: \{A : \bot..U\} \dashv \Gamma_2}{\Gamma_1 \vdash_{D_{<:}SK} x.A <: U \dashv \Gamma_2}\ \textsc{Sk-Sel2}$$

Fig. 8. Definition of strong kernel $D_{<:}$

$$\frac{}{\bullet \subseteq_{<:} \bullet}\ \textsc{Ope-Nil} \qquad \frac{\Gamma \subseteq_{<:} \Gamma'}{\Gamma; x : T \subseteq_{<:} \Gamma'}\ \textsc{Ope-Drop} \qquad \frac{\Gamma \subseteq_{<:} \Gamma' \qquad \Gamma \vdash_{D_{<:}} S <: U}{\Gamma; x : S \subseteq_{<:} \Gamma'; x : U}\ \textsc{Ope-Keep}$$

Fig. 9. Definition of $OPE_{<:}$

6.2 Properties of Strong Kernel $D_{<:}$

In this section, we will prove that the subtyping relation defined by strong kernel $D_{<:}$ is in between kernel $D_{<:}$ and full $D_{<:}$ in expressiveness. As a first step, we need to prove reflexivity.

Lemma 26. *Strong kernel $D_{<:}$ is reflexive.*

$$\Gamma_1 \vdash_{D_{<:}SK} T <: T \dashv \Gamma_2$$

Proof. By induction on T. ◻

In the next two theorems, we wish to show that strong kernel $D_{<:}$ is in between kernel $D_{<:}$ and full $D_{<:}$ in terms of expressiveness:

Theorem 27. *If $\Gamma \vdash_{D_{<:}K} S <: U$ then $\Gamma \vdash_{D_{<:}SK} S <: U \dashv \Gamma$.*

Proof. By induction on the derivation. The K-All case requires reflexivity of strong kernel $D_{<:}$. ◻

Theorem 28. *If $\Gamma \vdash_{D_{<:}SK} S <: U \dashv \Gamma$ then $\Gamma \vdash_{D_{<:}} S <: U$.*

Before we can prove this theorem, we need to define a new concept, a relationship between the two typing contexts.

Definition 13. *The order preserving sub-environment relation between two contexts, or $OPE_{<:}$, is defined in Figure 9.*

Intuitively, If $\Gamma \subseteq_{<:} \Gamma'$, then Γ is a more "informative" context than Γ'. $OPE_{<:}$ is a combination of the narrowing and weakening properties. The following properties of $OPE_{<:}$ confirm this intuition.

Lemma 29. *$OPE_{<:}$ is reflexive.*

$$\Gamma \subseteq_{<:} \Gamma$$

Lemma 30. *$OPE_{<:}$ is transitive.*

$$\text{If } \Gamma_1 \subseteq_{<:} \Gamma_2 \text{ and } \Gamma_2 \subseteq_{<:} \Gamma_3, \text{ then } \Gamma_1 \subseteq_{<:} \Gamma_3.$$

Theorem 31. *(respectfulness) Full $D_{<:}$ subtyping is preserved by $OPE_{<:}$.*
If $\Gamma \subseteq_{<:} \Gamma'$ and $\Gamma' \vdash_{D_{<:}} S <: U$, then $\Gamma \vdash_{D_{<:}} S <: U$.

Given these results, we can proceed to proving the soundness of strong kernel $D_{<:}$ with respect to full $D_{<:}$, Theorem 28. We prove a stronger result:

Theorem 32. *If $\Gamma_1 \vdash_{D_{<:}SK} S <: U \dashv \Gamma_2$, $\Gamma \subseteq_{<:} \Gamma_1$ and $\Gamma \subseteq_{<:} \Gamma_2$, then $\Gamma \vdash_{D_{<:}} S <: U$.*

Proof. By induction on the strong kernel subtyping derivation. □

Then Theorem 28 follows from reflexivity of $OPE_{<:}$.

Since we will show that strong kernel $D_{<:}$ is decidable, it cannot also be complete with respect to full $D_{<:}$. Strong kernel $D_{<:}$ is strictly weaker in the Sk-All rule. For example, consider the following derivation in full $D_{<:}$:

$$\dfrac{\dfrac{\text{straightforward}}{\vdash_{D_{<:}} \{A : \bot..\bot\} <: \{A : \bot..\top\}}\ \textsc{Bnd} \qquad \dfrac{\text{straightforward}}{x : \{A : \bot..\bot\} \vdash_{D_{<:}} x.A <: \bot}\ \textsc{Sel2}}{\vdash_{D_{<:}} \forall(x : \{A : \bot..\top\})x.A <: \forall(x : \{A : \bot..\bot\})\bot}\ \textsc{All}$$

This judgment is rejected by strong kernel $D_{<:}$ because the comparison of the returned types relies on the parameter type to the right of <:, which is not possible in strong kernel $D_{<:}$. Notice that this example uses aliasing information from the right parameter type (i.e. that $x.A$ is an alias of $\bot$) to reason about the left return type (i.e. that $x.A$ is a subtype of $\bot$), which is something that strong kernel $D_{<:}$ cannot do.

Another more obvious difference is that full $D_{<:}$ admits subtyping reflection but strong kernel $D_{<:}$ does not. For example, the following judgment is not admitted in strong kernel $D_{<:}$.

$$x : \{A : \top..\bot\} \vdash_{D_{<:}} \top <: \bot$$

These examples show that full $D_{<:}$ is strictly more expressive than strong kernel $D_{<:}$. In fact, even some subtyping relationships in full $F_{<:}$ are not admitted by strong kernel $D_{<:}$ (under the mapping in Definition 7). One example of such a subtyping relationship is:

$$X <: \top \vdash_{F_{<:}} (\forall Y <: \top.Y) <: (\forall Y <: X : X)$$

The $D_{<:}$ equivalent is *not* admitted by strong kernel:

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

This is because the following subderivation is rejected:

$$x : \{A : \bot..\top\}; y : \{A : \bot..\top\} \vdash_{D_{<:}SK} y.A <: x.A \dashv x : \{A : \bot..\top\}; y : \{A : \bot..x.A\}$$

From the left context, we can conclude only $y.A <: \top$. From the right context, we can conclude only $\bot <: x.A$. These two facts are insufficient to derive the goal $y.A <: x.A$.

On the other hand, strong kernel $D_{<:}$ *does admit* the motivating aliasing example from the beginning of this section:

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

$$\dfrac{\dfrac{\text{reflexivity}}{\Gamma \vdash_{D_{<:}SK} x.A >: \top \dashv \Gamma}\ \textsc{Sk-Sel1} \qquad \dfrac{}{\Gamma; y : x.A \vdash_{D_{<:}SK} \top <: \top \dashv \Gamma; y : \top}\ \textsc{Sk-Top}}{\Gamma \vdash_{D_{<:}SK} \forall(y : x.A)\top <: \forall(y : \top)\top \dashv \Gamma}\ \textsc{Sk-All}$$

$$\frac{}{\Gamma_1 \gg T <: \top \ll \Gamma_2}\ \text{SA-Top} \qquad \frac{}{\Gamma_1 \gg \bot <: T \ll \Gamma_2}\ \text{SA-Bot} \qquad \frac{}{\Gamma_1 \gg x.A <: x.A \ll \Gamma_2}\ \text{SA-VRefl}$$

$$\frac{\begin{array}{c}\Gamma_1 \gg S >: S' \ll \Gamma_2 \\ \Gamma_1 \gg U <: U' \ll \Gamma_2\end{array}}{\Gamma_1 \gg \{A : S..U\} <: \{A : S'..U'\} \ll \Gamma_2}\ \text{SA-Bnd} \qquad \frac{\begin{array}{c}\Gamma_1 \gg S >: S' \ll \Gamma_2 \\ \Gamma_1; x : S \gg U <: U' \ll \Gamma_2; x : S'\end{array}}{\Gamma_1 \gg \forall(x : S)U <: \forall(x : S')U' \ll \Gamma_2}\ \text{SA-All}$$

$$\frac{\begin{array}{c}\Gamma_2 \vdash x.A \searrow \Gamma_2' \vdash T \\ \Gamma_1 \gg S <: T \ll \Gamma_2'\end{array}}{\Gamma_1 \gg S <: x.A \ll \Gamma_2}\ \text{SA-Sel1} \qquad \frac{\begin{array}{c}\Gamma_1 \vdash x.A \nearrow \Gamma_1' \vdash T \\ \Gamma_1' \gg T <: U \ll \Gamma_2\end{array}}{\Gamma_1 \gg x.A <: U \ll \Gamma_2}\ \text{SA-Sel2}$$

Fig. 10. Definition of stare-at subtyping

In general, the Sk-All rule admits subtyping between function types with different (but contravariant) parameter types. This shows that strong kernel $D_{<:}$ is strictly more powerful than kernel $D_{<:}$.

6.3 Stare-at subtyping

It remains to show that strong kernel $D_{<:}$ is decidable. We will present the decision procedure first. We will prove some of its properties in the next section, and finally prove that it is a sound and complete decision procedure for strong kernel $D_{<:}$ in §6.5. The decision procedure is shown in Figure 10. We call it *stare-at subtyping*, inspired by the notation $\Gamma_1 \gg S <: U \ll \Gamma_2$. If we see $\gg$ and $\ll$ as eyes and $<:$ as a nose, then the notation looks like a face, and the two eyes are staring at the nose.

In the same way as for step subtyping, the stare-at subtyping algorithm searches for a derivation using the inference rules, backtracking when necessary. We will prove that this backtracking terminates (Theorem 42).

Stare-at subtyping generalizes step subtyping by operating on two contexts. One can think of stare-at subtyping as a collaborative game between two players, Alice and Bob. Alice is responsible for the context and type to the left of $<:$ or $>:$, while Bob is responsible for the other side. In particular, Alice and Bob are completely independent and do not need to see the contexts or types held by their collaborator. Most of the rules are just straightforward extensions of the corresponding rules of step subtyping with two contexts, except for three cases: SA-All, SA-Sel1 and SA-Sel2.

In the SA-All rule, the parameter types are allowed to be different, so there is an additional premise that compares the parameter types. This rule can handle not only the aliasing example, but also cases where S' is a strict subtype of S. When comparing the return types, Alice and Bob work on their own extended contexts, so subsequently, if Alice and Bob refer to x, they potentially see x at different types.

Similar to step subtyping, stare-at subtyping relies on another operation to handle path types which generalizes **Exposure**: **Revealing**. **Upcast** and **Downcast** are generalized accordingly to reflect the differences between **Exposure** and **Revealing**. Like in step subtyping, the Rv-Top, U-Top and D-bot rules only apply when no other rules apply and the three operations are all total.

Revealing is similar to **Exposure** in that it finds a non-path supertype of the given type, and its rules mirror those of **Exposure**. The difference is that in addition to a type, **Revealing** also returns a typing context. The typing context is a prefix of the input typing context long enough to type any free variables that may occur in the type that **Revealing** returns. This returned prefix context participates in further subtyping decisions and makes it quite easy to prove termination.

Revealing

$$\frac{T \text{ is not a path}}{\Gamma \vdash T \Uparrow \Gamma \vdash T}\ \textsc{Rv-Stop} \qquad \frac{}{\Gamma \vdash T \Uparrow \bullet \vdash \top}\ \textsc{Rv-Top}^* \qquad \frac{\Gamma_1 \vdash T \Uparrow \Gamma_1' \vdash \bot}{\Gamma_1; x : T; \Gamma_2 \vdash x.A \Uparrow \bullet \vdash \bot}\ \textsc{Rv-Bot}$$

$$\frac{\Gamma_1 \vdash T \Uparrow \Gamma_1' \vdash \{A : S..U\} \qquad \Gamma_1' \vdash U \Uparrow \Gamma_1'' \vdash U'}{\Gamma_1; x : T; \Gamma_2 \vdash x.A \Uparrow \Gamma_1'' \vdash U'}\ \textsc{Rv-Bnd}$$

Upcast/ Downcast

$$\frac{}{\Gamma \vdash x.A \nearrow \bullet \vdash \top}\ \textsc{U-Top}^* \qquad \frac{}{\Gamma \vdash x.A \searrow \bullet \vdash \bot}\ \textsc{D-bot}^*$$

$$\frac{\Gamma_1 \vdash T \Uparrow \Gamma_1' \vdash \bot}{\Gamma_1; x : T; \Gamma_2 \vdash x.A \nearrow \bullet \vdash \bot}\ \textsc{U-Bot} \qquad \frac{\Gamma_1 \vdash T \Uparrow \Gamma_1' \vdash \bot}{\Gamma_1; x : T; \Gamma_2 \vdash x.A \searrow \bullet \vdash \top}\ \textsc{D-Top}$$

$$\frac{\Gamma_1 \vdash T \Uparrow \Gamma_1' \vdash \{A : S..U\}}{\Gamma_1; x : T; \Gamma_2 \vdash x.A \nearrow \Gamma_1' \vdash U}\ \textsc{U-Bnd} \qquad \frac{\Gamma_1 \vdash T \Uparrow \Gamma_1' \vdash \{A : S..U\}}{\Gamma_1; x : T; \Gamma_2 \vdash x.A \searrow \Gamma_1' \vdash S}\ \textsc{D-Bnd}$$

Fig. 11. Definition of **Revealing** and new definitions of **Upcast** and **Downcast**

We design **Revealing** to return an extra context to facilitate the termination proof of stare-at subtyping, which we will show in the next section. In particular, the addition of a returned context is not motivated by the two contexts in stare-at subtyping. Since **Revealing** achieves the same functionality as **Exposure** (which we will formally examine in the next section), stare-at subtyping would still be sound and terminating if we replaced **Revealing** with **Exposure** everywhere, but the termination proof would be technically much more tedious.

The **Upcast** and **Downcast** rules have the same structure as those of step subtyping, except that they return the typing context that they receive from **Revealing**. In the cases where they return $\top$ or $\bot$, they return an empty context because these types have no free variables. Like in step subtyping, the result types of **Upcast** and **Downcast** are used in the SA-Sel1 and SA-Sel2 subtyping rules. These rules use the shortened typing context that is returned from **Upcast** or **Downcast** in their recursive subtyping premises.

6.4 Properties of Stare-at Subtyping

We will now prove two properties of stare-at subtyping: soundness with respect to full $D_{<:}$, and termination. To prove soundness, we must first prove basic lemams to ensure that **Revealing**, **Upcast**, and **Downcast** satisfy their intended specification.

Lemma 33. *(**Revealing** gives prefixes) If $\Gamma \vdash S \Uparrow \Gamma' \vdash U$, then Γ' is a prefix of Γ.*

Lemma 34. *(**Revealing** returns no path) If $\Gamma \vdash S \Uparrow \Gamma' \vdash U$, then U is not a path type.*

Lemma 35. *(soundness of **Revealing**) If $\Gamma \vdash S \Uparrow \Gamma' \vdash U$, then $\Gamma \vdash_{D_{<:}} S <: U$.*

Lemma 36. *(well-formedness condition) If $\Gamma \vdash S \Uparrow \Gamma' \vdash U$, Γ is well-formed and $fv(S) \subseteq dom(\Gamma)$, then Γ' is well-formed and $fv(U) \subseteq dom(\Gamma')$.*

All of the lemmas above can be proved by direct induction.

Lemma 37. *The following all hold.*

(1) If $\Gamma \vdash x.A \nearrow (\searrow) \Gamma' \vdash T$, then Γ' is a prefix of Γ.
(2) If $\Gamma \vdash x.A \nearrow \Gamma' \vdash T$, then $\Gamma \vdash_{D_{<:}} x.A <: T$.

(3) If $\Gamma \vdash x.A \searrow \Gamma' \vdash T$, then $\Gamma \vdash_{D_{<:}} T <: x.A$.
(4) If $\Gamma \vdash x.A \nearrow (\searrow)\Gamma' \vdash T$, Γ is well-formed and $x \in dom(\Gamma)$, then Γ' is well-formed and $fv(T) \subseteq dom(\Gamma')$.

This proof is even simpler because **Upcast** and **Downcast** are not even recursive.

Now we can proceed to prove soundness of stare-at subtyping. In the next section, we will prove a stronger result, that stare-at subtyping is in fact sound even with respect to strong kernel $D_{<:}$, but for now, we show only that it is sound with respect to full $D_{<:}$. Like the soundness proof of strong kernel $D_{<:}$ (Theorem 32), the induction requires a stronger statement with typing contexts related by the $OPE_{<:}$ relation.

Theorem 38. *(soundness of stare-at subtyping) If $\Gamma_1 \gg S <: U \ll \Gamma_2$, $\Gamma \subseteq_{<:} \Gamma_1$ and $\Gamma \subseteq_{<:} \Gamma_2$, then $\Gamma \vdash_{D_{<:}} S <: U$.*

Proof. By induction on the derivation of stare-at subtyping. □

A corollary is that if Alice and Bob begin with the same context, then stare-at subtyping is sound with respect to full $D_{<:}$.

Theorem 39. *If $\Gamma \gg S <: U \ll \Gamma$, then $\Gamma \vdash_{D_{<:}} S <: U$.*

Next, we want to examine the termination of the operations. First we want to make sure that **Revealing** terminates as an algorithm.

Lemma 40. ***Revealing*** *terminates as an algorithm.*

Proof. The measure is the length of the input context (the number of variables in its domain). □

Now we want to examine the termination of stare-at subtyping. We first define the structural measures for types and contexts.

Definition 14. *The measure M of types and contexts is defined by the following equations.*

$$\begin{aligned} M(\top) &= 1 \\ M(\bot) &= 1 \\ M(x.A) &= 2 \\ M(\forall(x : S)U) &= 1 + M(S) + M(U) \\ M(\{A : S..U\}) &= 1 + M(S) + M(U) \\ M(\Gamma) &= \sum_{x:T \in \Gamma} M(T) \end{aligned}$$

As we can see, the measure simply counts the syntactic size of types and contexts. We can show that **Revealing** does not increase the input measure and **Upcast** and **Downcast** strictly decrease it.

Lemma 41. *If $\Gamma \vdash S \Uparrow \Gamma' \vdash U$, then $M(\Gamma) + M(S) \geq M(\Gamma') + M(U)$.*
If $\Gamma \vdash x.A \nearrow (\searrow)\Gamma' \vdash U$, then $M(\Gamma) + M(x.A) > M(\Gamma') + M(U)$.

Theorem 42. *Stare-at subtyping terminates as an algorithm.*

Proof. The measure is the sum of measures of all inputs: for $\Gamma_1 \gg S <: U \ll \Gamma_2$, the measure is $M(\Gamma_1) + M(S) + M(U) + M(\Gamma_2)$. Since the measure just reflects the syntactic sizes, it is easy to see that it decreases in all of the cases other than SA-Sel1 and SA-Sel2. These two cases are proven by the previous lemma. Notice that the proof is this easy because Alice and Bob use the returned contexts from **Upcast** and **Downcast** in both cases. □

6.5 Soundness and Completeness of Stare-at Subtyping

In the previous section, we showed that stare-at subtyping terminates and is sound for full $D_{<:}$. In this section, we strengthen the soundness proof to strong kernel $D_{<:}$, and also prove completeness with respect to strong kernel $D_{<:}$, to show that the fragment of full $D_{<:}$ decided by stare-at subtyping is exactly strong kernel $D_{<:}$. Our overall approach will mirror the proofs from §5.3 of soundness and completeness of step subtyping with respect to kernel $D_{<:}$.

First, we connect **Revealing** with strong kernel $D_{<:}$.

Lemma 43. *If* $\Gamma_1 \vdash S \Uparrow \Gamma_1' \vdash T$ *and* $\Gamma_1 \vdash_{D_{<:}SK} T <: U \dashv \Gamma_2$, *then* $\Gamma_1 \vdash_{D_{<:}SK} S <: U \dashv \Gamma_2$.

In this lemma, the Γ_1' returned from **Revealing** is not used in the rest of the statement. The intuition is that strong kernel does not shrink the context as **Revealing** does so Γ_1' is irrelevant.

This is all we need to show that stare-at subtyping is sound with respect to strong kernel $D_{<:}$.

Theorem 44. *(soundness of stare-at subtyping w.r.t. strong kernel $D_{<:}$)*
If $\Gamma_1 \gg S <: U \ll \Gamma_2$, *then* $\Gamma_1 \vdash_{D_{<:}SK} S <: U \dashv \Gamma_2$.

Proof. The proof is done by induction on the derivation of stare-at subtyping and it is very similar to the one of Theorem 24. ◻

The completeness proof is slightly trickier, because in the SA-Sel1 and SA-Sel2 cases, Alice and Bob work on prefix contexts in the recursive calls. In contrast, in the Sk-Sel1 and Sk-Sel2 rules of strong kernel $D_{<:}$, the subtyping judgments in the premises use the same full contexts as the conclusions. Therefore, we need to make sure that working on smaller contexts will not change the outcome.

Theorem 45. *(strengthening of stare-at subtyping) If* $\Gamma_1; \Gamma_1'; \Gamma_1'' \gg S <: U \ll \Gamma_2; \Gamma_2'; \Gamma_2''$, $fv(S) \subseteq dom(\Gamma_1; \Gamma_1'')$ *and* $fv(U) \subseteq dom(\Gamma_2; \Gamma_2'')$, *then* $\Gamma_1; \Gamma_1'' \gg S <: U \ll \Gamma_2; \Gamma_2''$.

Proof. By induction on the derivation of stare-at subtyping. ◻

By taking Γ_1'' and Γ_2'' to be empty, we know Alice and Bob are safe to work on the prefix contexts.
Now we can prove the completeness of stare-at subtyping.

Theorem 46. *(completeness of stare-at subtyping w.r.t. strong kernel $D_{<:}$)*
If $\Gamma_1 \vdash_{D_{<:}SK} S <: U \dashv \Gamma_2$, *then* $\Gamma_1 \gg S <: U \ll \Gamma_2$.

Proof. The proof is similar to the one of Theorem 25. We also need to strengthen the statement to the following: if $\Gamma_1 \vdash_{D_{<:}SK} S <: U \dashv \Gamma_2$ and this derivation contains n steps, then $\Gamma_1 \gg S <: U \ll \Gamma_2$ and if U is of the form $\{A : T_1..T_2\}$, then $\Gamma_1 \vdash S \Uparrow \Gamma_1' \vdash S'$, and either

(1) $S' = \bot$, or
(2) $S' = \{A : T_1'..T_2'\}$ for some T_1' and T_2', such that
  (a) $\Gamma_1 \gg T_1 <: T_1' \ll \Gamma_2$ and
  (b) $\Gamma_1 \vdash_{D_{<:}SK} T_2' <: T_2 \dashv \Gamma_2$, and the number of steps in this derivation is less than or equal to n.

The Sk-Sel1 and Sk-Sel2 cases are trickier. After invoking the inductive hypothesis, due to the well-formedness condition of **Upcast** and **Downcast**, we apply Theorem 45 so that the eventual derivation of stare-at subtyping works in prefix contexts. ◻

Therefore, we conclude that strong kernel and stare-at subtyping are the same language.

Completeness may seem somewhat surprising since stare-at subtyping truncates the typing contexts in the SA-Sel1 and SA-Sel2 cases while strong kernel subtyping does not. Technically, the truncation is justified by Theorem 45. Intuitively, since the prefixes of the typing contexts cover the free variables of the relevant type, they do include all of the information necessary to reason

$$\frac{}{\Gamma_1 \vdash_{F_{<:}SK} T <: \top \dashv \Gamma_2}\ \text{F-Sk-Top} \qquad \frac{}{\Gamma_1 \vdash_{F_{<:}SK} X <: X \dashv \Gamma_2}\ \text{F-Sk-VarRefl}$$

$$\frac{X <: T \in \Gamma_1 \quad \Gamma_1 \vdash_{F_{<:}SK} T <: U \dashv \Gamma_2}{\Gamma_1 \vdash_{F_{<:}SK} X <: Y \dashv \Gamma_2}\ \text{F-Sk-Tvar} \qquad \frac{\Gamma_1 \vdash_{F_{<:}SK} S_1 >: S_2 \dashv \Gamma_2 \quad \Gamma_1 \vdash_{F_{<:}SK} U_1 <: U_2 \dashv \Gamma_2}{\Gamma_1 \vdash_{F_{<:}SK} S_1 \longrightarrow U_1 <: S_2 \longrightarrow U_2 \dashv \Gamma_2}\ \text{F-Sk-Fun}$$

$$\frac{\Gamma_1 \vdash_{F_{<:}SK} S_1 >: S_2 \dashv \Gamma_2 \qquad \Gamma_1; X <: S_1 \vdash_{F_{<:}SK} U_1 <: U_2 \dashv \Gamma_2; X <: S_2}{\Gamma_1 \vdash_{F_{<:}SK} (\forall X <: S_1.U_1) <: (\forall X <: S_2.U_2) \dashv \Gamma_2}\ \text{F-Sk-All}$$

Fig. 12. Definition of strong kernel $F_{<:}$

about that type. However, it is important to keep in mind that this is possible only because we have removed the SR rule. In a calculus with the SR rule, it is possible that $\Gamma \vdash_{D_{<:}} S <: U$ is false in some context Γ that binds all free variables of S and U, but that if we further extend the context with some Γ', that can make $\Gamma; \Gamma' \vdash_{D_{<:}} S <: U$ true due to new subtyping relationships introduced in Γ' by the SR rule.

7 STRONG KERNEL AND STARE-AT SUBTYPING IN $F_{<:}$

7.1 Strong Kernel $F_{<:}$

In the previous sections, we showed that we can relax the kernel constraint of identical parameter types of dependent function types by having two typing contexts, resulting in strong kernel $D_{<:}$, and that stare-at subtyping is its decision procedure. We will now show that strong kernel and stare-at subtyping can also be applied to $F_{<:}$ and that they have the expected properties.

The definition of subtyping in strong kernel $F_{<:}$ is shown in Figure 12. Like in strong kernel $D_{<:}$, the modified F-Sk-All rule pushes two potentially different parameter types into the two contexts. The other rules merely copy both contexts to their premises.

We have shown that strong kernel $F_{<:}$ is in between kernel $F_{<:}$ and full $F_{<:}$ in expressiveness.

Theorem 47. *If* $\Gamma \vdash_{F_{<:}K} S <: U$, *then* $\Gamma \vdash_{F_{<:}SK} S <: U \dashv \Gamma$.

Theorem 48. *If* $\Gamma \vdash_{F_{<:}SK} S <: U \dashv \Gamma$, *then* $\Gamma \vdash_{F_{<:}} S <: U$.

We now provide two examples to show that strong kernel $F_{<:}$ is stricly more expressive than kernel $F_{<:}$ and strictly less expressive then full $F_{<:}$.

The following judgment is rejected by kernel $F_{<:}$ because $\top$ is not identical to X in the parameter positions, but it is admitted by strong kernel $F_{<:}$.

$$X <: \top \vdash_{F_{<:}} (\forall Y <: \top.\top) <: (\forall Y <: X : \top)$$

The following strong kernel $F_{<:}$ derivation admits this judgment:

$$\frac{\dfrac{\dots}{X <: \top \vdash_{F_{<:}SK} \top >: X \dashv X <: \top}\ \text{F-Sk-Tvar} \qquad \text{reflexivity since return types are } \top}{X <: \top \vdash_{F_{<:}SK} (\forall Y <: \top.\top) <: (\forall Y <: X : \top) \dashv X <: \top}\ \text{F-Sk-All}$$

The following judgment is admitted by full $F_{<:}$, but is rejected in strong kernel $F_{<:}$.

$$X <: \top \vdash_{F_{<:}} (\forall Y <: \top.Y) <: (\forall Y <: X.X)$$

This is because the following subderivation is rejected, because the information in the left context is insufficient to derive $Y <: X$:

$$X <: \top; Y <: \top \vdash_{F_{<:}SK} Y <: X \dashv X <: \top; Y <: X$$

In summary, the expressiveness of strong kernel $F_{<:}$ is strictly between kernel and full $F_{<:}$.

7.2 Stare-at Subtyping for $F_{<:}$

Like for $D_{<:}$, stare-at subtyping can also be defined as a decision algorithm for strong kernel $F_{<:}$ subtyping. Since type variables in $F_{<:}$ are simpler than path types in $D_{<:}$, the **Revealing**, **Upcast**, and **Downcast** relations are not needed. To turn strong kernel $F_{<:}$ into stare-at subtyping, we

(1) change all strong kernel judgments $\Gamma_1 \vdash_{F_{<:}SK} S <: U \dashv \Gamma_2$ to stare-at judgments $\Gamma_1 \gg S <: U \ll \Gamma_2$ in Figure 12, and
(2) replace the F-Sk-Tvar rule with the following rule:

$$\frac{\Gamma_1 \gg T <: U \ll \Gamma_2}{\Gamma_1; X <: T; \Gamma_1' \gg X <: Y \ll \Gamma_2} \text{ F-SA-TVAR}$$

Notice that the context on the left is truncated in the premise, resembling the **Upcast** operation. This truncation is harmless, as shown by the completeness theorem below.

Following similar methods as for $D_{<:}$, we can show that stare-at subtyping for $F_{<:}$ is sound w.r.t. full $F_{<:}$ and strong kernel $F_{<:}$.

Theorem 49. *(soundness w.r.t. full $F_{<:}$) If $\Gamma \gg S <: U \ll \Gamma$, then $\Gamma \vdash_{F_{<:}} S <: U$.*

Theorem 50. *(soundness w.r.t. strong kernel $F_{<:}$) If $\Gamma_1 \gg S <: U \ll \Gamma_2$, then $\Gamma_1 \vdash_{F_{<:}SK} S <: U \dashv \Gamma_2$.*

Both soundness theorems can be proven by straightforward induction.

The completeness theorem is easier to prove than the one for $D_{<:}$. We do need to show that stare-at subtyping for $F_{<:}$ also enjoys a strengthening property similar to Theorem 45. However, the completeness theorem can then be proven directly by induction without strengthening the statement of the theorem like in Theorems 25 and 46.

Theorem 51. *(completeness w.r.t. strong kernel $F_{<:}$) If $\Gamma_1 \vdash_{F_{<:}SK} S <: U \dashv \Gamma_2$, then $\Gamma_1 \gg S <: U \ll \Gamma_2$.*

We conclude that strong kernel and stare-at subtyping are equivalent in $F_{<:}$ like they are in $D_{<:}$. To prove termination, we define a measure analogous to the one defined for $D_{<:}$ in Definition 14:

Definition 15. *The measure M of types and contexts is defined by the following equations.*

$$M(\top) = 1$$
$$M(X) = 2$$
$$M(S \to U) = 1 + M(S) + M(U)$$
$$M(\forall X <: S.U) = 1 + M(S) + M(U)$$
$$M(\Gamma) = \sum_{X <: T \in \Gamma} M(T)$$

It is easy to check that the measure decreases in each premise of the stare-at subtyping rules.

Theorem 52. *Stare-at subtyping terminates as an algorithm.*

This also allows us to conclude that strong kernel $F_{<:}$ is decidable.

8 DISCUSSION AND RELATED WORK

8.1 Undecidability of Subtyping Reflection

In §4.6, we showed that the TRANS rule and the SR rule are equivalent in terms of expressiveness, and that $D_{<:}$ and $D_{<:}$ without the SR rule are both undecidable. We also showed that kernel $D_{<:}$ is decidable.

Kernel $D_{<:}$ applies two modifications to $D_{<:}$: it makes the parameter types in the ALL rule identical, and it removes the SR rule. It is then interesting to ask whether kernel $D_{<:}$ with the SR rule is undecidable. We conjecture that it is, but we do not know how to prove it. We expect that the proof will not be straightforward. The first problem is to identify a suitable undecidable problem to reduce from. Most well-known undecidable problems have a clear correspondence to Turing machines, which have deterministic execution. On the other hand, (kernel) $D_{<:}$ can have multiple derivations witnessing the same conclusion. Therefore, the second step would be to find a deterministic fragment of $D_{<:}$ that is still undecidable due to subtyping reflection. Indeed, discovering a deterministic fragment was also the first step of Pierce [1992]. Given the complexity of $D_{<:}$, it is hard even to find the fragment that would achieve these criteria.

This problem is interesting because it investigates the effects that follow from supporting the SR rule. Currently, in both kernel and strong kernel $D_{<:}$, the SR rule is simply removed. This is consistent with the Scala compiler, which also does not implement this rule. However, is it possible to support a fragment of this rule? We know that in $D_{<:}$, the TRANS rule and the SR rule are equivalent, so recovering a fragment of subtyping reflection recovers a fragment of transitivity as well. Moreover, some uses of the rule are not necessarily bad. Consider the following example:

$$x : \{A : \bot..\top\}; y : \{A : \bot..\top\}; z : \{A : x.A..y.A\} \vdash_{D_{<:}} x.A <: y.A$$

In this judgment, before z, $x.A$ and $y.A$ show no particular relation, but z claims that $x.A$ is a subtype of $y.A$. This example does not look as bad as other subtyping reflection like the one asserting $\top$ is a subtype of $\bot$, because it is achievable. It would be nice to find a decidable fragment that supports examples such as this. Doing so will require a careful analysis of the decidability of subtyping reflection.

8.2 Related Work

There has been much work related to proving undecidability under certain settings of subtyping. Pierce [1992] presented a chain of reductions from two counter machines (TCM) to $F_{<:}$ and showed $F_{<:}$ undecidable. Kennedy and Pierce [2007] investigated a nominal calculus with variance, modelling the situations in Java, C# and Scala, and showed that this calculus is undecidable due to three factors: contraviarant generics, large class hierarchies, and multiple inheritance. Wehr and Thiemann [2009] considered two calculi with existential types, $\mathcal{EX}_{impl}$ and $\mathcal{EX}_{uplo}$, and proved both to be undecidable. Moreover, in $\mathcal{EX}_{uplo}$, each type variable has either upper or lower bounds but not both, so this calculus is related to $D_{<:}$, but since no variable has both lower and upper bounds, it does not expose the subtyping reflection phenomenon. Grigore [2017] proved Java generics undecidable by reducing Turing machines to a fragment of Java with contravariance.

So far, work on the *DOT* calculi mainly focused on soundness proofs [Amin et al. 2016; Rapoport et al. 2017; Rompf and Amin 2016]. Nieto [2017] presented step subtyping as a partial algorithm for *DOT* subtyping. In this paper, we have shown that the fragment of $D_{<:}$ decided by step subtyping is kernel $D_{<:}$. Aspinall and Compagnoni [2001] showed a calculus with dependent types and subtyping that is decidable due to the lack of a $\top$ type. Greenman et al. [2014] identified the Material-Shape Separation. This separation describes two different usages of interfaces, and as long as no interface is used in both ways, the type checking problem is decidable by a simple algorithm. Mackay et al.

[2020] defined Wyv_{core}, a calculus with path dependent types and recursive types, and provided subtyping decision algorithms by following the Material-Shape Separation. Path dependent types in Wyv_{core} do not have both lower and upper bounds as in $\mathcal{EX}_{uplo}$, so Wyv_{core} is weaker than *DOT* in expressive power. This paper also presented a decidable calculus, Wyv_{fix}, which employs the same method of subtyping with two contexts as in strong kernel $D_{<:}$.

The undecidability proof in this paper has been mechanized in Agda. There are other fundamental results on formalizing proofs of undecidability. Forster et al. [2018] mechanized undecibility proofs of various well-known undecidable problems, including the post correspondence problem (PCP), string rewriting (SR) and the modified post correspondence problem. Their proofs are based on Turing machines. In contrast, Forster and Smolka [2017] used a call-by-value lambda calculus as computational model. Forster and Larchey-Wendling [2019] proved undecibility of intuitionistic linear logic by reducing from PCP.

9 CONCLUSION

We have studied the decidability of typing and subtyping of the $D_{<:}$ calculus and several of its fragments. We first presented a counterexample showing that the previously proposed mapping from $F_{<:}$ to $D_{<:}$ cannot be used to prove undecidability of $D_{<:}$. We then discovered a normal form for $D_{<:}$ and proved its equivalence with the original $D_{<:}$ formulation. We used the normal form to prove $D_{<:}$ subtyping and typing undecidable by reductions from $F^{-}_{<:}$. We defined a kernel version of $D_{<:}$ by removing the subtyping reflection subtyping rule and restricting the subtyping rule for dependent function types to equal parameter types, as in kernel $F_{<:}$. We proved kernel $D_{<:}$ decidable, and showed that it is exactly the fragment of $D_{<:}$ that is handled by the step subtyping algorithm of Nieto [2017]. We defined strong kernel $D_{<:}$, a decidable fragment of $D_{<:}$ that is strictly in between kernel $D_{<:}$ and full $D_{<:}$ in terms of expressiveness, and in particular permits subtyping comparison between parameter types of dependent function types. This allows us to handle type aliases gracefully within the subtyping relation. Finally, we proposed stare-at subtyping as an algorithm for deciding subtyping in strong kernel $D_{<:}$. We have mechanized the proofs of our theoretical results using proof assistants, Agda for proofs of undecidability and Coq for correctness proofs of the decision algorithms for the decidable variants.

ACKNOWLEDGMENTS

We are grateful to anonymous reviewers for their constructive comments. This research was supported by the Natural Sciences and Engineering Research Council of Canada.

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*.

Nada Amin and Tiark Rompf. 2017. Type soundness proofs with definitional interpreters. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017*. ACM, 666–679. http://dl.acm.org/citation.cfm?id=3009866

Nada Amin, Tiark Rompf, and Martin Odersky. 2014. Foundations of Path-dependent Types. In *Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages & Applications (OOPSLA '14)*. ACM, New York, NY, USA, 233–249. https://doi.org/10.1145/2660193.2660216

David Aspinall and Adriana Compagnoni. 2001. Subtyping dependent types. *Theoretical Computer Science* 266, 1 (2001), 273 – 309. https://doi.org/10.1016/S0304-3975(00)00175-4

Hendrik Pieter Barendregt. 1985. *The lambda calculus - its syntax and semantics*. Studies in logic and the foundations of mathematics, Vol. 103. North-Holland.

L. Cardelli, S. Martini, J.C. Mitchell, and A. Scedrov. 1994. An Extension of System F with Subtyping. *Information and Computation* 109, 1 (1994), 4 – 56. https://doi.org/10.1006/inco.1994.1013

Luca Cardelli and Peter Wegner. 1985. On Understanding Types, Data Abstraction, and Polymorphism. *ACM Comput. Surv.* 17, 4 (Dec. 1985), 471–523. https://doi.org/10.1145/6041.6042

Adam Chlipala. 2013. *Certified Programming with Dependent Types: A Pragmatic Introduction to the Coq Proof Assistant.* The MIT Press.

Pierre-Louis Curien and Giorgio Ghelli. 1990. Coherence of Subsumption. In *CAAP '90, 15th Colloquium on Trees in Algebra and Programming, Copenhagen, Denmark, May 15-18, 1990, Proceedings (Lecture Notes in Computer Science)*, André Arnold (Ed.), Vol. 431. Springer, 132–146. https://doi.org/10.1007/3-540-52590-4_45

Yannick Forster, Edith Heiter, and Gert Smolka. 2018. Verification of PCP-Related Computational Reductions in Coq. In *Interactive Theorem Proving - 9th International Conference, ITP 2018, Oxford, UK, July 9-12, 2018 (LNCS 10895)*. Springer, 253–269. Preliminary version appeared as arXiv:1711.07023.

Yannick Forster and Dominique Larchey-Wendling. 2019. Certified Undecidability of Intuitionistic Linear Logic via Binary Stack Machines and Minsky Machines. In *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP 2019)*. ACM, New York, NY, USA, 104–117. https://doi.org/10.1145/3293880.3294096

Yannick Forster and Gert Smolka. 2017. Weak Call-by-Value Lambda Calculus as a Model of Computation in Coq. In *Interactive Theorem Proving - 8th International Conference, ITP 2017, Brasília, Brazil, September 26-29, 2017, Proceedings (Lecture Notes in Computer Science)*, Mauricio Ayala-Rincón and César A. Muñoz (Eds.), Vol. 10499. Springer, 189–206. https://doi.org/10.1007/978-3-319-66107-0_13

Ben Greenman, Fabian Muehlboeck, and Ross Tate. 2014. Getting F-bounded polymorphism into shape. In *ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '14, Edinburgh, United Kingdom - June 09 - 11, 2014*, Michael F. P. O'Boyle and Keshav Pingali (Eds.). ACM, 89–99. https://doi.org/10.1145/2594291.2594308

Radu Grigore. 2017. Java generics are turing complete. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017*. 73–85. http://dl.acm.org/citation.cfm?id=3009871

Zhong Sheng Hu. 2019. *Decidability and Algorithmic Analysis of Dependent Object Types (DOT).* Master's thesis. University of Waterloo. http://hdl.handle.net/10012/14964

Andrew Kennedy and Benjamin C. Pierce. 2007. On Decidability of Nominal Subtyping with Variance. In *International Workshop on Foundations and Developments of Object-Oriented Languages (FOOL/WOOD)* (international workshop on foundations and developments of object-oriented languages (fool/wood) ed.). https://www.microsoft.com/en-us/research/publication/on-decidability-of-nominal-subtyping-with-variance/

Julian Mackay, Alex Potanin, Lindsay Groves, and Jonathan Aldrich. 2020. Decidable Subtyping for Path Dependent Types. In *Proceedings of the 47th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2020, New Orleans, USA, January 22-24, 2020*. https://doi.org/10.1145/3371134

John C. Martin. 1997. *Introduction to Languages and the Theory of Computation* (2nd ed.). McGraw-Hill Higher Education.

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

Frank Pfenning. 2000. Structural Cut Elimination: I. Intuitionistic and Classical Logic. *Information and Computation* 157, 1 (2000), 84 – 141. https://doi.org/10.1006/inco.1999.2832

Benjamin C. Pierce. 1991. *Programming with intersection types and bounded polymorphism.* Ph.D. Dissertation. Carnegie Mellon University.

Benjamin C. Pierce. 1992. Bounded Quantification is Undecidable. In *Proceedings of the 19th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '92)*. ACM, New York, NY, USA, 305–315. https://doi.org/10.1145/143165.143228

Benjamin C. Pierce. 1997. *Bounded Quantification with Bottom.* Technical Report 492. Computer Science Department, Indiana University.

Benjamin C. Pierce. 2002. *Types and Programming Languages* (1st ed.). The MIT Press.

Benjamin C. Pierce. 2004. *Advanced Topics in Types and Programming Languages.* The MIT Press.

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

Agda Team. 2019. Agda 2.5.4.2.

The Coq Development Team. 2018. The Coq Proof Assistant, version 8.8.2. https://doi.org/10.5281/zenodo.1219885

Stefan Wehr and Peter Thiemann. 2009. On the Decidability of Subtyping with Bounded Existential Types. In *Proceedings of the 7th Asian Symposium on Programming Languages and Systems (APLAS '09)*. Springer-Verlag, Berlin, Heidelberg, 111–127. https://doi.org/10.1007/978-3-642-10672-9_10