Type-Level Programming with Match Types

OLIVIER BLANVILLAIN, EPFL, Switzerland
JONATHAN BRACHTHÄUSER, University of Tübingen, Germany
MAXIME KJAER, EPFL, Switzerland
MARTIN ODERSKY, EPFL, Switzerland

Type-level programming is becoming more and more popular in the realm of functional programming. However, the combination of type-level programming and subtyping remains largely unexplored in practical programming languages. This paper presents *match types*, a type-level equivalent of pattern matching. Match types integrate seamlessly into programming languages with subtyping and, despite their simplicity, offer significant additional expressiveness. We formalize the feature of match types in a calculus based on System $F_{<:}$ and prove its soundness. We practically evaluate our system by implementing match types in the Scala 3 reference compiler, thus making type-level programming readily available to a broad audience of programmers.

TECHNICAL REPORT

This technical report extends [Blanvillain et al. 2022] with appendices containing soundness proofs for System FM and System FMB.

1 INTRODUCTION

There is a growing interest in using *type-level computation* to increase the expressivity of type systems, express additional constraints on the type level, and thereby improve the safety of general-purpose software. What used to be an exclusive feature of dependently typed languages is slowly becoming accessible to everyday programmers. GHC Haskell has been at the forefront of making this a reality and already provides several extensions to support type-level programming. While Haskell is certainly not the only language moving towards dependent types, the trend seems to be limited to pure functional programming languages.

We believe that type-level programming is not necessarily incompatible with other programming paradigms and that the current division exists mainly due to a lack of attention from the research community. Unfortunately, most of the existing research conducted in this domain is not directly applicable to languages with *subtyping*. Although the combination of subtyping and type-level programming has been studied extensively on the theoretical side, through the means of dependently typed systems [Aspinall 1995; Courant 2003; Hutchins 2010; Stone and Harper 2000; Yang and Oliveira 2017; Zwanenburg 1999], the practical side remains largely unexplored.

One notable exception is the TypeScript language, which recently introduced a new feature called *conditional type*, a type-level ternary operator based on subtyping. A conditional type, written `S extends T ? Tt : Tf`, reduces to `Tt` when `S` is a subtype of `T`, to `Tf` when `S` is not a subtype of `T`, and is left unreduced when types variables do not allow to draw a conclusion. Unfortunately, the algorithm used to reduce such conditional types is both *unsound* and *incomplete*. Despite the unsoundness (discussed in Subsection 6.5), the addition of conditional types to TypeScript illustrates the practical need and timeliness of this feature.

This paper presents an alternative construct for type-level programming based on subtyping, which we call *match types*. As the name suggests, match types allow programmers to express types that perform pattern matching on types:

Authors' addresses: Olivier Blanvillain, EPFL, Switzerland, olivier.blanvillain@epfl.ch; Jonathan Brachthäuser, University of Tübingen, Germany, jonathan.brachthaeuser@uni-tuebingen.de; Maxime Kjaer, EPFL, Switzerland, maxime.kjaer@alumni.epfl.ch; Martin Odersky, EPFL, Switzerland, martin.odersky@epfl.ch.

```
type Elem[X] = X match {
  case String => Char
  case List[t] => Elem[t]
  case Any => X
}
```

The example, which we explain in detail in Section 2, defines the type `Elem` by matching on the type parameter `X`. We have implemented match types in the latest version of Scala, an industry-grade, production-ready compiler. Match types have received a great interest from the Scala community, and are already in active use.

In this paper, we explore the theoretical foundations of match types through the lens of a type system which extends System $F_{<:}$ with pattern matching at the term and type level. Our formalization serves two purposes: first, it gives a clear view on *how* we integrated match types in Scala's type system and precisely describes the changes needed on the subtyping relation to make this integration possible. Second, thanks to a type safety proof based on the standard progress and preservation theorems, it gives confidence that the design of match types is sensible and our implementation is sound.

Conditional types provide *concrete evidence* that our results are valuable beyond the context of Scala. Our results are directly applicable to TypeScript's type system and provide a clear path to fixing the unsoundness introduced by conditional types. Furthermore, we hope that match types can be useful as a reference for future designs of type-level programming features for languages with subtyping.

In summary, this paper makes the following contributions:

- We introduce programming with match types in Scala by means of an example and highlight the interaction of type-level programming and subtyping (Section 2).
- We formalize match types in the self-contained calculus System FM and prove it sound, providing a theoretical basis of our implementation (Section 3). The paper is accompanied by a mechanization of System FM, including proofs of progress and preservation.
- We describe our implementation of match types in the Scala compiler, discuss challenges, and relate the implementation to our formalization (Section 4).
- We evaluate match types in a case study, presenting a type-safe version of the NumPy library (Section 5).
- We motivate the design of our formalization relative to prior work, we review the extensive related work on type families in Haskell, and discuss the unsoundness of conditional types in TypeScript (Section 6).

2 OVERVIEW

In this section, we offer a brief introduction of match types in Scala by inspecting the example from the previous section in more detail:

```
type Elem[X] = X match {
  case String => Char
  case List[t] => Elem[t]
  case Any => X
}
```

This example defines a type `Elem` parametrized by one type parameter `X`. The right-hand side is defined in terms of a match on the type parameter – a *match type*. A match type reduces to one of

its right-hand sides, depending on the type of its scrutinee. For example, the above type reduces as follows:

```
Elem[String] =:= Char
Elem[Int] =:= Int
Elem[List[Int]] =:= Int
```

Here we use `S =:= T` to denote type equality between the two types `S` and `T`, witnessed by mutual subtyping. To reduce a match type, the scrutinee is compared to each pattern, one after the other, using *subtyping*. For example, although `String` is a subtype of both `String` and `Any` (the top of Scala's subtyping lattice), `Elem[String]` reduces to `Char` because the corresponding case appears first.

When the scrutinee type is a `List`, the match type `Elem` is defined recursively on the element type of the list. Hence, in our example `Elem[List[Int]]` first reduces to the type `Elem[Int]`, and eventually to the type `Int`.

2.1 A Lightweight Form of Dependent Typing

Match types enable a lightweight form of dependent typing, since a term-level pattern matching expression can be typed accordingly at the type level as a match type. Consider the following function definition:

```
def elem[X](x: X): Elem[X] = x match {
  case x: String => x.charAt(0)
  case x: List[t] => elem(x.head)
  case x: Any => x
}
```

This definition is well-typed because the match expression in `elem`'s body has the exact same scrutinee and pattern types as `Elem[X]` (the function's return type).

Thanks to Scala's type inference, a call to the `elem` function can have a result type that *depends* on a term-level parameter. For instance, in the expression `elem(1)`, the Scala compiler infers the singleton type `X = 1` for `elem`'s type parameter. This expression thus has type `Elem[1]`, which reduces to `Int` (via `Elem`'s third case). Similarly, in `elem(x)`, the compiler infers the singleton type `X = x.type` and the expression has type `Elem[x.type]`, which might reduce further at the callsite depending on `x`'s type.

In both examples, singleton types create a dependency between a type parameter and a term, which, by transitivity, results in a lightweight form of dependent typing, that is, a dependency between a term parameter and a function's result type.

2.2 Disjointness

Our design of match types induces an additional constraint on match type reduction: the scrutinee type must be known to be *disjoint* with all type patterns preceding the matching case. Informally, disjointness means that two types have no shared inhabitants.

The necessity for disjointness is best illustrated with an example. Consider `Seq[Int]`, the type of integer sequences. `Elem[Seq[Int]]` does not reduce:

(1) `Elem`'s first case does not apply because `Seq[Int]` is not a subtype of `String`.
(2) `Elem`'s second case does not apply because `Seq[Int]` is not a subtype of `List[Int]` (lists are sequences, but not the other way around).
(3) `Elem`'s third case is *not considered* because `Seq[Int]` and `List[Int]` are not disjoint.

Syntax

t ::=		T ::=	
	x — *variable*		X — *type variable*
	$\lambda x : T.\ t$ — *abstraction*		$T \to T$ — *type of functions*
	$\lambda X <: T.\ t$ — *type abstraction*		$\forall X <: T.\ T$ — *universal type*
	$t\ t$ — *application*		Top — *maximum type*
	$t\ T$ — *type application*		C — *class*
	new C — *constructor call*		$\{new\ C\}$ — *constructor singleton*
	$t\ match\{\overline{x : C \Rightarrow t}\}or\ t$ — *match expr.*		$T\ match\{\overline{T \Rightarrow T}\}or\ T$ — *match type*
v ::=		Γ ::=	
	$\lambda x : T.\ t$ — *abstraction*		Ø — *empty context*
	$\lambda X <: T.\ t$ — *type abstraction*		$\Gamma, x : T$ — *term binding*
	new C — *constructor call*		$\Gamma, X <: T$ — *type binding*

Evaluation

$$\frac{t_1 \longrightarrow t_1'}{t_1\ t_2 \longrightarrow t_1'\ t_2}\ \text{(E-App1)} \qquad \frac{t_2 \longrightarrow t_2'}{v_1\ t_2 \longrightarrow v_1\ t_2'}\ \text{(E-App2)} \qquad \frac{t_1 \longrightarrow t_1'}{t_1\ T_2 \longrightarrow t_1'\ T_2}\ \text{(E-TApp)}$$

$$(\lambda x : T_{11}.\ t_{12})\ v_2 \longrightarrow [x \mapsto v_2]t_{12}\ \text{(E-AppAbs)} \qquad (\lambda X <: T_{11}.\ t_{12})\ T_2 \longrightarrow [X \mapsto T_2]t_{12}\ \text{(E-TAppTAbs)}$$

$$\frac{t_s \longrightarrow t_s'}{t_s\ match\{x_i : C_i \Rightarrow t_i\}or\ t_d \longrightarrow t_s'\ match\{x_i : C_i \Rightarrow t_i\}or\ t_d}\ \text{(E-Match1)}$$

$$\frac{(C, C_n) \in \Psi \quad \forall m < n.\ (C, C_m) \notin \Psi}{new\ C\ match\{x_i : C_i \Rightarrow t_i\}or\ t_d \longrightarrow [x_n \mapsto new\ C]t_n}\ \text{(E-Match2)}$$

$$\frac{\forall m.\ (C, C_m) \notin \Psi}{new\ C\ match\{x_i : C_i \Rightarrow t_i\}or\ t_d \longrightarrow t_d}\ \text{(E-Match3)}$$

$$(\lambda x : T.\ t)\ match\{x_i : C_i \Rightarrow t_i\}or\ t_d \longrightarrow t_d\ \text{(E-Match4)}$$

$$(\lambda X <: T.\ t)\ match\{x_i : C_i \Rightarrow t_i\}or\ t_d \longrightarrow t_d\ \text{(E-Match5)}$$

Fig. 1. System FM syntax and evaluation rules for a given set of classes C with class inheritance Ψ and class disjointness Ξ. Highlights correspond to additions to System $F_{<:}$.

Therefore, the reduction algorithm gets stuck on the second case and the overall type is irreducible. Without disjointness, `Elem[Seq[Int]]` would reduce to `Seq[Int]` (via `Elem`'s third case), which would be unsound. For example, the expression `elem[Seq[Int]](List(1,2,3))` would have type `Seq[Int]`, but evaluates to the integer `1`.

Subsection 3.2 revisits this counterexample in a formal setting. Subsection 4.1 discusses our implementation of disjointness in the Scala compiler.

Subtyping

$$\Gamma \vdash S <: S \quad \text{(S-Refl)} \qquad \Gamma \vdash S <: \mathrm{Top} \quad \text{(S-Top)}$$

$$\Gamma \vdash \{new\ C\} <: C \quad \text{(S-Sin)} \qquad \frac{(C_1, C_2) \in \Psi}{\Gamma \vdash C_1 <: C_2} \quad \text{(S-Psi)}$$

$$\frac{\Gamma \vdash S <: U \quad \Gamma \vdash U <: T}{\Gamma \vdash S <: T} \quad \text{(S-Trans)} \qquad \frac{\Gamma \vdash T_1 <: S_1 \quad \Gamma \vdash S_2 <: T_2}{\Gamma \vdash S_1 \to S_2 <: T_1 \to T_2} \quad \text{(S-Arrow)}$$

$$\frac{X <: T \in \Gamma}{\Gamma \vdash X <: T} \quad \text{(S-TVar)} \qquad \frac{\Gamma, X <: U_1 \vdash S_2 <: T_2}{\Gamma \vdash (\forall X <: U_1.\ S_2) <: (\forall X <: U_1.\ T_2)} \quad \text{(S-All)}$$

$$\frac{\Gamma \vdash T_s <: S_n \quad \forall m < n.\ \Gamma \vdash \mathrm{disj}(T_s, S_m)}{\Gamma \vdash T_s\ match\{S_i \Rightarrow T_i\}or\ T_d =:= T_n} \quad \text{(S-Match1/2)}$$

$$\frac{\forall n.\ \Gamma \vdash \mathrm{disj}(T_s, S_n)}{\Gamma \vdash T_s\ match\{S_i \Rightarrow T_i\}or\ T_d =:= T_d} \quad \text{(S-Match3/4)}$$

$$\frac{\Gamma \vdash S_s <: T_s \quad \Gamma \vdash S_d <: T_d \quad \forall n.\ \Gamma \vdash S_n <: T_n}{\Gamma \vdash S_s\ match\{U_i \Rightarrow S_i\}or\ S_d <: \\ T_s\ match\{U_i \Rightarrow T_i\}or\ T_d} \quad \text{(S-Match5)}$$

Typing

$$\frac{\Gamma, x : T_1 \vdash t_2 : T_2}{\Gamma \vdash \lambda x : T_1.\ t_2 : T_1 \to T_2} \quad \text{(T-Abs)} \qquad \frac{\Gamma \vdash t_1 : T_{11} \to T_{12} \quad \Gamma \vdash t_2 : T_{11}}{\Gamma \vdash t_1\ t_2 : T_{12}} \quad \text{(T-App)}$$

$$\frac{\Gamma, X <: U_1 \vdash t_2 : T_2}{\Gamma \vdash \lambda X <: U_1.\ t_2 : \forall X <: U_1.\ T_2} \quad \text{(T-TAbs)} \qquad \frac{\Gamma \vdash t_1 : \forall X <: T_{11}.\ T_{12} \quad \Gamma \vdash T_2 <: T_{11}}{\Gamma \vdash t_1\ T_2 : [X \mapsto T_2]T_{12}} \quad \text{(T-TApp)}$$

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T} \quad \text{(T-Var)} \qquad \frac{\Gamma \vdash t : S \quad \Gamma \vdash S <: T}{\Gamma \vdash t : T} \quad \text{(T-Sub)} \qquad \Gamma \vdash new\ C : \{new\ C\} \quad \text{(T-Class)}$$

$$\frac{\Gamma \vdash t_s : T_s \quad \Gamma, x_i : C_i \vdash t_i : T_i \quad \Gamma \vdash t_d : T_d}{\Gamma \vdash t_s\ match\{x_i : C_i \Rightarrow t_i\}or\ t_d : T_s\ match\{C_i \Rightarrow T_i\}or\ T_d} \quad \text{(T-Match)}$$

Disjointness

$$\frac{(C_1, C_2) \in \Xi}{\Gamma \vdash \mathrm{disj}(C_1, C_2)} \quad \text{(D-Xi)} \qquad \frac{(C_1, C_2) \notin \Psi}{\Gamma \vdash \mathrm{disj}(\{new\ C_1\}, C_2)} \quad \text{(D-Psi)}$$

$$\frac{\Gamma \vdash S <: U \quad \Gamma \vdash \mathrm{disj}(U, T)}{\Gamma \vdash \mathrm{disj}(S, T)} \quad \text{(D-Sub)}$$

$$\Gamma \vdash \mathrm{disj}(T_1 \to T_2, C) \quad \text{(D-Arrow)}$$

$$\Gamma \vdash \mathrm{disj}(\forall X <: T_1.\ T_2, C) \quad \text{(D-All)}$$

Fig. 2. System FM type system for a given set of classes C with class inheritance Ψ and class disjointness Ξ. Highlights correspond to additions to System $F_{<:}$.

3 FORMALIZATION

In this section, we formally present System FM, an extension of System $F_{<:}$ [Cardelli et al. 1994] with pattern matching, opaque classes, and match types. Figure 1 defines FM's syntax and evaluation relation. Figure 2 defines FM's type system, composed of three relations: typing, subtyping, and type disjointness. We discuss differences to System $F_{<:}$ in the following subsections (Subsection 3.1 and 3.2). In Subsection 3.3, we outline a proof of type safety for System FM. In Subsection 3.4, we present an extension of System FM with support for binding pattern variables in type patterns.

3.1 Classes

System FM is parametrized by a set of classes C with class inheritance Ψ and class disjointness Ξ. The class inheritance forms a partial order on C, that is, it is reflexive, antisymmetric and transitive. The class disjointness is symmetric relation over C. Subtyping between classes is dictated by Ψ via the S-Psi rule.

The inheritance and disjointness parameters can be understood as a representation of a hierarchy of Scala traits and classes. For example, `trait C1; class C2 extends C1` is represented in FM as $\Psi = \{(C_1, C_2)\};\ \ \Xi = \{\}$. This representation also models the fact that certain types cannot possibly have common instances. For example, `class C3; class C4` is represented as $\Psi = \{\};\ \ \Xi = \{(C_3, C_4), (C_4, C_3)\}$, since Scala disallows multiple class inheritance. Inheritance and disjointness must be consistent in the sense that $(A, B) \in \Xi$ implies that there is no class C such that $(C, A) \in \Psi$ and $(C, B) \in \Psi$.

Each class in C gives rise to a constructor (written *new* C), a type (written C), and a constructor singleton type (written $\{new\,C\}$). The type C denotes all values that inherit C, while the constructor singleton type $\{new\,C\}$ denotes a single value: C's constructor call.

This parametric approach allows us to model class inheritance as found in object-oriented languages without the need for dedicated syntax for classes and data type definitions. Although our approach might appear simplistic, it can easily model advanced object-oriented features such as multiple inheritance. We discuss the encoding of Scala's types into System FM in Subsection 4.1.

Our type system refers to classes by names and therefore mixes structural and nominal types. Names are useful to give a direct correspondence between runtime tags and compile-time types. As we will see, runtime tags are essential to runtime type testing and play a central role in the evaluation of pattern matching.

3.2 Matches

System FM supports both pattern matching on the term level (*match expressions*) as well as on the type-level (*match types*). Matches, both on terms and on types, are composed of a scrutinee, a list of cases and a default expression/type. Each case consists of a *type test* and a corresponding expression/type. At the term level, a type test consists of an inheritance check against a particular class (this is also known as a typecase [Abadi et al. 1991]). At the type level, a type test corresponds to a subtyping test with a particular type. This disparity reflects the difference between runtime, where type tests are implemented using class tables, and compile time, where types are compared using the type system in its full extent. We discuss the representation of Scala types at runtime in Subsection 4.6.

Throughout this paper, we use the abbreviated syntax $t_s\ \ match\{x_i : C_i \Rightarrow t_i\}\ or\ t_d$ to denote an arbitrary number of cases, that is, $\exists n \in \mathbb{N}.\ t_s\ \ match\{x_1 : C_1 \Rightarrow t_1; \ldots; x_n : C_n \Rightarrow t_n\}\ or\ t_d$.

Match expressions and match types are related by the T-Match typing rule. This rule operates by typing each component of a match expression to then assemble the corresponding match type.

Example A. For example, given two disjoint classes A and B, and an empty class inheritance ($\Psi = \mathrm{Id}, \Xi = \{(\mathrm{A},\mathrm{B}),(\mathrm{B},\mathrm{A})\}$); the following function:
$\mathrm{f} = \lambda \mathrm{X} <: \mathrm{Top}.\ \lambda \mathrm{x} : \mathrm{X}.\ \mathrm{x}\ \mathit{match}\{\mathrm{a} : \mathrm{A} \Rightarrow \mathrm{foo}; \mathrm{b} : \mathrm{B} \Rightarrow \mathrm{bar}\}\mathit{or}\,\mathrm{buzz}$
can be typed precisely as
$\mathrm{f} : \forall \mathrm{X} <: \mathrm{Top}.\ \mathrm{X} \to \mathrm{X}\ \mathit{match}\{\mathrm{A} \Rightarrow \mathrm{Foo}; \mathrm{B} \Rightarrow \mathrm{Bar}\}\mathit{or}\,\mathrm{Buzz}$
where foo, bar and buzz are expressions with types Foo, Bar and Buzz, respectively.

The cases of a match expression are evaluated *sequentially*: the scrutinee is checked using the type test of each case, one after the other. The overall expression reduces to the expression that corresponds to the first successful type test (E-Match2). When no type test succeeds, the match evaluates to its default expression (E-Match3/4/5). For instance, given the function f defined in Example A, the expression $(\mathrm{f}\ \mathrm{A}\ (\mathit{new}\ \mathrm{A}))$ evaluates to foo and $(\mathrm{f}\ \mathrm{B}\ (\mathit{new}\ \mathrm{B}))$ to bar.

Match Type Reduction. The subtyping relation contains five rules for match type reduction, S-Match1/2/3/4/5. These rules are defined in pairs using the =:= shorthand notation, where $\mathrm{S} =:= \mathrm{T}$ means that S and T are in a mutual subtyping relation. More precisely, S-Match1/2 in Figure 2 corresponds to two typing rules with identical premises and symmetrical conclusion, and the same goes for S-Match3/4.

The typing rules for match type reduction are best explained as generalizations of the evaluation relation. Given a match type $\mathrm{M} = \mathrm{T}_s\ \mathit{match}\{\mathrm{C}_i \Rightarrow \mathrm{T}_i\}\mathit{or}\,\mathrm{T}_d$, M reduces to T_i if and only if, for every value t_s in T_s, the term level expression $\mathrm{t}_s\ \mathit{match}\{\mathrm{x}_i : \mathrm{C}_i \Rightarrow \mathrm{t}_i\}\mathit{or}\,\mathrm{t}_d$ evaluates to t_i.

The S-Match1/2 rules correspond to the evaluation of a match expression to its nth case (E-Match2):

$$\frac{\Gamma \vdash \mathrm{T}_s <: \mathrm{S}_n \qquad \forall m < n.\ \Gamma \vdash \mathrm{disj}(\mathrm{T}_s, \mathrm{S}_m)}{\Gamma \vdash \mathrm{T}_s\ \mathit{match}\{\mathrm{S}_i \Rightarrow \mathrm{T}_i\}\mathit{or}\,\mathrm{T}_d =:= \mathrm{T}_n} \qquad \text{(S-Match1/2)}$$

The first premise ensures that the nth case type test will succeed for every possible value in the scrutinee type T_s. Conversely, the second premise is a disjointness judgment, which ensures that no value in the scrutinee type would result in a successful type test for cases prior to the nth case. The S-Match3/4 rules correspond to an evaluation to the default case (E-Match2), and require disjointness between the scrutinee type and each type test type:

$$\frac{\forall n.\ \Gamma \vdash \mathrm{disj}(\mathrm{T}_s, \mathrm{S}_n)}{\Gamma \vdash \mathrm{T}_s\ \mathit{match}\{\mathrm{S}_i \Rightarrow \mathrm{T}_i\}\mathit{or}\,\mathrm{T}_d =:= \mathrm{T}_d} \qquad \text{(S-Match3/4)}$$

Disjointness between two classes can be concluded directly using the D-Xi rule which uses the class disjointness Ξ. Likewise, disjointness between a constructor singleton type and a class can be concluded directly by inspecting the class inheritance Ψ (D-Psi). Function types and universal types are disjoint from classes as they are inhabited by different values (D-Arrow, D-All) and thus will never match. The last disjointness rule, D-Sub, states that if U and T are disjoint, then all subtypes of U are also disjoint with T.

Example B. We continue developing Example A by showing how match type reduction rules can be used to conclude that $(\mathrm{f}\ \mathrm{B}\ (\mathit{new}\ \mathrm{B}))$ has type Bar. Using T-TApp and T-App, the expression can be typed as follows:
$\mathrm{f}\ \mathrm{B}\ (\mathit{new}\ \mathrm{B}) : \mathrm{B}\ \mathit{match}\{\mathrm{A} \Rightarrow \mathrm{Foo}; \mathrm{B} \Rightarrow \mathrm{Bar}\}\mathit{or}\,\mathrm{Buzz}$
Since our example assumes an empty class inheritance and $(\mathrm{A},\mathrm{B}) \in \Xi$, the S-Match1 rule gives:

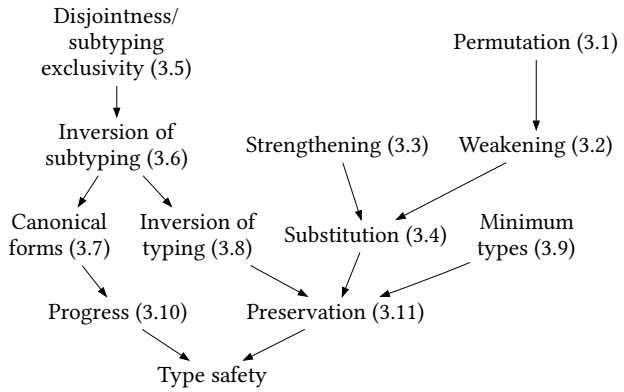

Fig. 3. Structure of the type safety proof. Arrows represent implications between lemmas and theorems.

$\emptyset \vdash \mathrm{B}\ \mathit{match}\{\mathrm{A} \Rightarrow \mathrm{Foo}; \mathrm{B} \Rightarrow \mathrm{Bar}\}\ \mathit{or}\ \mathrm{Buzz} <: \mathrm{Bar}$
Finally, using T-Sub we get $(\mathrm{f}\ \mathrm{B}\ (\mathit{new}\ \mathrm{B})) : \mathrm{Bar}$.

Subtyping and Disjointness. One might wonder what happens if we simplify the match type reduction rules by replacing premises of the form $\Gamma \vdash \mathrm{disj}(\mathrm{T}, \mathrm{C})$ by seemingly equivalent premises of the form $(\mathrm{T}, \mathrm{C}) \notin \Psi$. Unfortunately, the resulting system would be unsound, which can be demonstrated with a counterexample. Let us assume the function f defined in Example A, adding a new class E with $\Psi = \{(\mathrm{E}, \mathrm{A}), (\mathrm{E}, \mathrm{B})\}$ and $\Xi = \{\}$. Now, consider the term $(\mathrm{f}\ \mathrm{B}\ (\mathit{new}\ \mathrm{E}))$. Since we have $\emptyset \vdash \mathrm{E} <: \mathrm{B}$, this function application is well-typed and, given that $(\mathrm{E}, \mathrm{A}) \in \Psi$, evaluates to foo. The term-level and type-level reductions are inconsistent! The unsoundness arises when using $(\mathrm{B}, \mathrm{A}) \notin \Psi$ with the modified S-Match1 rule to wrongly conclude that $(\mathrm{f}\ \mathrm{B}\ (\mathit{new}\ \mathrm{E}))$ has type Bar. This would result in an inconsistency between types $(\mathrm{e} : \mathrm{Bar})$ and evaluation $(\mathrm{e} \longrightarrow \mathrm{foo})$, and violate type soundness. In System FM, the match type obtained when typing $(\mathrm{f}\ \mathrm{B}\ (\mathit{new}\ \mathrm{E}))$ does not reduce since the scrutinee type B is neither disjoint with, nor a subtype of the first pattern type test A. In this case, the unreduced match type is assigned "as is". Unreduced types can appear as the result of programming error, but can also be due to the local irreducibility of a match type. For instance, the body of f is typed with an unreduced type, as shown in Example A, but that type can later become reducible depending on type variable instantiations.

3.3 Type Safety

We show the type safety of System FM through the usual progress and preservation theorems. This section provides an overview of the proof structure and states the involved lemmas and theorems. Detailed pen-and-paper proofs are available in the appendix of this report. A Coq mechanization of our proofs is available in the supplementary material of this report [Blanvillain et al. 2021a]. Our mechanization uses the locally nameless representation by Aydemir et al. [2008] to model variable bindings, and a simplified representation of match types with exactly one case per match. Matches with multiple cases can be expressed by nesting match types in default cases.

Figure 3 gives an overview of the proof structure by showing implications between the various lemmas and theorems. The basic structure resembles that of System $\mathrm{F}_{<:}$'s standard safety proof from [Pierce 2002]. We continue our presentation by introducing the lemmas and theorems used in our type safety proof.

$$\frac{\vdots}{\Gamma \vdash S =:= T}\ (\text{S-Match1/2}) \qquad \frac{\vdots}{\Gamma \vdash S =:= T}\ (\text{S-Match3/4}) \qquad \frac{\Gamma \vdash S \rightleftharpoons U \quad \Gamma \vdash U \rightleftharpoons T}{\Gamma \vdash S \rightleftharpoons T}$$

Fig. 4. Definition of the auxiliary relation $\rightleftharpoons$, used to state inversion of subtyping (Lemma 3.6).

Preliminary Lemmas. Our proof begins with preliminary technical lemmas: Lemma 3.1 (Permutation), Lemma 3.2 (Weakening), Lemma 3.3 (Strengthening), Lemma 3.4 (Substitution). We omit stating these lemmas as they are entirely standard yet relatively lengthy given that they span the three relations of our system: typing, subtyping, and disjointness. As usual, their proofs follow by mutual inductions on derivations.

Disjointness / Subtyping Exclusivity. The following non-standard lemma is necessary to prevent overlap between the S-Match1/2 and S-Match3/4 rules.

Lemma 3.5 (Disjointness/subtyping exclusivity).
The type disjointness and subtyping relations are mutually exclusive.

If such overlap would be allowed, match types could reduce in several different ways, resulting in an unsound system. We prove Lemma 3.5 by contradiction. Our proof uses a mapping from System FM's types into non-empty subsets of a newly defined set $P = \{\Lambda, V\} \cup C$. Elements of P can be understood as equivalence classes for FM's types. We show that the subtyping relation in FM corresponds to a subset relation in P, and that the type disjointness relation in FM (disj) corresponds to set disjointness in P. This set-theoretical view lets us conclude the desired result directly. In our Coq mechanization, we axiomatize this lemma and delegate to the pen-and-paper proof.

Inversion of Subtyping. The following Lemma 3.6 allows us to perform inversion on the subtyping relation, which is important to show canonical forms (Lemma 3.7) and inversion of typing (Lemma 3.8). Stating the lemma requires the definition of a new relation denoted $\Gamma \vdash S \rightleftharpoons T$, defined in Figure 4. It represents evidence of the mutual subtyping between a match type S and a type T with the additional constraint that this evidence was exclusively constructed using pairwise applications of S-Match1/2, S-Match3/4, and S-Trans in both directions. Intuitively, $\Gamma \vdash S \rightleftharpoons T$ is handier than two independent derivations of $\Gamma \vdash S <: T$ and $\Gamma \vdash T <: S$ because it allows simultaneous induction on both subtyping directions.

Lemma 3.6 (Inversion of subtyping).

(1) *If* $\Gamma \vdash S_s$ *match* $\{U_i \Rightarrow S_i\}$ *or* $S_d \rightleftharpoons T$, *then either:*
   (a) $\Gamma \vdash S_s <: U_n$, $\forall m < n.\ \Gamma \vdash \text{disj}(S_s, U_m)$ *and* S_n *is a match type with* $\Gamma \vdash S_n \rightleftharpoons T$,
   (b) $\Gamma \vdash S_s <: U_n$, $\forall m < n.\ \Gamma \vdash \text{disj}(S_s, U_m)$ *and* $S_n = T$,
   (c) $\forall n.\ \Gamma \vdash \text{disj}(S_s, U_n)$ *and* S_d *is a match type with* $\Gamma \vdash S_d \rightleftharpoons T$,
   (d) $\forall n.\ \Gamma \vdash \text{disj}(S_s, U_n)$ *and* $S_d = T$.
(2) *If* $\Gamma \vdash S <: X$, *or* $\Gamma \vdash S <: T$ *where* T *is a match type with* $\Gamma \vdash T \rightleftharpoons X$, *then either*
   (a) S *is a match type with* $\Gamma \vdash S \rightleftharpoons Y$, *for some* Y,
   (b) S *is a type variable.*
(3) *If* $\Gamma \vdash S <: T_1 \to T_2$, *or* $\Gamma \vdash S <: T$ *where* T *is a match type with* $\Gamma \vdash T \rightleftharpoons T_1 \to T_2$, *then either*
   (a) S *is a match type with* $\Gamma \vdash S \rightleftharpoons S_1 \to S_2$, *for some* S_1, S_2 *such that* $\Gamma \vdash T_1 <: S_1$ *and* $\Gamma \vdash S_2 <: T_2$,
   (b) S *is a match type with* $\Gamma \vdash S \rightleftharpoons X$, *for some* X,
   (c) S *is a type variable,*

   (d) S *has the form* $S_1 \to S_2$ *with* $\Gamma \vdash T_1 <: S_1$ *and* $\Gamma \vdash S_2 <: T_2$.
(4) *If* $\Gamma \vdash S <: \forall X <: U_1.\ T_2$, *or* $\Gamma \vdash S <: T$ *where* T *is a match type with* $\Gamma \vdash T \rightleftharpoons \forall X <: U_1.\ T_2$, *then either*
   (a) S *is a match type with* $\Gamma \vdash S \rightleftharpoons \forall X <: U_1.\ S_2$, *for some* S_2 *such that* $\Gamma, X <: U_1 \vdash S_2 <: T_2$,
   (b) S *is a match type with* $\Gamma \vdash S \rightleftharpoons X$, *for some* X,
   (c) S *is a type variable,*
   (d) S *has the form* $\forall X <: U_1.\ S_2$ *and* $\Gamma, X <: U_1 \vdash S_2 <: T_2$.

The first point of Lemma 3.6 uses the structure of the $\rightleftharpoons$ to provide a form of inversion, which we use to prove each of the subsequent points. In comparison with the corresponding inversion lemma in $F_{<:}$'s safety proof, the statement and the proof of Lemma 3.6 are longer and more intricate. This difference is inevitable, given that match type reduction rules allow match expressions to be typed as the result of their reduction, which complexifies the inversion.

Similarly to inversion of subtyping, our canonical forms lemma is non-standard in that it uses a disjunction in its premise to account for match types.

Lemma 3.7 (Canonical forms).

(1) *If* $\Gamma \vdash t : T$, *where either* T *is a type variable, or* T *is a match type with* $\Gamma \vdash T \rightleftharpoons X$, *then* t *is not a closed value.*
(2) *If* v *is a closed value with* $\Gamma \vdash v : T$ *where either* $T = T_1 \to T_2$, *or* T *is a match type and* $\Gamma \vdash T \rightleftharpoons T_1 \to T_2$, *then* v *has the form* $\lambda x : S_1.\ t_2$.
(3) *If* v *is a closed value with* $\Gamma \vdash v : T$ *where either* $T = \forall X <: U_1.\ T_2$, *or* T *is a match type and* $\Gamma \vdash T \rightleftharpoons \forall X <: U_1.\ T_2$, *then* v *has the form* $\lambda X <: U_1.\ t_2$.

Proof of Soundness. The remaining proof of soundness is mostly standard. Lemma 3.8 and 3.9 are simple inversions of typing rules. Lemma 3.9 is needed in the proof of preservation to recover subtyping bounds from typing judgments. The proofs proceed by routine induction on derivations.

Lemma 3.8 (Inversion of typing).

(1) *If* $\Gamma \vdash \lambda x : S_1.\ s_2 : T$ *and* $\Gamma \vdash T <: U_1 \to U_2$, *then* $\Gamma \vdash U_1 <: S_1$ *and there is some* S_2 *such that* $\Gamma, x : S_1 \vdash s_2 : S_2$ *and* $\Gamma \vdash S_2 <: U_2$.
(2) *If* $\Gamma \vdash \lambda X <: S_1.\ s_2 : T$ *and* $\Gamma \vdash T <: (\forall X <: U_1.\ U_2)$, *then* $U_1 = S_1$ *and there is some* S_2 *such that* $\Gamma, X <: S_1 \vdash s_2 : S_2$ *and* $\Gamma, X <: S_1 \vdash S_2 <: U_2$.

Lemma 3.9 (Minimum types).

(1) *If* $\Gamma \vdash$ *new* $C : T$ *then* $\Gamma \vdash \{\text{new } C\} <: T$.
(2) *If* $\Gamma \vdash \lambda x : T_1.\ t_2 : T$ *then there is some* T_2 *such that* $\Gamma \vdash T_1 \to T_2 <: T$.
(3) *If* $\Gamma \vdash \lambda X <: U_1.\ t_2 : T$ *then there is some* T_2 *such that* $\Gamma \vdash \forall X <: U_1.\ T_2 <: T$.

With these lemmas in place, the proofs of progress and preservation are straightforward.

Theorem 3.10 (Progress).
If t *is a closed, well-typed term, then either* t *is a value or there is some* t' *such that* $t \longrightarrow t'$.

Theorem 3.11 (Preservation).
If $\Gamma \vdash t : T$ *and* $t \longrightarrow t'$ *then* $\Gamma \vdash t' : T$.

3.4 Type Binding Extension

In this section, we present System FMB, an extension of System FM with support for binding pattern variables in type patterns. FMB is parametrized by two sets of classes, A and B, representing non-parametric and parametric classes, respectively. A parametric class, written B T, takes exactly

Syntax

$$
\begin{array}{l|l}
\mathrm{C} ::= & \mathrm{t} ::= \ldots \\
\quad \mathrm{A} \qquad \textit{ground class} & \quad \mathrm{t}\ \textit{match}\ [\mathrm{X}]\ \{\overline{\mathrm{x} : \mathrm{C} \Rightarrow \mathrm{t}}\}\ \textit{or}\ \mathrm{t} \qquad \textit{match expr.} \\
\quad \mathrm{B}\ \mathrm{T} \qquad \textit{parametric class} & \mathrm{T} ::= \ldots \\
 & \quad \mathrm{T}\ \textit{match}\ [\mathrm{X}]\ \{\overline{\mathrm{T} \Rightarrow \mathrm{T}}\}\ \textit{or}\ \mathrm{T} \qquad \textit{match type}
\end{array}
$$

Evaluation

$$\frac{(\mathrm{C}, [\mathrm{X} \mapsto \mathrm{U}]\,\mathrm{C}_n) \in \Psi \qquad \forall m < n.\ \forall \mathrm{T}.\ (\mathrm{C}, [\mathrm{X} \mapsto \mathrm{T}]\,\mathrm{C}_m) \notin \Psi}{\textit{new}\ \mathrm{C}\ \textit{match}\ [\mathrm{X}]\ \{\mathrm{x}_i : \mathrm{C}_i \Rightarrow \mathrm{t}_i\}\ \textit{or}\ \mathrm{t}_d \longrightarrow [\mathrm{X} \mapsto \mathrm{U}]\,[\mathrm{x}_n \mapsto \textit{new}\ \mathrm{C}]\mathrm{t}_n} \quad \text{(BE-Match2)}$$

$$\frac{\forall m.\ \forall \mathrm{T}.\ (\mathrm{C}, [\mathrm{X} \mapsto \mathrm{T}]\,\mathrm{C}_m) \notin \Psi}{\textit{new}\ \mathrm{C}\ \textit{match}\ [\mathrm{X}]\ \{\mathrm{x}_i : \mathrm{C}_i \Rightarrow \mathrm{t}_i\}\ \textit{or}\ \mathrm{t}_d \longrightarrow \mathrm{t}_d} \quad \text{(BE-Match3)}$$

Subtyping

$$\frac{\Gamma, \mathrm{X} <: \mathrm{U} \vdash \mathrm{T}_s <: \mathrm{S}_n \qquad \forall m < n.\ \Gamma, \mathrm{X} <: \mathrm{Top} \vdash \mathrm{disj}(\mathrm{T}_s, \mathrm{S}_m)}{\Gamma \vdash \mathrm{T}_s\ \textit{match}\ [\mathrm{X}]\ \{\mathrm{S}_i \Rightarrow \mathrm{T}_i\}\ \textit{or}\ \mathrm{T}_d =:= [\mathrm{X} \mapsto \mathrm{U}]\,\mathrm{T}_n} \quad \text{(BS-Match1/2)}$$

$$\frac{\forall n.\ \Gamma, \mathrm{X} <: \mathrm{Top} \vdash \mathrm{disj}(\mathrm{T}_s, \mathrm{S}_n)}{\Gamma \vdash \mathrm{T}_s\ \textit{match}\ [\mathrm{X}]\ \{\mathrm{S}_i \Rightarrow \mathrm{T}_i\}\ \textit{or}\ \mathrm{T}_d =:= \mathrm{T}_d} \quad \text{(BS-Match3/4)}$$

$$\frac{\Gamma \vdash \mathrm{S}_s <: \mathrm{T}_s \qquad \Gamma \vdash \mathrm{S}_d <: \mathrm{T}_d \qquad \forall n.\ \Gamma, \mathrm{X} <: \mathrm{Top} \vdash \mathrm{S}_n <: \mathrm{T}_n}{\begin{array}{c}\Gamma \vdash \mathrm{S}_s\ \textit{match}\ [\mathrm{X}]\ \{\mathrm{U}_i \Rightarrow \mathrm{S}_i\}\ \textit{or}\ \mathrm{S}_d <: \\ \mathrm{T}_s\ \textit{match}\ [\mathrm{X}]\ \{\mathrm{U}_i \Rightarrow \mathrm{T}_i\}\ \textit{or}\ \mathrm{T}_d\end{array}} \quad \text{(BS-Match5)}$$

Typing

$$\frac{\Gamma \vdash \mathrm{t}_s : \mathrm{T}_s \qquad \Gamma, \mathrm{X} <: \mathrm{Top}, \mathrm{x}_i : \mathrm{C}_i \vdash \mathrm{t}_i : \mathrm{T}_i \qquad \Gamma \vdash \mathrm{t}_d : \mathrm{T}_d}{\Gamma \vdash \mathrm{t}_s\ \textit{match}\ [\mathrm{X}]\ \{\mathrm{x}_i : \mathrm{C}_i \Rightarrow \mathrm{t}_i\}\ \textit{or}\ \mathrm{t}_d : \mathrm{T}_s\ \textit{match}\ [\mathrm{X}]\ \{\mathrm{C}_i \Rightarrow \mathrm{T}_i\}\ \textit{or}\ \mathrm{T}_d} \quad \text{(BT-Match)}$$

Fig. 5. System FMB syntax, evaluation and typing rules for a given set of ground classes A, set of parametric classes B, class inheritance Ψ, and class disjointness Ξ. Highlights correspond to changes made to System FM.

one type parameter[1]. We redefine C to be a syntactic object defined as $\mathrm{C} ::= \mathrm{A} | \mathrm{B}\ \mathrm{T}$. The class inheritance Ψ and class disjointness Ξ remain as binary relations on C. A generic instantiation in the class hierarchy is represented as an element of Ψ, for example, `A1 extends B2[A3]` is represented as $(\mathrm{A}_1, \mathrm{B}_2\ \mathrm{A}_3) \in \Psi$. Generic inheritance is represented using multiple entries in Ψ, for example, `B1[T] extends B2[T]` is represented as $\forall \mathrm{T}.\ (\mathrm{B}_1\ \mathrm{T}, \mathrm{B}_2\ \mathrm{T}) \in \Psi$. This approach allows us to reuse most of FM's definitions. Indeed, our formal development treats C, Ψ, and Ξ as mathematical objects, and is compatible with FMB's new definition of classes.

In Figure 5, we define System FMB syntax and rules, where changes to System FM are highlighted in gray. FMB's new syntax for match expressions and match types adds a *pattern variable* to

[1]The restriction to a single type parameter is for presentation purposes. Both System FMB's type system and type-safety proof can easily be adapted to support a variable number of binding variables.

each construct. In t_s *match*$[X]\{x_i : C_i \Rightarrow t_i\}$*or* t_d, the pattern variable X is available to bind type parameters in C_i patterns.

The definitions of S-Match3/4, S-Match5, and T-Match require minor adjustments to account for the pattern variable in typing contexts. Note that pattern variables appear in contexts unconditionally, regardless of whether or not those variables are used in the corresponding patterns.

In the new subtyping rule for non-default match reduction, BS-Match1/2, the first premise instantiates the pattern variable X to some type U such that the scrutinee type is a subtype of the nth pattern:

$$\frac{\Gamma, X <: U \vdash T_s <: S_n \qquad \forall m < n.\ \Gamma, X <: \mathrm{Top} \vdash \mathrm{disj}(T_s, S_m)}{\Gamma \vdash T_s\ \mathit{match}\ [X]\ \{S_i \Rightarrow T_i\}\ \mathit{or}\ T_d =:= [X \mapsto U]\, T_n} \quad \text{(BS-Match1/2)}$$

Here U is completely unconstrained: any instantiation of X such that $T_s <: S_n$ would be admissible. The disjointness judgments use a weaker upper bound for X than the subtyping judgment (X <: Top instead of X <: U). This is because the scrutinee type must be shown disjoint with non-matching pattern types for every possible instantiation of X. In an algorithmic system, U would be computed during type inference by constraint solving.

The new evaluation rule for non-default match reduction uses a similar mechanism: it looks for the *first case* where the pattern variable can be instantiated such that the scrutinee inherits the corresponding pattern:

$$\frac{(C, [X \mapsto U]\, C_n) \in \Psi \qquad \forall m < n.\ \forall T.\ (C, [X \mapsto T]\, C_m) \notin \Psi}{\mathit{new}\ C\ \mathit{match}\ [X]\ \{x_i : C_i \Rightarrow t_i\}\ \mathit{or}\ t_d \longrightarrow [X \mapsto U]\, [x_n \mapsto \mathit{new}\ C] t_n} \quad \text{(BE-Match2)}$$

The second premise rules out non-matching cases with a universal quantifier ranging over all types. A concrete implementation would certainly opt for a more efficient approach, for instance by implementing Ψ as a lookup table.

Example C. Consider the following class hierarchy with two ground classes: Char and String, and a single parametric class List, such that String extends List Char:

$$A = \{\mathrm{Char}, \mathrm{String}\} \qquad B = \{\mathrm{List}\}$$
$$\Psi = \{(\mathrm{String}, \mathrm{List\ Char})\} \cup \mathrm{Id} \qquad \Xi = \{\}$$
$$f = \lambda x : \mathrm{Top}.\ x\ \mathit{match}[X]\{xs : \mathrm{List}\ X \Rightarrow \mathrm{foo}\}\mathit{or}\ \mathrm{bar}$$

The function f matches its arguments against the List X pattern, where X is a pattern variable. We examine the evaluation of two applications of f:

(1) f(*new* String) matches against List X with X = Char and evaluates to $[X \mapsto \mathrm{Char}][x \mapsto$ *new* String]foo via BE-Match2 (since $(\mathrm{String}, \mathrm{List\ Char}) \in \Psi$).
(2) f(*new* List Top) also matches List X, this time with X = Top, and evaluates to $[X \mapsto \mathrm{Top}][x \mapsto$ *new* List Top]foo via BE-Match2. Here $(\mathrm{List\ Top}, \mathrm{List\ Top}) \in \Psi$ follows from Ψ's reflexivity.

We established the type safety of System FMB by adapting System FM's pen-and-paper proof. The required changes are lengthy, but relatively uninteresting; it boils down to additional bookkeeping of pattern variables in contexts. The main takeaway from FMB's type safety is that the proof does not require additional constraints on type U in BS-Match1/2. As a result, algorithmic implementations are free to use any mechanism to come up with instantiations of pattern variables.

4 IMPLEMENTATION

Match types are implemented in Dotty, the reference compiler for Scala 3. This section explains how our implementation relates to the formalization presented in Section 3.

In the compiler, match-type reduction happens during subtyping, just like in System FM. In order for subtyping to remain algorithmic, match type reduction rules are never used to introduce new match types, but only to simplify the ones present in the original program. The reduction algorithm closely follows the typing rules of Section 3. The scrutinee type is compared with each pattern sequentially. If the scrutinee is a subtype of the first pattern type, the match type reduces. Otherwise, if the scrutinee can be shown to be disjoint with the first pattern type, the algorithm proceeds to the next pattern. If the algorithm reaches a pattern where neither subtyping nor disjointness can be concluded, the reduction is aborted and the match type remains unreduced.

4.1 Disjointness in Scala

Separate compilation is the biggest obstacle to concluding that two types are disjoint. Indeed, in Scala, all traits and classes are extensible by default. Because Scala programs are compiled with an open-world assumption, it is common for types to be effectively disjoint in the current compilation unit, but due to potential extensions in future compilations, the compiler must stay conservative.

Separate compilation is the reason why our formalization requires two different parameters to describe its class hierarchy. One particular instantiation of System FM can be thought of as a model of Scala's type system for a particular compilation unit, where C represents the set of classes declared *so far*. The class inheritance, Ψ, remains valid for all subsequent compilation units: new class definitions do not alter the inheritance between previously defined classes. However, the inheritance parameter (Ψ) is, on its own, not sufficient to conclude that two classes are disjoint: new class definitions can introduce new overlaps between existing classes. For this reason, our formalization uses a separate parameter to describe class disjointness (Ξ). To account for separate compilation, Ξ should only contain pairs of classes which would remain disjoint despite potential additions to the current set of classes.

Thankfully, Scala provides several ways to restrict extensibility. The *sealed* and *final* annotations on traits and classes directly restrict the extensibility of annotated types: sealed types can only be extended in the same file as its declaration, thereby providing a way to enumerate all the children of a type. Thus, disjointness of sealed traits and classes can be computed recursively by iterating over all the possible subtypes of that type. The main distinction between traits and classes is that a class can extend at most one superclass. This property allows the compiler to assert that classes are disjoint with a simple check: given two classes A and B, if neither A <: B nor B <: A, then no class could possibly extend both A and B, and those two types are disjoint.

As an example, consider the following Scala definitions (left-hand side), and the corresponding instantiation of System FM (right-hand side):

```
sealed trait Part
final class Wheel extends Part
final class DiscBrake extends Part

trait Vehicle
class Bicycle extends Vehicle
class RoadBike extends Bicycle

class Helmet
```

$$\mathrm{C} = \{\mathrm{P}, \mathrm{W}, \mathrm{D}, \mathrm{V}, \mathrm{B}, \mathrm{R}, \mathrm{H}\}$$

$$\Psi = \{(\mathrm{W}, \mathrm{P}), (\mathrm{D}, \mathrm{P}), (\mathrm{B}, \mathrm{V}), (\mathrm{R}, \mathrm{B}), (\mathrm{R}, \mathrm{V})\}$$

$$\begin{aligned}\Xi = \{&(\mathrm{P}, \mathrm{V}), (\mathrm{P}, \mathrm{B}), (\mathrm{P}, \mathrm{R}), (\mathrm{P}, \mathrm{H}),\\ &(\mathrm{W}, \mathrm{V}), (\mathrm{W}, \mathrm{B}), (\mathrm{W}, \mathrm{R}), (\mathrm{W}, \mathrm{H}),\\ &(\mathrm{D}, \mathrm{V}), (\mathrm{D}, \mathrm{B}), (\mathrm{D}, \mathrm{R}), (\mathrm{D}, \mathrm{H}),\\ &(\mathrm{W}, \mathrm{D}), (\mathrm{B}, \mathrm{H}), (\mathrm{R}, \mathrm{H})\}\end{aligned}$$

The classes and inheritance relation are practically isomorphic between the two representation: Scala classes have a one-to-one correspondence to their FM counterparts (abbreviated with initials)

and the inheritance only contains an additional entry for R and V, obtained by transitivity (Ψ's reflexivity and Ξ's symmetry are omitted for brevity).

P is declared sealed, meaning that no additional parts can be defined outside of this compilation unit. As a result, we can enumerate all parts to conclude that none are vehicles and $(\mathrm{P}, \mathrm{V}) \in \Xi$. Note that this would not be the case if either W or D was declared non-final, since extending those classes would indirectly create new parts. B and H are both classes that do not inherit each other, which implies that $(\mathrm{B}, \mathrm{H}) \in \Xi$ given that Scala classes can extend at most one class. V and H, however, cannot be concluded disjoint in those definitions. If that turns out to be a desirable property, disjointness could easily be obtained by sealing V or finalizing H.

4.2 Empty Types

An important limitation of System FM compared to Scala's type system is that it does not support empty types. The bottom of Scala's subtyping lattice, called `Nothing`, provides a direct way to refer to empty sets of values. Intersection types also provide a way to construct uninhabited types given that Scala does not forbid intersecting two disjoint types. Empty types are problematic for the match type reduction algorithm as they break the fundamental assumption that two types cannot be both disjoint and subtypes (Lemma 3.5). To account for this, our implementation uses an additional *inhabitance check* on scrutinee types before attempting any reduction.

To show why empty types are problematic, we can construct an example where, in the absence of an inhabitance check, the same match type could be reduced differently in two different contexts:

```
type M[X] = X match {
  case Int => String
  case String => Int
}
class C {
  type X
  def f(bad: M[X & String]): Int = bad
}
class D extends C {
  type X = Int
}
```

In this example, the definition of `f` in `C` type-checks because `X & String` and `Int` are disjoint (since `String` and `Int` are disjoint) and `M[X & String]` reduces to `Int` (`M`'s second case applies). Class `D` refines the definition of `C` by giving concrete definition of `X`. The unsoundness manifests itself in the body of class `D`, where `X & String` is a subtype of `Int` and `M[X & String]` reduces to `String` (`M`'s first case applies). There, it is possible to call the function `f` with a string argument, which would result in a runtime error. Checking for scrutinee inhabitance prevents this class of errors. In this example, it would prevent `M[Int & String]` from reducing given that `Int & String` is not inhabited.

4.3 Null Values

In Scala 3, null values no longer inhabit every type: nullable types require explicit annotations of the form `A | Null` [Nieto et al. 2020]. Our implementation of subtyping and disjointness handles union types and therefore needs no particular treatment of null values.

4.4 Variance

Scala supports variance annotations on type parameters of higher-kinded types. These annotations allow programmers to specify how the subtyping of annotated parameters influences the subtyping

of the higher-kinded type. For instance, `type F[+T]` defines a type `F` that is covariant in its first type parameter, meaning that `T1 <: T2` implies `F[T1] <: F[T2]`. Contravariance, written `type G[-T]`, has the opposite meaning: `T1 <: T2` implies `G[T2] <: G[T1]`.

It would appear that co- and contravariant types are always overlapping, given that, for all types `X`, `F[Nothing] <: F[X]` and `G[Any] <: G[X]` (where `Nothing` and `Any` are Scala's bottom and top types). However, in the case of covariant parameters, an exception can be made when the said type parameter corresponds to a field or a constructor parameter: the `Nothing` instantiation can be ruled out because no runtime program can produce a value of that type.

Scala tuples, for example, fall into this category. `Tuple2`, the class for pairs, is defined as follows:

```
case class Tuple2[+T1, +T2](_1: T1, _2: T2)
```

Given two instantiations of this this type, `Tuple2[Int, X]` and `Tuple2[String,X]`, although `Tuple2[Nothing,X]` is a subtype of both, there is no runtime value of type `Tuple2[Nothing,X]` (since `Nothing` is uninhabited), and, as a result, those two types are disjoint. Our disjointness algorithm implements this kind of reasoning to conclude disjointness in the presence of covariant type parameters.

4.5 Pattern Matching Exhaustivity

The Scala compiler checks for pattern matching exhaustivity to prevent runtime exceptions caused by missing cases. Exhaustivity checking uses static knowledge about the class hierarchy (such as the sealed and final annotations) to check that every value in the scrutinee type is covered by the pattern clauses [Liu 2016]. Non-exhaustive patterns are compiled with an additional "catch-all" case which throws a runtime exception. System FM uses default cases as a replacement for systematic exhaustivity checks or runtime exceptions.

4.6 Types at Runtime

Scala's primary platform is the Java virtual machine (JVM). On the JVM, Scala is compiled using *partial erasures*, where parts of types are preserved and translated to the JVM's type systems, and other parts are erased [Schinz 2005]. For instance, ground classes are compiled directly to JVM classes, but type parameters and type variables are erased and replaced by their bounds.

Erasure directly affects the type tests that can be performed at runtime. For example, while `case xs: List[Int] =>` is a syntactically valid pattern, it will lead to a compiler warning since the `Int` type parameter is eliminated by erasure and cannot be checked at runtime.

This restriction is reflected in our formalism by the difference between the evaluation rules for match expressions and the reduction rules for match types: evaluation is limited to inheritance checks on statically defined classes ($(C, C_n) \in \Psi$ in E-Match2/3), as opposed to the match type reduction rules which are defined using the subtyping and type disjointness relations ($\Gamma \vdash T_s <: S_n$ and $\Gamma \vdash \text{disj}(T_s, S_m)$ in S-Match1/2/3/4).

In System FMB (Subsection 3.4), match expressions support two sorts of parametric patterns: they can be either instantiated (*match*$\{\text{xs} : \text{List Int}\}$, where $\text{Int} \in A$), or use a binding pattern variable (*match*$[X]\{\text{xs} : \text{List X}\}$, where X is a pattern variable). In this sense, System FMB is *more expressive* than Scala, where instantiated patterns are not available at the term level due to type erasure.

4.7 Non-Termination

Unlike our calculus, the Scala implementation also allows match types to be defined recursively. Recursive match types can cause subtyping checks to loop indefinitely. Our implementation does not check match types for termination, as any such check would necessarily limit expressiveness or convenience. Instead, we detect divergence during match type reduction using a fuel mechanism. The compiler is given an initial amount of fuel, which is consumed one unit at a time on every

reduction step. If the compiler runs out of fuel, the reduction is aborted with a "recursion limit exceeded" error. The current implementation uses a fixed amount of initial fuel. Although this seems to be sufficient for most practical purposes, we plan on making it configurable. This mechanism is completely standard and already used in other programing languages with unbounded recursion at the type level [Eisenberg 2016; Eisenberg et al. 2014; Sjoberg 2015].

4.8 Inference

System FM's type rules enable any match expression to be typed as a match type, but the situation is different in the full Scala language. Pattern matching in Scala supports many sorts of patterns [Emir et al. 2007], most of which do not have a match type counterpart. Furthermore, typing match expressions as match types is not enabled by default in order to preserve backward compatibility. Instead, explicit type annotations must be provided.

4.9 Caching

Scala's type-checking algorithm makes heavy use of caching to improve its performance. Special care must be taken when caching the result of match type reduction, given that the subtyping and disjointness checks are context-dependent. Our implementation uses a context-aware cache for match types that automatically invalidates reduction results when match types are reduced in new contexts. An example where naive caching would be incorrect can be found in Subsection 6.4.

4.10 Size of the Implementation

In terms of lines of code, our match type implementation is a relatively modest addition to the Scala compiler: the overall changes amount to around 1500 lines (excluding tests and documentation).

5 CASE STUDY: SHAPE-SAFE NUMPY

In this section, we present a case study to show how match types can be used to express complex type constraints, which in turn can prevent certain programming errors at compile time. To this end, we outline the type-level implementation of a library for multidimensional arrays which mimics the NumPy API [Harris et al. 2020]. The goal of our library is to provide a *shape*-safe interface for manipulating n-dimensional arrays (abbreviated ndarrays), where array shapes and indices are checked for errors at compile-time rather than at runtime. Shape and indexing errors in ndarrays is a widely acknowledged problem [Barham and Isard 2019; Rush 2019], and several solutions have already been proposed, notably in the form of libraries that rely on type-level programming [Chen 2017; Huang et al. 2021]. Our library uses match types to provide a shape-safe NumPy-like interface.

Scala programmers have a long history of using ad-hoc solutions for type-level programming [Blanvillain et al. 2021b; Pilquist and Scodec open-source contributors 2021; Sabin and Shapeless open-source contributors 2021]. These solutions have several downsides, such as being slow to compile and cumbersome to use. Match types aim at simplifying type-level programming by providing first-class language support. We believe that the approach presented in this case study is an improvement over the status quo because it does not use metaprogramming or convoluted implicit-based encodings to express type-level operations.

5.1 Shape Errors in Python

In the example Python code below, the `img_batch` ndarray is a batch of 25 randomly generated RGB images of size 256×256. The code aims to compute a vector of length 25 containing the average grayscale color of each image in the batch, and then create a square 5×5 image of the average grayscale colors. However, this code contains a shape error and will throw an error at runtime.

```
import numpy as np
img_batch = np.random.normal(size=(25, 256, 256, 3))
avg_colors = np.mean(img_batch, (0, 1, 2))
avg_color_square = np.reshape(avg_colors, (5, 5))
```

The error is in the call to `np.mean`, which takes as argument a list of axes to reduce along. Unfortunately, the arguments to `np.mean` are off-by-one and result in a vector of length 3 (`img_batch`'s last dimension) instead of the intended length of 25 (`img_batch`'s first dimension); `avg_color` thus contains the average RGB color of the batch instead of the average grayscale color for each image. The reshape operation will then fail at runtime, as it cannot reshape a 3-element vector into a 25-element matrix. This error can be difficult to spot, given that NumPy's interface for reducing along multiple axes is index-based. Runtime errors like this one can be particularly frustrating when they occur late in a long-running computation.

In the remainder of this section, we show how match types can be used to prevent this class of error. After a preliminary introduction of singleton types (Subsection 5.2), we introduce ndarray shapes at the type level using HLists (Subsection 5.3). In Subsection 5.4, we show type-level implementations of the `np.mean` and `np.reshape` operations using match types. Finally, we show how this newly defined API can detect and report the error from our original Python example.

5.2 Singleton Types

Scala supports singleton types, which are types inhabited by a single value [Leontiev et al. 2014]. For instance, the singleton type `1` denotes the type containing the integer value `1`. The Scala standard library contains several predefined match types to perform arithmetic operations at the type level. For instance, `type +[A <: Int,B <: Int]` represents the addition of integer singleton types. Internally, the compiler is special-cased to implement these arithmetic operations using constant folding. Representing numbers with singleton types is desirable for practical purposes, but not absolutely necessary for this case study (for instance, Peano numerals could be used instead).

5.3 Type-Level Array Shape

The shape of an ndarray is a list of dimension lengths; we say that the shape $(a_1, a_2, \ldots, a_n)$ has n dimensions. For instance, a three-by-four matrix is a two-dimensional ndarray of shape $(3, 4)$. We represent the shape of an ndarray at the type level using a heterogeneous type list, or HList for short [Kiselyov et al. 2004]. We define an HList called `Shape` as an ADT with two constructors[2], `#:` and `∅`.

```
enum Shape {
  case #:[H <: Int, T <: Shape](head: H, tail: T)
  case ∅
}
```

This data type definition allows us to write lists of dimension sizes, both at the term level as `#:(3, #:(4, ∅))`, and at the type level as `#:[3, #:[4, ∅]]`. The HList type can equivalently be written with `#:` in infix notation, as `3 #: 4 #: ∅`.

To represent ndarrays, we define the `NDArray` type. This type is indexed by the ndarray element type (`T`), and by the ndarray shape (`S`), represented as an HList:

```
trait NDArray[T, S <: Shape]
```

[2]In the interest of clarity, our presentation omits type bounds that are necessary to guide type inference, for example in the definition of the `Shape` data type.

The goal of our presentation is to define type-safe operations on `NDArrays`. Since our focus is on the type-level, we do not include value-level counterparts to `T` and `S` in the definition of `NDArray`, but this would be necessary in a complete implementation.

To construct ndarrays, we define `random_normal`, which creates an ndarray of a given shape, where all elements are random `Floats`:[3]

```
def random_normal[S <: Shape](shape: S): NDArray[Float, S] = ???
```

5.4 Computation on Shapes with Match Types

Encoding the types and shapes of ndarrays in the types allows us to readily provide type- and shape-safety for simple ndarray operations. For instance, the element-wise Hadamard product, written as `np.multiply(x, y)`, requires the `x` and `y` ndarrays to have the same shape and element types. This constraint does not require any match types, but can simply be expressed as:

```
def multiply[T, S <: Shape](x: NDArray[T, S], y: NDArray[T, S]): NDArray[T, S] = ???
```

However, we will need the additional expressiveness of match types to implement more complex constraints on array shapes, such as for reshapes (5.4.1) and reductions (5.4.2).

5.4.1 Reshape. An operation commonly used in NumPy is `np.reshape`, which changes the shape of an ndarray, but does not change its values. A restriction imposed by the NumPy API is that the output shape must have the same number of elements as the input shape. The number of elements of a shape is the product of the sizes of its dimensions; an ndarray of shape `2 #: 3 #: 4 #: ∅` has $2 \times 3 \times 4 = 24$ elements. Note that an ndarray of shape `∅` is a scalar, and thus has a single element. This can be naturally expressed with a match type:

```
type NumElements[X <: Shape] <: Int =
  X match {
    case ∅ => 1
    case head #: tail => head * NumElements[tail]
  }
```

To restrict reshaping to be applicable only on valid shapes, the type system must support encoding type equality constraints. For this, we make use of Scala's implicit parameters, and of the `=:=` type equality constraint that is a part of Scala's standard library.

```
def reshape[T, From <: Shape, To <: Shape](array: NDArray[T, From], newshape: To)
    (implicit ev: NumElements[From] =:= NumElements[To]): NDArray[T, To] = ???
```

With this definition of `reshape`, the compiler will only accept a usage of `reshape` if it is able to prove that the number of elements in the old shape is the same as in the new shape. This example illustrates how match types can be used in concert with existing features like singleton types and implicit resolution to express powerful constraints.

5.4.2 Reduction. The NumPy API provides a variety of functions to reduce along a set of axes of an ndarray, such as `np.mean(ndarray, axes)` or `np.var(ndarray, axes)`. The `axes` parameter is a list of indices of dimensions, listing exactly those dimensions that will no longer be present in the output ndarray. The dimension indices can be unordered and repeated, and out-of-bounds indices result in an error. In the Python API, passing the `None` value instead of a list of axes reduces along all axes, meaning that the operation returns a scalar. Note that this is the opposite of passing `∅`, which means that we reduce over no axes (effectively a no-op).

[3] This snippet uses the triple question mark operator, Scala's standard notation for missing or omitted implementations.

This behavior is more complex than the previous examples, but can still be described accurately by a match type. We use a match type called `Reduce` to compute the return shape of the operation.

```
def mean[T, S <: Shape, A <: Shape](array: NDArray[T, S], axes: A): NDArray[T,
    Reduce[S, A]] = ???
```

We implement reductions along a given list of indices with logic similar to two nested loops. The outer loop, `Loop`, traverses the shape and counts the current index. The inner loops are implemented using `Contains` and `Remove`, standard operations on HLists (omitted). When `Loop` reaches the end of the list, if there are still axes to remove, these are out of bounds for the initial shape: the match type intentionally gets stuck in such cases.

```
type Reduce[S <: Shape, Axes <: None | Shape] <: Shape =
  Axes match {
    case None => Ø
    case Shape => Loop[S, Axes, 0]
  }

type Loop[S <: Shape, Axes <: Shape, I <: Int] <: Shape =
  S match {
    case head #: tail => Contains[Axes, I] match {
      case true => Loop[tail, Remove[Axes, I], I + 1]
      case false => head #: Loop[tail, Axes, I + 1]
    }
    case Ø => Axes match {
      case Ø => Ø
      // otherwise, do not reduce further
    }
  }
```

5.5 Shape safety

Having defined `random_normal`, `reshape` and `mean`, we can rewrite our original Python example in Scala:

```
val img_batch = random_normal(#:(25, #:(256, #:(256, #:(3, Ø)))))
val avg_colors = mean(img_batch, #:(0, #:(1, #:(2, Ø))))
val avg_color_square = reshape(avg_colors, #:(5, #:(5, Ø)))
```

As expected, the call to `mean` returns a tensor of shape `3 #: Ø`. Therefore, the call to `reshape` does not type-check since the input shape has 3 elements instead of 25. If we fix the off by one error to reduce along the correct indices, `1 #: 2 #: 3 #: Ø`, the call to `reshape` type-checks and `avg_color` has shape `25 #: Ø`, as expected.

6 RELATED WORK

In this section, we provide a review of existing work and relate match types to dependently typed calculi with subtyping, intensional type analysis, type families and roles in Haskell, and conditional types in TypeScript.

6.1 Dependently Typed Calculi with Subtyping

There is a vast amount of literature on type systems combining subtyping with dependent types, justifying a full survey to relate it appropriately. Instead, we offer a condensed summary of our reading journey and explain what led us to decide on using System $F_{<:}$ as a foundation for our formalization.

Dependently typed calculi typically use the same language to describe terms and types. This unification is also commonly used in the presence of subtyping [Hutchins 2010; Yang and Oliveira 2017; Zwanenburg 1999]. For systems with a complete term/type symmetry, this is a natural design, as it is concise and simplifies the meta-theory. Unfortunately, the lack of distinction between term and type level renders these systems impractical for our purpose, given that our research takes place in the context of an existing language with a clear term/type separation.

Singleton types provide an interesting middle ground between unified and separate syntax and have also been studied in conjunction with dependent types and subtyping [Aspinall 1995; Courant 2003; Stone and Harper 2000]. Singleton types give a mechanism to refer to terms *in* types, usually by means of a set-like syntax. This mechanism is appealing because it allows type system designers to cherry-pick the term constructs that should be allowed in types. When multiple constructs are shared between terms and types, singleton types provide a clear economy of concepts. In our study of match types, a minimal use of singleton types would result in sharing a single constructor between the term and the type languages: the constructor for matches. It is unclear if the benefits in doing so would outweigh the additional complexity.

Dependent Object Types (DOT) are, to this day, the most significant effort in formalizing Scala's type system [Amin et al. 2016]. DOT does not directly support any form of type-level computation. We considered using DOT as a starting point for our work, however, despite the recent effort to simplify DOT's soundness proof [Giarrusso et al. 2020; Rapoport et al. 2017], extending DOT remains too big of a challenge to concisely describe language extensions.

After several attempts at formalizing match types within existing systems, we decided to pursue a simpler route of adding new constructs to a system without dependent types. After all, the primary purpose of System FM is to serve as a medium to concisely *explain* our type-checking algorithm for match types. For this reason, we built our work on top of System $F_{<:}$, which we believe should be the simplest, most familiar calculus among the systems cited in this section.

6.2 Intensional Type Analysis

In their work on intensional type analysis [Harper and Morrisett 1995], Harper and Morrisett introduce the λ_i^{ML} calculus that supports structural analysis of types. In λ_i^{ML}, types are represented as expressions that can be inspected by case analysis using a "typecase" construct, available both at the term and at the type level. Match types can be seen as an extension of intensional type analysis to work with object-oriented class hierarchies and subtyping. Whereas patterns in λ_i^{ML} are limited to a fixed set of disjoint types, match types need to deal with open class hierarchies, of which not all members are known at compile-time. This means pattern types can overlap, and we need to perform an analysis of disjointness between the scrutinee type and each pattern type. Disjointness allows for sound reduction in the presence of overlapping patterns and abstract scrutinee types, while retaining the natural sequential evaluation order of pattern matching.

6.3 Type Families in Haskell

Haskell's type families allow programmers to define type-level functions using pattern matching [Chakravarty et al. 2005; Schrijvers et al. 2008]. By default, type families are open, which means that a definition can spread across multiple files and compilations. This flexibility induces a

substantial restriction on type family definitions: patterns must not overlap. One benefit of this restriction is that it prevents any ambiguity in the reduction of type families (patterns are pairwise disjoint), which is required given the distributed nature of definitions. Open type families are well-suited to be used in conjunction with type classes, since both constructs have open-ended definitions with non-overlapping constraints.

Closed type families (CTFs), as introduced by Eisenberg et al. [2014], allow for overlapping cases in type family definitions. Unlike the open variant, CTF reduction is performed sequentially, based on unification and apartness checks. In this regard, CTFs are closely related to match types. In fact, if we replace unification checks with subtyping and apartness checks with disjointness, their reduction algorithm is practically identical to ours, from a high-level perspective.

By default, Haskell checks for termination of recursive type families, but this check can be disabled to increase type families' expressiveness. Although the formalization presented in [Eisenberg et al. 2014] does not cover non-terminating families, the paper discusses a soundness problem caused by non-termination. The problem only occurs in the presence of repeated type bindings in patterns. In Scala, match types (and pattern matching in general) do not allow repeated bindings and are therefore not affected by this problem.

6.4 Roles in Haskell

Haskell's roles were introduced by Weirich et al. [2011] to fix a long-standing unsoundness caused by the interaction of open type families and the **`newtype`** construct. We understand roles as type annotations which specify whether a given type can safely be nominally compared to other types, or if *representational equality* (RE) should be used instead. The word representation in RE refers to the runtime representation of a type. In particular, RE dealiases **`newtype`** constructs.

In their introductory example, Weirich et al. show how type family reduction can lead to unsoundness in the absence of role annotations. Their presentation also includes a hypothetical translation of their example to Standard ML, which translates directly to Scala as follows:

```
trait AgeClass {
  type Age
  def addAge(a: Age, i: Int): Int
}
object AgeObject extends AgeClass {
  type Age = Int
  def addAge(a: Age, i: Int): Int = a + i
}
```

In this example, type `Age` is abstract in `AgeClass` and concrete in `AgeObject`. In the pre-role Haskell equivalent of this example, the unsoundness comes when the above definitions are combined with a type family that discriminates `Age` and `Int`. Such type family would reduce differently in `AgeClass`, where the types are different, and in `AgeObject`, where those two types are synonyms, which can easily be exploited to obtain a runtime error.

Luckily, our design of match types is not affected by this issue. The reason comes from the use of subtyping (and disjointness), which shields our implementation from incorrectly discriminating `Age` from `Int` when `Age` is abstract. Consider the following match type definition (directly translated from the Haskell example):

```
type M[X] = X match {
  case Age => Char
  case Int => Bool
```

```
}
```

Our algorithm would not reduce `M[Int]` to `Bool` in `AgeClass` as this reduction would require evidence that `Age` and `Int` are disjoint, which cannot be constructed when `Age` is an unbounded abstract type.

6.5 Conditional Types in TypeScript

TypeScript's conditional types, briefly mentioned in the introduction, are a type-level ternary operator based on subtyping. Conditional types can be nested into a sequence of patterns that evaluate in order, making them similar to match types.

The TypeScript language specification briefly describes the algorithm used to reduce conditional types in the presence of type variables [TypeScript development team 2020]. Given a type `S extends T ? Tt : Tf`, the TypeScript compiler first replaces all the type parameters in `S` and `T` by `any` (the top of TypeScript's subtyping lattice). If the resulting types (after substitution) are not subtypes, the overall condition is reduced to `Tf`. Unfortunately, this algorithm is both *unsound* and *incomplete*.

The unsoundness is caused by the incorrect widening of type parameters in contravariant position. Although TypeScript does not have syntax for variance annotations, function types are covariant in their return type and contravariant in their arguments. The conditional type unification algorithm wrongly approximates `X => string` to `any => string` and unifies the former with latter, which can lead to a runtime errors.

The incompleteness comes from the fact that type parameter approximation does not account for type parameter bounds. Consider the following example:

```
type M<X> = X extends string ? A : B
function f<X extends string>: M<X> = new A
```

Here, TypeScript's reduction algorithm fails to recognize that `new A` can be typed as `M<X>`, even though `X` is clearly a subtype of `string` in `f`'s body.

Although the situation is concerning, it might not be as bad as it seems given that soundness is a non-goal of TypeScript's type system [Bierman et al. 2014]. Nevertheless, we believe that the results of this paper are directly applicable to conditional types and could be used to improve TypeScript's type checker.

7 CONCLUSION

In this paper, we introduced *match types*, a lightweight mechanism for type-level programming that integrates seamlessly in subtyping-based programming languages. We formalized match types in System FM, a calculus based on System $F_{<:}$, and proved it sound. Furthermore, we implemented match types in the Scala 3 compiler, making them readily available to a large audience of programmers. A key insight for sound match types is the notion of disjointness, which complements subtyping in the match type reduction algorithm. In the future, we plan to investigate inference of match types to avoid code duplication in programs that operate both at the term and the type level.

REFERENCES

Martín Abadi, Luca Cardelli, Benjamin Pierce, and Gordon Plotkin. 1991. Dynamic Typing in a Statically Typed Language. *ACM Trans. Program. Lang. Syst.* 13, 2 (April 1991), 237–268. https://doi.org/10.1145/103135.103138

Nada Amin, Samuel Grütter, Martin Odersky, Tiark Rompf, and Sandro Stucki. 2016. *The Essence of Dependent Object Types.* Springer International Publishing, Cham, 249–272. https://doi.org/10.1007/978-3-319-30936-1_14

David Aspinall. 1995. Subtyping with singleton types. In *International Workshop on Computer Science Logic.* Springer, Springer Berlin Heidelberg, Berlin, Heidelberg, 1–15. https://doi.org/10.1007/BFb0022243

Brian Aydemir, Arthur Charguéraud, Benjamin C. Pierce, Randy Pollack, and Stephanie Weirich. 2008. Engineering Formal Metatheory. In *Proceedings of the 35th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (San Francisco, California, USA) *(POPL'08).* Association for Computing Machinery, New York, NY, USA, 3–15. https://doi.org/10.1145/1328438.1328443

Paul Barham and Michael Isard. 2019. Machine Learning Systems Are Stuck in a Rut. In *Proceedings of the Workshop on Hot Topics in Operating Systems* (Bertinoro, Italy) *(HotOS '19).* ACM, New York, NY, USA, 177–183. https://doi.org/10.1145/3317550.3321441

Gavin Bierman, Martín Abadi, and Mads Torgersen. 2014. Understanding TypeScript. In *Proceedings of the 28th European Conference on ECOOP 2014 — Object-Oriented Programming - Volume 8586.* Springer-Verlag, Berlin, Heidelberg, 257–281. https://doi.org/10.1007/978-3-662-44202-9_11

Olivier Blanvillain, Jonathan Brachthäuser, Maxime Kjaer, and Martin Odersky. 2021a. *Type-Level Programming with Match Types Artifact.* https://doi.org/10.5281/zenodo.5568850

Olivier Blanvillain, Jonathan Immanuel Brachthäuser, Maxime Kjaer, and Martin Odersky. 2022. Type-Level Programming with Match Types. In *Proceedings of the ACM SIGPLAN Symposium on Principles of Programming Languages (POPL'22).* ACM, New York, NY, USA. https://doi.org/10.1145/3498698

Olivier Blanvillain, Marios Iliofotou, Adelbert Chang, Gleb Kanterov, and other open-source contributors. 2016–2021b. Frameless. https://github.com/typelevel/frameless.

Luca Cardelli, Simone Martini, John C Mitchell, and Andre Scedrov. 1994. An extension of system F with subtyping. *Information and computation* 109, 1-2 (1994), 4–56. https://doi.org/10.1006/inco.1994.1013

Manuel M. T. Chakravarty, Gabriele Keller, and Simon Peyton Jones. 2005. Associated Type Synonyms. *SIGPLAN Not.* 40, 9 (Sept. 2005), 241–253. https://doi.org/10.1145/1090189.1086397

Tongfei Chen. 2017. Typesafe Abstractions for Tensor Operations (Short Paper). In *Proceedings of the 8th ACM SIGPLAN International Symposium on Scala* (Vancouver, BC, Canada) *(SCALA 2017).* Association for Computing Machinery, New York, NY, USA, 45–50. https://doi.org/10.1145/3136000.3136001

Judicaël Courant. 2003. Strong normalization with singleton types. *Electronic Notes in Theoretical Computer Science* 70, 1 (2003), 53–71. https://doi.org/10.1016/S1571-0661(04)80490-0

Richard A Eisenberg. 2016. *Dependent types in Haskell: Theory and practice.* University of Pennsylvania.

Richard A. Eisenberg, Dimitrios Vytiniotis, Simon Peyton Jones, and Stephanie Weirich. 2014. Closed Type Families with Overlapping Equations. In *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (San Diego, California, USA) *(POPL'14).* Association for Computing Machinery, New York, NY, USA, 671–683. https://doi.org/10.1145/2535838.2535856

Burak Emir, Martin Odersky, and John Williams. 2007. Matching Objects with Patterns. In *Proceedings of the 21st European Conference on Object-Oriented Programming* (Berlin, Germany) *(ECOOP'07).* Springer-Verlag, Berlin, Heidelberg, 273–298. https://doi.org/10.1007/978-3-540-73589-2_14

Paolo G. Giarrusso, Léo Stefanesco, Amin Timany, Lars Birkedal, and Robbert Krebbers. 2020. Scala Step-by-Step: Soundness for DOT with Step-Indexed Logical Relations in Iris. *Proc. ACM Program. Lang.* 4, ICFP, Article 114 (2020), 29 pages. https://doi.org/10.1145/3408996

Robert Harper and Greg Morrisett. 1995. Compiling Polymorphism Using Intensional Type Analysis. In *Proceedings of the 22nd ACM SIGPLAN-SIGACT Symposium on Principle s of Programming Languages* (San Francisco, California, USA) *(POPL '95).* Association for Computing Machinery, New York, NY, USA, 130–141. https://doi.org/10.1145/199448.199475

Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. 2020. Array programming with NumPy. *Nature* 585, 7825 (Sept. 2020), 357–362. https://doi.org/10.1038/s41586-020-2649-2

Austin Huang, Sam Stites, and Torsten Scholak. 2017–2021. HaskTorch. https://github.com/hasktorch/hasktorch.

DeLesley S. Hutchins. 2010. Pure Subtype Systems. In *Proceedings of the 37th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (Madrid, Spain) *(POPL'10).* Association for Computing Machinery, New York, NY, USA, 287–298. https://doi.org/10.1145/1706299.1706334

Oleg Kiselyov, Ralf Lämmel, and Keean Schupke. 2004. Strongly Typed Heterogeneous Collections. In *Proceedings of the 2004 ACM SIGPLAN Workshop on Haskell* (Snowbird, Utah, USA) *(Haskell '04)*. Association for Computing Machinery, New York, NY, USA, 96–107. https://doi.org/10.1145/1017472.1017488

George Leontiev, Eugene Burmako, Jason Zaugg, Adriaan Moors, Paul Phillips, Oron Port, and Miles Sabin. 2014. *SIP-23 - Literal-Based Singleton Types*. Scala Center. https://docs.scala-lang.org/sips/42.type.html

Fengyun Liu. 2016. A Generic Algorithm for Checking Exhaustivity of Pattern Matching (Short Paper). In *Proceedings of the 2016 7th ACM SIGPLAN Symposium on Scala* (Amsterdam, Netherlands) *(SCALA 2016)*. Association for Computing Machinery, New York, NY, USA, 61–64. https://doi.org/10.1145/2998392.2998401

Abel Nieto, Yaoyu Zhao, Ondřej Lhoták, Angela Chang, and Justin Pu. 2020. Scala with Explicit Nulls. In *34th European Conference on Object-Oriented Programming (ECOOP 2020) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 166)*, Robert Hirschfeld and Tobias Pape (Eds.). Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 25:1–25:26. https://doi.org/10.4230/LIPIcs.ECOOP.2020.25

Benjamin C. Pierce. 2002. *Types and programming languages*. MIT press.

Michael Pilquist and Scodec open-source contributors. 2013–2021. Scodec. https://github.com/scodec/scodec.

Marianna Rapoport, Ifaz Kabir, Paul He, and Ondvrej 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

Alexander Rush. 2019. *Tensor Considered Harmful*. Harvard NLP. https://nlp.seas.harvard.edu/NamedTensor

Miles Sabin and Shapeless open-source contributors. 2011–2021. Shapeless. https://github.com/milessabin/shapeless.

Michel Schinz. 2005. *Compiling Scala for the Java virtual machine*. Ph.D. Dissertation. EPFL, Lausanne. https://doi.org/10.5075/epfl-thesis-3302

Tom Schrijvers, Simon Peyton Jones, Manuel Chakravarty, and Martin Sulzmann. 2008. Type Checking with Open Type Functions. In *Proceedings of the 13th ACM SIGPLAN International Conference on Functional Programming* (Victoria, BC, Canada) *(ICFP'08)*. Association for Computing Machinery, New York, NY, USA, 51–62. https://doi.org/10.1145/1411204.1411215

Vilhelm Sjoberg. 2015. *A Dependently Typed Language with Nontermination*. Ph.D. Dissertation. University of Pennsylvania.

Christopher A. Stone and Robert Harper. 2000. Deciding Type Equivalence in a Language with Singleton Kinds. In *Proceedings of the 27th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (Boston, MA, USA) *(POPL'00)*. Association for Computing Machinery, New York, NY, USA, 214–227. https://doi.org/10.1145/325694.325724

TypeScript development team. 2020. *The TypeScript Handbook*. Microsoft Corporation. https://www.typescriptlang.org/

Stephanie Weirich, Dimitrios Vytiniotis, Simon Peyton Jones, and Steve Zdancewic. 2011. Generative Type Abstraction and Type-Level Computation. In *Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (Austin, Texas, USA) *(POPL'11)*. Association for Computing Machinery, New York, NY, USA, 227–240. https://doi.org/10.1145/1926385.1926411

Yanpeng Yang and Bruno C. d. S. Oliveira. 2017. Unifying Typing and Subtyping. *Proc. ACM Program. Lang.* 1, OOPSLA, Article 47 (2017), 26 pages. https://doi.org/10.1145/3133871

Jan Zwanenburg. 1999. Pure type systems with subtyping. In *International Conference on Typed Lambda Calculi and Applications*. Springer Berlin Heidelberg, Berlin, Heidelberg, 381–396. https://doi.org/10.1007/3-540-48959-2_27

APPENDIX A: A TYPE SAFETY PROOF FOR SYSTEM FM

LEMMA 3.1 (PERMUTATION). *If* Γ *and* Δ *are well-formed and* Δ *is a permutation of* Γ*, then:*

1. *If* $\Gamma \vdash \mathrm{disj}(S, T)$*, then* $\Delta \vdash \mathrm{disj}(S, T)$.
2. *If* $\Gamma \vdash S <: T$*, then* $\Delta \vdash S <: T$.
3. *If* $\Gamma \vdash t : T$*, then* $\Delta \vdash t : T$.

Proof: We prove 1. and 2. simultaneously by induction on two derivations of $\Gamma \vdash \mathrm{disj}(V, W)$ and $\Gamma \vdash S <: T$. More precisely, the induction is done on the cumulative depth of both derivation tree.

1. $\Gamma \vdash \mathrm{disj}(V, W)$
   - *Case* D-XI: $V = C_1 \quad W = C_2 \quad (C_1, C_2) \in \Xi$
     Using D-XI with context Δ directly leads to the desired result.
   - *Case* D-PSI: $V = \{\mathit{new}\, C_1\} \quad W = C_2 \quad (C_1, C_2) \notin \Psi$
     Using D-PSI with context Δ directly leads to the desired result.
   - *Case* D-SUB: $\Gamma \vdash V <: U \quad \Gamma \vdash \mathrm{disj}(U, W)$
     By the IH we get $\Delta \vdash \mathrm{disj}(U, W)$. Using the 2nd part of the lemma we obtain $\Delta \vdash V <: U$. The result follows from D-SUB.
   - *Case* D-ARROW: $V = V_1 \to V_2 \quad W = C$
     Using D-ARROW with context Δ directly leads to the desired result.
   - *Case* D-ALL: $V = \forall X <: V_1.\, V_2 \quad W = C$
     Using D-ALL with context Δ directly leads to the desired result.
2. $\Gamma \vdash S <: T$.
   - *Case* S-REFL: $T = S$
     Using S-REFL with context Δ directly leads to the desired result.
   - *Case* S-TRANS: $\Gamma \vdash S <: U \quad \Gamma \vdash U <: T$
     The result follows directly from the IH and S-TRANS.
   - *Case* S-TOP: $T = \mathrm{Top}$
     Using S-TOP with context Δ directly leads to the desired result.
   - *Case* S-SIN: $S = \{\mathit{new}\, C\} \quad T = C$
     Using S-SIN with context Δ directly leads to the desired result.
   - *Case* S-TVAR: $S = X \quad X <: T \in \Gamma$ Since Δ is a permutation of Γ, $X <: T \in \Delta$, and the result follows from S-TVAR.
   - *Case* S-ARROW: $S = S_1 \to S_2 \quad T = T_1 \to T_2$
     $\Gamma \vdash T_1 <: S_1 \quad \Gamma \vdash S_2 <: T_2$
     The result follows directly from the IH and S-ARROW.
   - *Case* S-ALL: $S = \forall X <: U_1.\, S_2 \quad T = \forall X <: U_1.\, T_2 \quad \Gamma, X <: U_1 \vdash S_2 <: T_2$
     If Δ is a permutation of Γ, then $\Delta, X <: U_1$ is a permutation of $\Gamma, X <: U_1$. Therefore, we can use the IH to get $\Delta, X <: U_1 \vdash S_2 <: T_2$. The result follows from S-ALL.
   - *Case* S-PSI: $S = C_1 \quad T = C_2 \quad (C_1, C_2) \in \Psi$
     Using S-PSI with context Δ directly leads to the desired result.
   - *Case* S-MATCH1/2: $T_1 = T_n \quad T_2 = T_s \ \mathit{match}\{S_i \Rightarrow T_i\}\mathit{or}\, T_d$
     $\Gamma \vdash T_s <: S_n \quad \forall m < n.\, \Gamma \vdash \mathrm{disj}(T_s, S_m)$
     By the 1st part of the lemma we get $\forall m < n.\, \Delta \vdash \mathrm{disj}(T_s, S_m)$. From the IH we obtain $\Delta \vdash T_s <: S_n$. The result follows from S-MATCH1/2.
   - *Case* S-MATCH3/4: $S = T_d \quad T = T_s \ \mathit{match}\{S_i \Rightarrow T_i\}\mathit{or}\, T_d \quad \forall n.\, \Gamma \vdash \mathrm{disj}(T_s, S_n)$
     The result follows from the 1st part of the lemma and S-MATCH3/4.
   - *Case* S-MATCH5: $S = S_s \ \mathit{match}\{U_i \Rightarrow S_i\}\mathit{or}\, S_d \quad T = T_s \ \mathit{match}\{U_i \Rightarrow T_i\}\mathit{or}\, T_d$
     $\Gamma \vdash S_s <: T_s \quad \forall n.\, \Gamma \vdash S_n <: T_n \quad \Gamma \vdash S_d <: T_d$
     The result follows directly from the IH and S-MATCH5.

3. By induction on a derivation of $\Gamma \vdash t : T$
   - *Case* T-Var: $t = x \quad x : T \in \Gamma$
     Since Δ is a permutation of Γ, $x : T \in \Delta$, and the result follows from T-Var.
   - *Case* T-Abs: $t = \lambda x : T_1 t_2 \quad T = T_1 \to T_2 \quad \Gamma, x : T_1 \vdash t_2 : T_2$
     If Δ is a permutation of Γ, then $\Delta, x : T_1$ is a permutation of $\Gamma, x : T_1$. Therefore, we can use the IH to get $\Delta, x : T_1 \vdash t_2 : T_2$. The result follows from T-Abs.
   - *Case* T-App: $t = t_1 t_2 \quad T = T_{12} \quad \Gamma \vdash t_1 : T_{11} \to T_{12} \quad \Gamma \vdash t_2 : T_{11}$
     The result follows directly from the IH and T-App.
   - *Case* T-TAbs: $t = \lambda X <: U_1 .\ t_2 \quad T = \forall X <: U_1 .\ T_2 \quad \Gamma, X <: U_1 \vdash t_2 : T_2$
     If Δ is a permutation of Γ, then $\Delta, X <: U_1$ is a permutation of $\Gamma, X <: U_1$. Therefore, we can use the IH to get $\Delta, X <: U_1 \vdash t_2 : T_2$. The result follows from T-TAbs.
   - *Case* T-TApp: $t = t_1 T_2 \quad T = [X \mapsto T_2]T_{12}$
     $\Gamma \vdash t_1 : (\forall X <: U_1 .\ T_{12}) \quad \Gamma \vdash T_2 <: U_1$
     By the IH we get $\Delta \vdash t_1 : (\forall X <: U_1 .\ T_{12})$. Using the 2nd part of the lemma we obtain $\Delta \vdash T_2 <: U_1$. The result follows from T-TApp.
   - *Case* T-Sub: $\Gamma \vdash t : S \quad \Gamma \vdash S <: T$
     By the IH we get $\Delta \vdash t : S$. Using the 2nd part of the lemma we obtain $\Delta \vdash S <: T$. The result follows from T-Sub.
   - *Case* T-Class: $t = new\, C \quad T = \{new\, C\}$
     Using T-Class with context Δ directly leads to the desired result.
   - *Case* T-Match: $t = t_s\ match\{x_i : C_i \Rightarrow t_i\}or\, t_d \quad T = T_s\ match\{C_i \Rightarrow T_i\}or\, T_d$
     $\Gamma \vdash t_s : T_s \quad \Gamma, x_i : C_i \vdash t_i : T_i \quad \Gamma \vdash t_d : T_d$
     If Δ is a permutation of Γ, then $\Delta, x_i : C_i$ is a permutation of $\Gamma, x_i : C_i$. Therefore the result follows directly from the IH and T-Match.

□

Lemma 3.2 (Weakening).

1. *If* $\Gamma \vdash \mathrm{disj}(S, T)$ *and* $\Gamma, X <: U$ *is well formed,*
   then $\Gamma, X <: U \vdash \mathrm{disj}(S, T)$.
2. *If* $\Gamma \vdash S <: T$ *and* $\Gamma, X <: U$ *is well formed,*
   then $\Gamma, X <: U \vdash S <: T$.
3. *If* $\Gamma \vdash S <: T$ *and* $\Gamma, x : U$ *is well formed,*
   then $\Gamma, x : U \vdash S <: T$.
4. *If* $\Gamma \vdash t : T$ *and* $\Gamma, x : U$ *is well formed,*
   then $\Gamma, x : U \vdash t : T$.
5. *If* $\Gamma \vdash t : T$ *and* $\Gamma, X <: U$ *is well formed,*
   then $\Gamma, X <: U \vdash t : T$.

Proof: We prove 1. and 2. simultaneously by induction on two derivations of $\Gamma \vdash \mathrm{disj}(V, W)$ and $\Gamma \vdash S <: T$. More precisely, the induction is done on the cumulative depth of both derivation tree.

1. $\Gamma \vdash \mathrm{disj}(V, W)$
   - *Case* D-Xi: $V = C_1 \quad W = C_2 \quad (C_1, C_2) \in \Xi$
     Using D-Xi with context $\Gamma, X <: U$ directly leads to the desired result.
   - *Case* D-Psi: $V = \{new\, C_1\} \quad W = C_2 \quad (C_1, C_2) \notin \Psi$
     Using D-Psi with context $\Gamma, X <: U$ directly leads to the desired result.
   - *Case* D-Sub: $\Gamma \vdash V <: U \quad \Gamma \vdash \mathrm{disj}(U, W)$
     By the IH we get $\Gamma \vdash \mathrm{disj}(U, W)$. Using the 2nd part of the lemma we obtain $\Gamma, X <: U \vdash V <: U$. The result follows from D-Sub.

- *Case* D-Arrow: $V = V_1 \to V_2 \quad W = C$
  Using D-Arrow with context $\Gamma, X <: U$ directly leads to the desired result.
- *Case* D-All: $V = \forall X <: V_1.\ V_2 \quad W = C$
  Using D-All with context $\Gamma, X <: U$ directly leads to the desired result.

2. $\Gamma \vdash S <: T$.
   - *Case* S-Refl: $T = S$
     Using S-Refl with context $\Gamma, X <: U$ directly leads to the desired result.
   - *Case* S-Trans: $\Gamma \vdash S <: U \quad \Gamma \vdash U <: T$
     The result follows from the IH and S-Trans.
   - *Case* S-Top: $T = Top$
     Using S-Top with context $\Gamma, X <: U$ directly leads to the desired result.
   - *Case* S-Sin: $S = \{new\ C\} \quad T = C$
     Using S-Sin with context $\Gamma, X <: U$ directly leads to the desired result.
   - *Case* S-TVar: $S = Y \quad Y <: T \in \Gamma$
     If $Y <: T \in \Gamma$, then $Y <: T \in \Gamma, X <: U$ and the result follows from S-TVar.
   - *Case* S-Arrow: $S = S_1 \to S_2 \quad T = T_1 \to T_2$
     $\Gamma \vdash T_1 <: S_1 \quad \Gamma \vdash S_2 <: T_2$
     The result follows from the IH and S-Arrow.
   - *Case* S-All: $S = \forall Y <: U_1.\ S_2 \quad T = \forall Y <: U_1.\ T_2 \quad \Gamma, Y <: U_1 \vdash S_2 <: T_2$
     Using the IH with context $\Gamma_{IH} = \Gamma, Y <: U_1$, we get $\Gamma, Y <: U_1, X <: U \vdash S_2 <: T_2$. From Lemma 3.1, $\Gamma, X <: U, Y <: U_1 \vdash S_2 <: T_2$ The result follows from S-All.
   - *Case* S-Psi: $S = C_1 \quad T = C_2 \quad (C_1, C_2) \in \Psi$
     Using S-Psi with context $\Gamma, X <: U$ directly leads to the desired result.
   - *Case* S-Match1/2: $T_1 = T_n \quad T_2 = T_s\ match\{S_i \Rightarrow T_i\}or\ T_d$
     $\Gamma \vdash T_s <: S_n \quad \forall m < n.\ \Gamma \vdash disj(T_s, S_m)$
     From the IH we get $\Gamma, X <: U \vdash T_s <: S_n$. Using the 1st part of the lemma we obtain $\forall m < n.\ \Gamma, X <: U \vdash disj(T_s, S_m)$. The result follows from S-Match1/2.
   - *Case* S-Match3/4: $T_1 = T_d \quad T_2 = T_s\ match\{S_i \Rightarrow T_i\}or\ T_d \quad \forall n.\ \Gamma \vdash disj(T_s, S_n)$
     Using the 1st part of the lemma we get $\forall n.\ \Gamma, X <: U \vdash disj(T_s, S_n)$. The result follows from S-Match3/4.
   - *Case* S-Match5: $S = S_s\ match\{S_i \Rightarrow T_i\}or\ T_d \quad T = T_s\ match\{S_i \Rightarrow U_i\}or\ U_d$
     $\Gamma \vdash S_s <: T_s \quad \forall n.\ \Gamma \vdash T_n <: U_n \quad \Gamma \vdash T_d <: U_d$
     We use the IH on each premise and the result follows directly from S-Match5.

3. By inspection of the subtyping rules, it is clear that typing assumbtions play no role in subtyping derivations.

4. By induction on a derivation of $\Gamma \vdash t : T$.
   - *Case* T-Var: $t = y \quad y : T \in \Gamma$
     If $y : T \in \Gamma$ then $y : T \in \Gamma, x : U$ and the result follows from T-Var.
   - *Case* T-Abs: $t = \lambda y : T_1 t_2 \quad T = T_1 \to T_2 \quad \Gamma, y : T_1 \vdash t_2 : T_2$
     Using the IH with $\Gamma_{IH} = \Gamma, y : T_1$ we get $\Gamma, y : T_1, x : U \vdash t_2 : T_2$. From Lemma 3.1, $\Gamma, x : U, y : T_1 \vdash t_2 : T_2$. The result follows from T-Abs.
   - *Case* T-App: $t = t_1 t_2 \quad T = T_{12} \quad \Gamma \vdash t_1 : T_{11} \to T_{12} \quad \Gamma \vdash t_2 : T_{12}$
     We use the IH on each premise and the result follows from T-App.
   - *Case* T-TAbs: $t = \lambda X <: T_1.\ t_2 \quad T = \forall X <: T_1.\ T_2 \quad \Gamma, X <: T_1 \vdash t_2 : T_2$
     Using the IH with $\Gamma_{IH} = \Gamma, X <: T_1$ we get $\Gamma, X <: T_1, x : U \vdash t_2 : T_2$. From Lemma 3.1, $\Gamma, x : U, X <: T_1 \vdash t_2 : T_2$ The result follows from T-TAbs.
   - *Case* T-TApp: $t = t_1 T_2 \quad T = [X \mapsto T_2]T_{12}$
     $\Gamma \vdash t_1 : (\forall X <: T_{11}.\ T_{12}) \quad \Gamma \vdash T_2 <: T_{11}$

Using the IH on the left premise we get $\Gamma, x : U \vdash t_1 : (\forall X <: T_{11}.\ T_{12})$. Using the 3rd part of the lemma on the right premise we obtain $\Gamma, x : U \vdash T_2 <: T_{11}$. The result follows from T-TApp.

- *Case* T-Sub: $\Gamma \vdash t : S \quad \Gamma \vdash S <: T$
  Using the IH on the left premise we get $\Gamma, x : U \vdash t : S$. Using the 3rd part of the lemma on the right premise we obtain $\Gamma, x : U \vdash S <: T$. The result follows from T-Sub.
- *Case* T-Class: $t = \mathit{new}\, C \quad T = \{\mathit{new}\, C\}$
  Using T-Class with context $\Gamma, x : U$ directly leads to the desired result.
- *Case* T-Match: $t = t_s\ \mathit{match}\{x_i : C_i \Rightarrow t_i\}\mathit{or}\, t_d \quad T = T_s\ \mathit{match}\{C_i \Rightarrow T_i\}\mathit{or}\, T_d$
  $\Gamma \vdash t_s : T_s \quad \Gamma, x_i : C_i \vdash t_i : T_i \quad \Gamma \vdash t_d : T_d$
  We use the IH on each premise and the result follows directly from Lemma 3.1 and T-Match.

5. By induction on a derivation of $\Gamma \vdash t : T$.
   - *Case* T-Var: $t = x \quad x : T \in \Gamma$
     If $y : T \in \Gamma$ then $y : T \in \Gamma, X <: U$ and the result follows from T-Var.
   - *Case* T-Abs: $t = \lambda x : T_1 t_2 \quad T = T_1 \to T_2 \quad \Gamma, x : T_1 \vdash t_2 : T_2$
     Using the IH with $\Gamma_{IH} = \Gamma, x : T_1$ we get $\Gamma, x : T_1, X <: U \vdash t_2 : T_2$. From Lemma 3.1, $\Gamma, X <: U, x : T_1 \vdash t_2 : T_2$. The result follows from T-Abs.
   - *Case* T-App: $t = t_1 t_2 \quad T = T_{12} \quad \Gamma \vdash t_1 : T_{11} \to T_{12} \quad \Gamma \vdash t_2 : T_{12}$
     We use the IH on each premise and the result follows from T-App.
   - *Case* T-TAbs: $t = \lambda Y <: T_1.\ t_2 \quad T = \forall Y <: T_1.\ T_2 \quad \Gamma, Y <: T_1 \vdash t_2 : T_2$
     Using the IH with $\Gamma_{IH} = \Gamma, Y <: T_1$ we get $\Gamma, Y <: T_1, X <: U \vdash t_2 : T_2$. From Lemma 3.1, $\Gamma, X <: U, Y <: T_1, \vdash t_2 : T_2$. The result follows from T-TAbs.
   - *Case* T-TApp: $t = t_1 T_2 \quad T = [Y \mapsto T_2]T_{12}$
     $\Gamma \vdash t_1 : (\forall Y <: T_{11}.\ T_{12}) \quad \Gamma \vdash T_2 <: T_{11}$
     Using the IH on the left premise we get $\Gamma, X <: U \vdash t_1 : (\forall X <: T_{11}.\ T_{12})$. Using the 2nd part of the lemma on the right premise we obtain $\Gamma, X <: U \vdash T_2 <: T_{11}$. The result follows from T-TApp.
   - *Case* T-Sub: $\Gamma \vdash t : S \quad \Gamma \vdash S <: T$
     Using the IH on the left premise we get $\Gamma, S <: U \vdash t : S$. Using the 2nd part of the lemma on the right premise we obtain $\Gamma, X <: U \vdash S <: T$. The result follows from T-Sub.
   - *Case* T-Class: $t = \mathit{new}\, C \quad T = \{\mathit{new}\, C\}$
     Using T-Class with context $\Gamma, X <: U$ directly leads to the desired result.
   - *Case* T-Match: $t = t_s\ \mathit{match}\{x_i : C_i \Rightarrow t_i\}\mathit{or}\, t_d \quad T = T_s\ \mathit{match}\{C_i \Rightarrow T_i\}\mathit{or}\, T_d$
     $\Gamma \vdash t_s : T_s \quad \Gamma, x_i : C_i \vdash t_i : T_i \quad \Gamma \vdash t_d : T_d$
     We use the IH on each premise and the result follows directly from Lemma 3.1 and T-Match.

□

Lemma 3.3 (Strengthening). *If* $\Gamma, x : T, \Delta \vdash S <: T$, *then* $\Gamma, \Delta \vdash S <: T$.

Proof: By inspection of the subtyping rules, it is clear that typing assumbtions play no role in subtyping derivations.

□

Lemma 3.4 (Substitution).

1. *If* $\Gamma, X <: Q, \Delta \vdash \mathrm{disj}(S, T)$ *and* $\Gamma \vdash P <: Q$,
   then $\Gamma, [X \mapsto P]\Delta \vdash \mathrm{disj}([X \mapsto P]S, [X \mapsto P]T)$.

2. *If* $\Gamma, X <: Q, \Delta \vdash S <: T$ *and* $\Gamma \vdash P <: Q$,
   then $\Gamma, [X \mapsto P]\Delta \vdash [X \mapsto P]S <: [X \mapsto P]T$.
3. *If* $\Gamma, X <: Q, \Delta \vdash t : T$ *and* $\Gamma \vdash P <: Q$,
   then $\Gamma, [X \mapsto P]\Delta \vdash [X \mapsto P]t : [X \mapsto P]T$.
4. *If* $\Gamma, x : Q, \Delta \vdash t : T$ *and* $\Gamma \vdash q : Q$,
   then $\Gamma, \Delta \vdash [x \mapsto q]t : T$.

Proof: We prove 1. and 2. simultaneously by induction on two derivations of $\Gamma, X <: Q, \Delta \vdash \mathrm{disj}(V, W)$ and $\Gamma, X <: Q, \Delta \vdash S <: T$. More precisely, the induction is done on the cumulative depth of both derivation tree.

1. $\Gamma, X <: Q, \Delta \vdash \mathrm{disj}(V, W)$
   - *Case* D-Xi: $V = C_1 \quad W = C_2 \quad (C_1, C_2) \in \Xi$
     Since $[X \mapsto P]C_1 = C_1$ and $[X \mapsto P]C_2 = C_2$, we can use D-Xi with context $\Gamma, [X \mapsto P]\Delta$ to obtain the desired result.
   - *Case* D-Psi: $V = \{new\, C_1\} \quad W = C_2 \quad (C_1, C_2) \notin \Psi$
     Since $[X \mapsto P]\{new\, C_1\} = \{new\, C_1\}$ and $[X \mapsto P]C_2 = C_2$, we can use D-Psi with context $\Gamma, [X \mapsto P]\Delta$ to obtain the desired result.
   - *Case* D-Sub: $\Gamma, X <: Q, \Delta \vdash V <: U \quad \Gamma, X <: Q, \Delta \vdash \mathrm{disj}(U, W)$
     By the IH we get $\Gamma, [X \mapsto P]\Delta \vdash \mathrm{disj}([X \mapsto P]U, [X \mapsto P]W)$. Using the 2nd part of the lemma we obtain $\Gamma, [X \mapsto P]\Delta \vdash [X \mapsto P]V <: [X \mapsto P]U$. The result follows from D-Sub.
   - *Case* D-Arrow: $V = V_1 \to V_2 \quad W = C$
     Since $[X \mapsto P](V_1 \to V_2) = [X \mapsto P]V_1 \to [X \mapsto P]V_2$ and $[X \mapsto P]C = C$, we can use D-Arrow with context $\Gamma, [X \mapsto P]\Delta$ to obtain the desired result.
   - *Case* D-All: $V = \forall Y <: V_1.\ V_2 \quad W = C$
     Since $[X \mapsto P](\forall Y <: V_1.\ V_2) = \forall Y <: [X \mapsto P]V_1.\ [X \mapsto P]V_2$ and $[X \mapsto P]C = C$, we can use D-All with context $\Gamma, [X \mapsto P]\Delta$ to obtain the desired result.
2. $\Gamma, X <: Q, \Delta \vdash S <: T$.
   - *Case* S-Refl: $T = S$
     The result follows directly from S-Refl.
   - *Case* S-Trans: $\Gamma, X <: Q, \Delta \vdash S <: U \quad \Gamma, X <: Q, \Delta \vdash U <: T$
     The result follows directly from the IH and S-Trans.
   - *Case* S-Top: $T = \mathrm{Top}$
     $[X \mapsto P]\mathrm{Top} = \mathrm{Top}$ and the result follows from S-Top.
   - *Case* S-Sin: $S = \{new\, C\} \quad T = C$
     $[X \mapsto P]\{new\, C\} = \{new\, C\}$, $[X \mapsto P]C = C$, and the result follows from S-Sin.
   - *Case* S-TVar: $S = Y \quad Y <: T \in (\Gamma, X <: Q, \Delta)$
     By context well-formedness, $Y <: T \in (\Gamma, X <: Q, \Delta)$ can be decomposed into 3 subcases:
     Subcase $Y <: T \in \Gamma$:
     By context well-formedness X does not appear in Γ consequently is also absent from Y and T. Hence $Y = [X \mapsto P]Y$, $T = [X \mapsto P]T$ and $[X \mapsto P]Y <: [X \mapsto P]T \in (\Gamma, [X \mapsto P]\Delta)$. The result follows from S-TVar.
     Subcase $Y <: T \in \Delta$:
     $[X \mapsto P]Y <: [X \mapsto P]T \in [X \mapsto P]\Delta$ and $[X \mapsto P]Y <: [X \mapsto P]T \in (\Gamma, [X \mapsto P]\Delta)$. The result follows from S-TVar.
     Subcase $Y <: T = X <: Q$ (*i.e.* $Y = X$ and $T = Q$):
     By context well-formedness, X doesn't appear in Q, and $[X \mapsto P]T = [X \mapsto P]Q = Q$. Also $[X \mapsto P]S = [X \mapsto P]X = P$ and $[X \mapsto P]T = Q$. As a result, $\Gamma \vdash P <: Q$ implies

$\Gamma \vdash [X \mapsto P]S <: [X \mapsto P]T$. Using Lemma 3.2 we get $\Gamma, [X \mapsto P]\Delta \vdash [X \mapsto P]S <: [X \mapsto P]T$, as required.

- *Case* S-Arrow: $S = S_1 \to S_2 \quad T = T_1 \to T_2$
  $\Gamma, X <: Q, \Delta \vdash T_1 <: S_1 \quad \Gamma, X <: Q, \Delta \vdash S_2 <: T_2$
  $[X \mapsto P](S_1 \to S_2) = [X \mapsto P]S_1 \to [X \mapsto P]S_2$ and $[X \mapsto P](T_1 \to T_2) = [X \mapsto P]T_1 \to [X \mapsto P]T_2$. The result follows from the IH and S-Arrow.
- *Case* S-All: $S = \forall Y <: U_1.\, S_2 \quad T = \forall Y <: U_1.\, T_2 \quad \Gamma, X <: Q, \Delta, Y <: U_1 \vdash S_2 <: T_2$
  We instantiate the IH with $\Delta_{IH} = (\Delta, Y <: U_1)$ to obtain $\Gamma, [X \mapsto P]\Delta, Y <: [X \mapsto P]U_1 \vdash [X \mapsto P]S_2 <: [X \mapsto P]T_2$. Using S-All, we get $\Gamma, [X \mapsto P]\Delta \vdash (\forall Y <: [X \mapsto P]U_1.\, [X \mapsto P]S_2) <: (\forall Y <: [X \mapsto P]U_1.\, [X \mapsto P]T_2)$, that is, $\Gamma, [X \mapsto P]\Delta \vdash [X \mapsto P](\forall Y <: U_1.\, S_2) <: [X \mapsto P](\forall Y <: U_1.\, T_2)$, as required.
- *Case* S-Psi: $S = C_1 \quad T = C_2 \quad (C_1, C_2) \in \Psi$
  $[X \mapsto P]C_1 = C_1$ and $[X \mapsto P]C_2 = C_2$. The result follows from S-Psi.
- *Case* S-Match1/2: $T_1 = T_n \quad T_2 = T_s\ \mathit{match}\{S_i \Rightarrow T_i\}\mathit{or}\, T_d$
  $\Gamma, X <: Q, \Delta \vdash T_s <: S_n \quad \forall m < n.\, \Gamma, X <: Q, \Delta \vdash \mathrm{disj}(T_s, S_m)$
  Using the 1st part of the lemma we get $\forall m < n.\, \Gamma, [X \mapsto P]\Delta \vdash \mathrm{disj}([X \mapsto P]T_s, [X \mapsto P]S_m)$. By the IH we get $\Gamma, [X \mapsto P]\Delta \vdash [X \mapsto P]T_s <: S_n$. $[X \mapsto P]T = [X \mapsto P]T_s\ \mathit{match}\{[X \mapsto P]S_i \Rightarrow [X \mapsto P]T_i\}\mathit{or}\, [X \mapsto P]T_d$, and the result follows from S-Match1/2.
- *Case* S-Match3/4: $S = T_d \quad T = T_s\ \mathit{match}\{S_i \Rightarrow T_i\}\mathit{or}\, T_d$
  $\forall n.\, \Gamma, X <: Q, \Delta \vdash \mathrm{disj}(T_s, S_n)$
  By the 1st part of the lemma we get $\forall n.\, \Gamma, [X \mapsto P]\Delta \vdash \mathrm{disj}([X \mapsto P]T_s, [X \mapsto P]S_n)$ and the result follows from S-Match3/4.
- *Case* S-Match5: $S = S_s\ \mathit{match}\{U_i \Rightarrow S_i\}\mathit{or}\, S_d \quad T = T_s\ \mathit{match}\{U_i \Rightarrow T_i\}\mathit{or}\, T_d$
  $\Gamma, X <: Q, \Delta \vdash S_s <: T_s \quad \forall n.\, \Gamma, X <: Q, \Delta \vdash S_n <: T_n$
  $\Gamma, X <: Q, \Delta \vdash S_d <: T_d$
  The result follows directly from the IH.

3. By induction on a derivation of $\Gamma, X <: Q, \Delta \vdash t : T$.
   - *Case* T-Var: $t = x \quad x : T \in \Gamma, X <: Q, \Delta$
     $[X \mapsto P]x = x$. T-Var's premise can be divided in two subcases:
     Subcase $x : T \in \Gamma$:
     Context well-formedness impiles that there is no occurence of X in T and $[X \mapsto P]T = T$. Also, $x : T \in \Gamma, [X \mapsto P]\Delta$ and the result follows from T-Var.
     Subcase $x : T \in \Delta$:
     Context well-formedness impiles that there is a unique occurence of $x : T$ in Δ, that is, there exists Δ_1, Δ_2 such that $\Delta = \Delta_1, x : T, \Delta_2$, $x \notin \Delta_1$ and $x \notin \Delta_2$. As a result, $[X \mapsto P]\Delta = [X \mapsto P]\Delta_1, x : [X \mapsto P]T, [X \mapsto P]\Delta_2$ and $x : [X \mapsto P]T \in \Gamma, [X \mapsto P]\Delta$. The result follows from T-Var.
   - *Case* T-Abs: $t = \lambda x : T_1 t_2 \quad T = T_1 \to T_2 \quad \Gamma, X <: Q, \Delta, x : T_1 \vdash t_2 : T_2$
     $[X \mapsto P](\lambda x : T_1 t_2) = \lambda x : [X \mapsto P]T_1 [X \mapsto P]t_2$ and $[X \mapsto P](T_1 \to T_2) = [X \mapsto P]T_1 \to [X \mapsto P]T_2$. We instantiate the IH with $\Delta_{IH} = (\Delta, x : T_1)$ to obtain $\Gamma, [X \mapsto P]\Delta, x : [X \mapsto P]T_1 \vdash [X \mapsto P]t_2 : [X \mapsto P]T_2$. The result follows from T-Abs.
   - *Case* T-App: $t = t_1 t_2 \quad T = T_{12} \quad \Gamma, X <: Q, \Delta \vdash t_1 : T_{11} \to T_{12} \quad \Gamma, X <: Q, \Delta \vdash t_2 : T_{11}$
     $[X \mapsto P](t_1 t_2) = [X \mapsto P]t_1 [X \mapsto P]t_2$ and $[X \mapsto P]T_{11} \to T_{12} = [X \mapsto P]T_{11} \to [X \mapsto P]T_{12}$. The result follows directly from the IH and T-App.
   - *Case* T-TAbs: $t = \lambda Y <: T_1.\, t_2 \quad T = \forall Y <: T_1.\, T_2 \quad \Gamma, X <: Q, \Delta, Y <: T_1 \vdash t_2 : T_2$

$[X \mapsto P](\lambda Y <: T_1.\ t_2) = \lambda Y <: [X \mapsto P]T_1.\ [X \mapsto P]t_2$ and $[X \mapsto P](\forall Y <: T_1.\ T_2) = \forall Y <: [X \mapsto P]T_1.\ [X \mapsto P]T_2$. We instantiate the IH with $\Delta_{IH} = (\Delta, Y <: T_1)$ to obtain $\Gamma, [X \mapsto P]\Delta, Y <: [X \mapsto P]T_1 \vdash [X \mapsto P]t_2 : [X \mapsto P]T_2$. The result follows from T-TAbs.

- *Case* T-TApp: $t = t_1 T_2 \quad T = [Y \mapsto T_2]T_{12} \quad \Gamma, X <: Q, \Delta \vdash t_1 : (\forall Y <: T_{11}.\ T_{12}) \quad \Gamma, X <: Q, \Delta \vdash T_2 <: T_{11}$
  By the IH, $\Gamma, [X \mapsto P]\Delta \vdash [X \mapsto P]t_1 : [X \mapsto P](\forall Y <: T_{11}.\ T_{12})$, i.e., $\Gamma, [X \mapsto P]\Delta \vdash [X \mapsto P]t_1 : \forall Y <: [X \mapsto P]T_{11}.\ [X \mapsto P]T_{12}$. Using the second part of the lemma we get, $\Gamma, [X \mapsto P]\Delta \vdash [X \mapsto P]T_2 <: [X \mapsto P]T_{11}$. By T-TApp we get, $\Gamma, [X \mapsto P]\Delta \vdash [X \mapsto P]t_1\,[X \mapsto P]T_2 : [Y \mapsto [X \mapsto P]T_2][X \mapsto P]T_{12}$, i.e., $\Gamma, [X \mapsto P]\Delta \vdash [X \mapsto P](t_1 T_2) : [X \mapsto P]([Y \mapsto T_2]T_{12})$, as required.
- *Case* T-Sub: $\Gamma, X <: Q, \Delta \vdash t : S \quad \Gamma, X <: Q, \Delta \vdash S <: T$
  By the IH, $\Gamma, [X \mapsto P]\Delta \vdash [X \mapsto P]t : [X \mapsto P]S$. Using the second part of the lemma we get, $\Gamma, [X \mapsto P]\Delta \vdash [X \mapsto P]S <: [X \mapsto P]T$. The result follows from T-Sub.
- *Case* T-Class: $t = new\ C \quad T = \{new\ C\}$
  $[X \mapsto P]new\ C = new\ C$ and using T-Class with context $\Gamma, [X \mapsto P]\Delta$ directly leads to the desired result.
- *Case* T-Match: $t = t_s\ match\{x_i : C_i \Rightarrow t_i\}or\ t_d \quad T = T_s\ match\{C_i \Rightarrow T_i\}or\ T_d \quad \Gamma, X <: Q, \Delta \vdash t_s : T_s \quad \Gamma, X <: Q, \Delta, x_i : C_i \vdash t_i : T_i \quad \Gamma, X <: Q, \Delta \vdash t_d : T_d$
  $[X \mapsto P](t_s\ match\{x_i : C_i \Rightarrow t_i\}or\ t_d) = [X \mapsto P]t_s\ match\{x_i : C_i \Rightarrow [X \mapsto P]t_i\}or\ [X \mapsto P]t_d$.
  $[X \mapsto P](T_s\ match\{C_i \Rightarrow T_i\}or\ T_d) = [X \mapsto P]T_s\ match\{C_i \Rightarrow [X \mapsto P]T_i\}or\ [X \mapsto P]T_d$.
  The result follows directly from the IH and T-App.

4. By induction on a derivation of $\Gamma, x : Q, \Delta \vdash t : T$.
   - *Case* T-Var: $t = y \quad y : T \in \Gamma, x : Q, \Delta$
     By context well-formedness, the premise can be decomposed into 3 subcases:
     Subcase $y : T \in \Gamma$:
     $[x \mapsto q]y = y$ and $y : T \in \Gamma, \Delta$. The result follows from T-Var.
     Subcase $y : T \in \Delta$:
     Ditto.
     Subcase $y : T = x : Q$ (i.e. $y = x$ and $T = Q$):
     Using $\Gamma \vdash q : Q$ and Lemma 3.2 we get $\Gamma, \Delta \vdash q : Q$. Since $[x \mapsto q]y = q$ and $T = Q$, $\Gamma, \Delta \vdash [x \mapsto q]y : T$, as required.
   - *Case* T-Abs: $t = \lambda y : T_1 t_2 \quad T = T_1 \to T_2 \quad \Gamma, x : Q, \Delta, y : T_1 \vdash t_2 : T_2$
     We instantiate the IH with $\Delta_{IH} = (\Delta, y : T_1)$ to obtain $\Gamma, \Delta, y : T_1 \vdash [x \mapsto q]t_2 : T_2$. $[x \mapsto q](\lambda y : T_1 t_2) = \lambda y : T_1 [x \mapsto q]t_2$ and the result follows from T-Abs.
   - *Case* T-App: $t = t_1 t_2 \quad T = T_{12} \quad \Gamma, x : Q, \Delta \vdash t_1 : T_{11} \to T_{12} \quad \Gamma, x : Q, \Delta \vdash t_2 : T_{11}$
     $[x \mapsto q](t_1 t_2) = ([x \mapsto q]t_1)([x \mapsto q]t_2)$ and the result follows from the IH and T-App.
   - *Case* T-TAbs: $t = \lambda X <: U_1.\ t_2 \quad T = \forall X <: U_1.\ T_2 \quad \Gamma, x : Q, \Delta, X <: U_1 \vdash t_2 : T_2$
     We instantiate the IH with $\Delta_{IH} = (\Delta, X <: U_1)$ to obtain $\Gamma, \Delta, X <: U_1 \vdash [x \mapsto q]t_2 : T_2$. $[x \mapsto q](\lambda X <: U_1.\ t_2) = \lambda X <: U_1.\ [x \mapsto q]t_2$ and the result follows from T-TAbs.
   - *Case* T-TApp: $t = t_1 T_2 \quad T = [X \mapsto T_2]T_{12} \quad \Gamma, x : Q, \Delta \vdash t_1 : (\forall X <: U_1.\ T_{12}) \quad \Gamma, x : Q, \Delta \vdash T_2 <: U_1$
     By Lemma 3.3, $\Gamma, \Delta \vdash T_2 <: U_1$. From the IH we get $\Gamma, \Delta \vdash [x \mapsto q]t_1 : (\forall X <: U_1.\ T_{12})$. $[x \mapsto q](t_1 T_2) = [x \mapsto q]t_1 T_2$ and the result follows from T-TApp.
   - *Case* T-Sub: $\Gamma, x : Q, \Delta \vdash t : S \quad \Gamma, x : Q, \Delta \vdash S <: T$
     By Lemma 3.3, $\Gamma, \Delta \vdash S <: T$. The result follows from the IH and T-Sub.
   - *Case* T-Class: $t = new\ C \quad T = \{new\ C\}$
     Since $[x \mapsto q]new\ C = new\ C$, using T-Class with context Γ, Δ directly leads to the desired result.

– *Case* T-Match: $\mathrm{t} = \mathrm{t}_s\ \mathit{match}\{\mathrm{x}_i : \mathrm{C}_i \Rightarrow \mathrm{t}_i\}\mathit{or}\ \mathrm{t}_d \quad \mathrm{T} = \mathrm{T}_s\ \mathit{match}\{\mathrm{C}_i \Rightarrow \mathrm{T}_i\}\mathit{or}\ \mathrm{T}_d$
$\Gamma, \mathrm{x} : \mathrm{Q}, \Delta \vdash \mathrm{t}_s : \mathrm{T}_s \quad \Gamma, \mathrm{x} : \mathrm{Q}, \Delta, \mathrm{x}_i : \mathrm{C}_i \vdash \mathrm{t}_i : \mathrm{T}_i \quad \Gamma, \mathrm{x} : \mathrm{Q}, \Delta \vdash \mathrm{t}_d : \mathrm{T}_d$
$[\mathrm{x} \mapsto \mathrm{q}](\mathrm{t}_s\ \mathit{match}\{\mathrm{x}_i : \mathrm{C}_i \Rightarrow \mathrm{t}_i\}\mathit{or}\ \mathrm{t}_d) = [\mathrm{x} \mapsto \mathrm{q}]\mathrm{t}_s\ \mathit{match}\{\mathrm{x}_i : \mathrm{C}_i \Rightarrow [\mathrm{x} \mapsto \mathrm{q}]\mathrm{t}_i\}\mathit{or}\ [\mathrm{x} \mapsto \mathrm{q}]\mathrm{t}_d$,
and the result follows from the IH and T-Match.

□

Lemma 3.5 (Disjointness/subtyping exclusivity).
The type disjointness and subtyping relations are mutually exclusive.

Proof: We prove mutual exclusivity of type disjointness and subtyping by first defining $[\![\cdot]\!]_\Gamma$, a mapping from System FM types (in a given context Γ) into non-empty subsets of a newly defined set P. We then show that the subtyping relation in FM corresponds to a subset relation in P, and that the type disjointness relation in FM (disj) corresponds to set disjointness relation in P.

This set-theoretical view of subtyping and disjointness renders the proof trivial. Indeed, suppose there exist two types S and T with $\Gamma \vdash \mathrm{S} <: \mathrm{T}$ and $\Gamma \vdash \mathrm{disj}(\mathrm{S}, \mathrm{T})$. $[\![\mathrm{S}]\!]_\Gamma$ and $[\![\mathrm{T}]\!]_\Gamma$ are two non-empty sets which are both intersecting and disjoint, a contradiction.

We first define P as $\mathrm{P} = \{\Lambda, \mathrm{V}\} \cup \mathrm{C}$. Elements of P can be understood as equivalence classes for System FM values: Λ corresponds to all abstraction values, V corresponds to type abstraction value, and elements of C correspond to their respective constructors. The definition of $[\![\cdot]\!]_\Gamma$ makes this correspondence apparent.

We define $[\![\cdot]\!]_\Gamma$, a mapping of System FM types into subsets of P in a given context Γ:

$$
\begin{aligned}
[\![\mathrm{Top}]\!]_\Gamma &= \mathrm{P} \\
[\![\mathrm{X}]\!]_\Gamma &= [\![\mathrm{T}]\!]_\Gamma \quad \text{where } \mathrm{X} <: \mathrm{T} \in \Gamma \\
[\![\mathrm{T}_1 \to \mathrm{T}_2]\!]_\Gamma &= \{\Lambda\} \\
[\![\forall \mathrm{X} <: \mathrm{U}_1.\ \mathrm{T}_2]\!]_\Gamma &= \{\mathrm{V}\} \\
[\![\{\mathit{new}\ \mathrm{C}_1\}]\!]_\Gamma &= \{\mathrm{C}_1\} \\
[\![\mathrm{C}_1]\!]_\Gamma &= \{\mathrm{c} \in \mathrm{C} \mid (\mathrm{c}, \mathrm{C}_1) \in \Psi\} \\
[\![\mathrm{T}_s\ \mathit{match}\{\mathrm{S}_i \Rightarrow \mathrm{T}_i\}\mathit{or}\ \mathrm{T}_d]\!]_\Gamma &= \begin{cases} [\![\mathrm{T}_n]\!]_\Gamma & \text{if } [\![\mathrm{T}_s]\!]_\Gamma \subset [\![\mathrm{S}_n]\!]_\Gamma \\ & \text{and } \forall m < n.\ [\![\mathrm{T}_s]\!]_\Gamma \cap [\![\mathrm{S}_m]\!]_\Gamma = \{\} \\ [\![\mathrm{T}_d]\!]_\Gamma & \text{if } \forall m.\ [\![\mathrm{T}_s]\!]_\Gamma \cap [\![\mathrm{S}_m]\!]_\Gamma = \{\} \\ \mathrm{P} & \text{otherwise} \end{cases}
\end{aligned}
$$

We show that the subtyping relation in FM corresponds to a subset relation in P, and that the type disjointness relation in FM (disj) corresponds to set disjointness in P. In other words, we prove the follows statements:

1. $\Gamma \vdash \mathrm{S} <: \mathrm{T}$ implies $[\![\mathrm{S}]\!]_\Gamma \subset [\![\mathrm{T}]\!]_\Gamma$,
2. $\Gamma \vdash \mathrm{disj}(\mathrm{S}, \mathrm{T})$ implies $[\![\mathrm{S}]\!]_\Gamma \cap [\![\mathrm{T}]\!]_\Gamma = \{\}$,

Both statements are proved simultaneously by induction on derivations of $\Gamma \vdash \mathrm{disj}(\mathrm{V}, \mathrm{W})$ and $\Gamma \vdash \mathrm{S} <: \mathrm{T}$. More precisely, the induction is done on the cumulative depth of both derivation tree.

1. ($\Gamma \vdash \mathrm{S} <: \mathrm{T}$ implies $[\![\mathrm{S}]\!]_\Gamma \subset [\![\mathrm{T}]\!]_\Gamma$)
   – *Case* S-Refl: $\mathrm{T} = \mathrm{S}$
   $[\![\mathrm{S}]\!]_\Gamma = [\![\mathrm{T}]\!]_\Gamma$ and the result is immediate.
   – *Case* S-Trans: $\Gamma \vdash \mathrm{S} <: \mathrm{U} \quad \Gamma \vdash \mathrm{U} <: \mathrm{T}$
   By the IH, $[\![\mathrm{S}]\!]_\Gamma \subset [\![\mathrm{U}]\!]_\Gamma$ and $[\![\mathrm{U}]\!]_\Gamma \subset [\![\mathrm{T}]\!]_\Gamma$. From subset transitivity, $[\![\mathrm{S}]\!]_\Gamma \subset [\![\mathrm{T}]\!]_\Gamma$, as required.

- *Case* S-Top: $S = \text{Top}$
  $[\![\text{Top}]\!]_\Gamma = P$ coincides with the codomain of $[\![\cdot]\!]_\Gamma$. Therefore, for any type T, $[\![T]\!]_\Gamma \subset [\![\text{Top}]\!]_\Gamma$, as required.
- *Case* S-Sin: $S = \{\mathit{new}\, C_1\} \quad T = C_1$
  $[\![\{\mathit{new}\, C_1\}]\!]_\Gamma = \{C_1\}$ and $[\![C_1]\!]_\Gamma = \{c \in C \mid (c, C_1) \in \Psi\}$. The result follows by reflexivity of Ψ.
- *Case* S-TVar: $S = X \quad X <: T \in \Gamma$
  By definition $[\![X]\!]_\Gamma = [\![T]\!]_\Gamma$, and the result is immediate.
- *Case* S-Arrow: $S = S_1 \to S_2 \quad T = T_1 \to T_2$
  $\Gamma \vdash T_1 <: S_1 \quad \Gamma \vdash S_2 <: T_2$
  $[\![S_1 \to S_2]\!]_\Gamma = [\![T_1 \to T_2]\!]_\Gamma = \{\Lambda\}$ and the result is immediate.
- *Case* S-All: $S = \forall X <: U_1.\, S_2 \quad T = \forall X <: U_1.\, T_2 \quad \Gamma, X <: U_1 \vdash S_2 <: T_2$
  $[\![\forall X <: U_1.\, S_2]\!]_\Gamma = [\![\forall X <: U_1.\, T_2]\!]_\Gamma = \{V\}$ and the result is immediate.
- *Case* S-Psi: $S = C_1 \quad T = C_2 \quad (C_1, C_2) \in \Psi$
  $[\![C_1]\!]_\Gamma = \{c \in C \mid (c, C_1) \in \Psi\}$ and $[\![C_2]\!]_\Gamma = \{c \in C \mid (c, C_2) \in \Psi\}$. By transitivity of Ψ, $[\![C_1]\!]_\Gamma \subset [\![C_2]\!]_\Gamma$, as required.
- *Case* S-Match1/2: $T_1 = T_n \quad T_2 = T_s\ \mathit{match}\{S_i \Rightarrow T_i\}\mathit{or}\, T_d$
  $\Gamma \vdash T_s <: S_n \quad \forall m < n.\ \Gamma \vdash \text{disj}(T_s, S_m)$
  Using the IH we get $[\![T_s]\!]_\Gamma \subset [\![S_n]\!]_\Gamma$, $\forall m < n.\ [\![T_s]\!]_\Gamma \cap [\![S_m]\!]_\Gamma = \{\}$ and $[\![T_s\ \mathit{match}\{S_i \Rightarrow T_i\}\mathit{or}\, T_d]\!]_\Gamma = [\![T_n]\!]_\Gamma$. This last equality implies both $[\![T_1]\!]_\Gamma \subset [\![T_2]\!]_\Gamma$ and $[\![T_2]\!]_\Gamma \subset [\![T_1]\!]_\Gamma$, as required.
- *Case* S-Match3/4: $T_1 = T_d \quad T_2 = T_s\ \mathit{match}\{S_i \Rightarrow T_i\}\mathit{or}\, T_d$
  $\forall n.\ \Gamma \vdash \text{disj}(T_s, S_n)$
  By the IH $\forall n.\ [\![T_s]\!]_\Gamma \cap [\![S_n]\!]_\Gamma = \{\}$. By definition, $[\![T_s\ \mathit{match}\{S_i \Rightarrow T_i\}\mathit{or}\, T_d]\!]_\Gamma = [\![T_d]\!]_\Gamma$. This equality implies both $[\![T_1]\!]_\Gamma \subset [\![T_2]\!]_\Gamma$ and $[\![T_2]\!]_\Gamma \subset [\![T_1]\!]_\Gamma$, the desired result for S-Match3 and S-Match4 respectively.
- *Case* S-Match5: $S = S_s\ \mathit{match}\{U_i \Rightarrow S_i\}\mathit{or}\, S_d \quad T = T_s\ \mathit{match}\{U_i \Rightarrow T_i\}\mathit{or}\, T_d$
  $\Gamma \vdash S_s <: T_s \quad \forall n.\ \Gamma \vdash S_n <: T_n \quad \Gamma \vdash S_d <: T_d$
  By case analysis on $[\![T]\!]_\Gamma$:
  a. $[\![T]\!]_\Gamma = [\![T_n]\!]_\Gamma$ and $[\![T_s]\!]_\Gamma \subset [\![U_n]\!]_\Gamma$ and $\forall m < n.\ [\![T_s]\!]_\Gamma \cap [\![U_m]\!]_\Gamma = \{\}$:
     The IH gives $[\![S_s]\!]_\Gamma \subset [\![T_s]\!]_\Gamma$. Using $\forall m < n.\ [\![T_s]\!]_\Gamma \cap [\![U_m]\!]_\Gamma = \{\}$, we get $\forall m < n.\ [\![S_s]\!]_\Gamma \cap [\![U_m]\!]_\Gamma = \{\}$. Using $[\![T_s]\!]_\Gamma \subset [\![U_n]\!]_\Gamma$, we get $[\![S_s]\!]_\Gamma \subset [\![U_n]\!]_\Gamma$ (by subset transitivity). Therefore, $[\![S]\!]_\Gamma = [\![S_n]\!]_\Gamma$, and the result follows from the IH.
  b. $[\![T]\!]_\Gamma = [\![T_d]\!]_\Gamma$ and $\forall n.\ [\![T_s]\!]_\Gamma \cap [\![U_n]\!]_\Gamma = \{\}$:
     The IH gives $[\![S_s]\!]_\Gamma \subset [\![T_s]\!]_\Gamma$. Using $\forall n.\ [\![T_s]\!]_\Gamma \cap [\![U_n]\!]_\Gamma = \{\}$ we get $\forall n.\ [\![S_s]\!]_\Gamma \cap [\![U_n]\!]_\Gamma = \{\}$. Therefore, $[\![S]\!]_\Gamma = [\![S_d]\!]_\Gamma$, and the result follows from the IH.
  c. $[\![T]\!]_\Gamma = P$:
     P is the codomain of $[\![\cdot]\!]_\Gamma$, therefore $[\![S]\!]_\Gamma \subset [\![T]\!]_\Gamma$, as required.

2. ($\Gamma \vdash \text{disj}(S, T)$ implies $[\![S]\!]_\Gamma \cap [\![T]\!]_\Gamma = \{\}$)
   - *Case* D-Xi: $S = C_1 \quad T = C_2 \quad (C_1, C_2) \in \Xi$
     $[\![C_1]\!]_\Gamma = \{c \in C \mid (c, C_1) \in \Psi\}$ and $[\![C_2]\!]_\Gamma = \{c \in C \mid (c, C_2) \in \Psi\}$. By definition of Ξ, there is no class c such that both $(c, C_1) \in \Psi$ and $(c, C_2) \in \Psi$, and $[\![C_1]\!]_\Gamma \cap [\![C_2]\!]_\Gamma = \{\}$.
   - *Case* D-Psi: $S = \{\mathit{new}\, C_1\} \quad T = C_2 \quad (C_1, C_2) \notin \Psi$
     $[\![\{\mathit{new}\, C_1\}]\!]_\Gamma = C_1$ and $[\![C_2]\!]_\Gamma = \{c \in C \mid (c, C_2) \in \Psi\}$. Since $(C_1, C_2) \notin \Psi$, $C_1 \notin [\![C_2]\!]_\Gamma$ and $[\![C_1]\!]_\Gamma \cap [\![C_2]\!]_\Gamma = \{\}$.
   - *Case* D-Sub: $\Gamma \vdash S <: U \quad \Gamma \vdash \text{disj}(U, T)$
     By the IH, $[\![S]\!]_\Gamma \subset [\![U]\!]_\Gamma$ and $[\![U]\!]_\Gamma \cap [\![T]\!]_\Gamma = \{\}$. $[\![S]\!]_\Gamma \cap [\![T]\!]_\Gamma = \{\}$ follows directly.
   - *Case* D-Arrow: $S = S_1 \to S_2 \quad T = C_1$

$$\frac{\vdots}{\Gamma \vdash S =:= T}\ (\text{S-Match1/2}) \qquad \frac{\vdots}{\Gamma \vdash S =:= T}\ (\text{S-Match3/4}) \qquad \frac{\Gamma \vdash S \rightleftharpoons U \quad \Gamma \vdash U \rightleftharpoons T}{\Gamma \vdash S \rightleftharpoons T}$$

Fig. 1. Definition of the auxiliary relation $\rightleftharpoons$, used to state inversion of subtyping.

$[\![C_1]\!]_\Gamma = \{c \in C \mid (c, C_1) \in \Psi\}$, $[\![S_1 \to S_2]\!]_\Gamma = \{\Lambda\}$ and $[\![C_1]\!]_\Gamma \cap [\![S_1 \to S_2]\!]_\Gamma = \{\}$, as required.

- *Case* D-All: $S = \forall X <: S_1.\ S_2 \quad T = C_1$
  $[\![C_1]\!]_\Gamma = \{c \in C \mid (c, C_1) \in \Psi\}$, $[\![\forall X <: S_1.\ S_2]\!]_\Gamma = \{S\}$ and $[\![C_1]\!]_\Gamma \cap [\![\forall X <: S_1.\ S_2]\!]_\Gamma = \{\}$, as required.

□

Definition. $\Gamma \vdash S \rightleftharpoons T$ *(defined in Figure 1) represents evidence of the mutual subtyping between a match type* S *and a type* T *with the additional constraint that this evidence was exclusively constructed using pairwise applications of S-Match1/2, S-Match3/4, and S-Trans in both directions.*

Lemma 3.6 (Inversion of subtyping).

1. *If* $\Gamma \vdash S_s$ *match* $\{U_i \Rightarrow S_i\}$ *or* $S_d \rightleftharpoons T$*, then either:*
   a. $\Gamma \vdash S_s <: U_n$, $\forall m < n.\ \Gamma \vdash \text{disj}(S_s, U_m)$ *and* S_n *is a match type with* $\Gamma \vdash S_n \rightleftharpoons T$*,*
   b. $\Gamma \vdash S_s <: U_n$, $\forall m < n.\ \Gamma \vdash \text{disj}(S_s, U_m)$ *and* $S_n = T$*,*
   c. $\forall n.\ \Gamma \vdash \text{disj}(S_s, U_n)$ *and* S_d *is a match type with* $\Gamma \vdash S_d \rightleftharpoons T$*,*
   d. $\forall n.\ \Gamma \vdash \text{disj}(S_s, U_n)$ *and* $S_d = T$*.*
2. *If* $\Gamma \vdash S <: X$*, or* $\Gamma \vdash S <: T$ *where* T *is a match type with* $\Gamma \vdash T \rightleftharpoons X$*, then either*
   a. S *is a match type with* $\Gamma \vdash S \rightleftharpoons Y$*, for some* Y*,*
   b. S *is a type variable.*
3. *If* $\Gamma \vdash S <: T_1 \to T_2$*, or* $\Gamma \vdash S <: T$ *where* T *is a match type with* $\Gamma \vdash T \rightleftharpoons T_1 \to T_2$*, then either*
   a. S *is a match type with* $\Gamma \vdash S \rightleftharpoons S_1 \to S_2$*, for some* S_1, S_2 *such that* $\Gamma \vdash T_1 <: S_1$ *and* $\Gamma \vdash S_2 <: T_2$*,*
   b. S *is a match type with* $\Gamma \vdash S \rightleftharpoons X$*, for some* X*,*
   c. S *is a type variable,*
   d. S *has the form* $S_1 \to S_2$ *with* $\Gamma \vdash T_1 <: S_1$ *and* $\Gamma \vdash S_2 <: T_2$*.*
4. *If* $\Gamma \vdash S <: \forall X <: U_1.\ T_2$*, or* $\Gamma \vdash S <: T$ *where* T *is a match type with* $\Gamma \vdash T \rightleftharpoons \forall X <: U_1.\ T_2$*, then either*
   a. S *is a match type with* $\Gamma \vdash S \rightleftharpoons \forall X <: U_1.\ S_2$*, for some* S_2 *such that* $\Gamma, X <: U_1 \vdash S_2 <: T_2$*,*
   b. S *is a match type with* $\Gamma \vdash S \rightleftharpoons X$*, for some* X*,*
   c. S *is a type variable,*
   d. S *has the form* $\forall X <: U_1.\ S_2$ *and* $\Gamma, X <: U_1 \vdash S_2 <: T_2$*.*

Proof:

1. By induction on a derivation on $\Gamma \vdash S \rightleftharpoons T$.
   - *Case* S-Match1/2: $S = S_s$ *match* $\{U_i \Rightarrow S_i\}$ *or* $S_d \quad T = S_n$
     $\Gamma \vdash S_s <: U_n \quad \forall m < n.\ \Gamma \vdash \text{disj}(S_s, U_m)$
     $S_n = T$ and the result follows directly from the premises.
   - *Case* S-Match3/4: $S = S_s$ *match* $\{U_i \Rightarrow S_i\}$ *or* $S_d \quad T = S_d \quad \forall n.\ \Gamma \vdash \text{disj}(S_s, U_n)$
     $S_d = T$ and the result follows directly from the premises.
   - *Case* S-Trans: $\Gamma \vdash S \rightleftharpoons U \quad \Gamma \vdash U \rightleftharpoons T$
     By definition of the $\rightleftharpoons$ relation, $\Gamma \vdash U \rightleftharpoons T$ implies that U is a match type. Using the IH on the left premise we get 4 subcases:

   a. $\Gamma \vdash S_s <: U_n$, $\forall m < n.\ \Gamma \vdash \text{disj}(S_s, U_m)$ and S_n is a match type with $\Gamma \vdash S_n \rightleftharpoons U$:
      Using S-Trans we get $\Gamma \vdash S_n \rightleftharpoons T$, as requiered.
   b. $\Gamma \vdash S_s <: U_n$, $\forall m < n.\ \Gamma \vdash \text{disj}(S_s, U_m)$ and $S_n = U$:
      The result is immediate.
   c. $\forall n.\ \Gamma \vdash \text{disj}(S_s, U_n)$ and S_d is a match type with $\Gamma \vdash S_d \rightleftharpoons U$:
      Using S-Trans we get $\Gamma \vdash S_d \rightleftharpoons T$, as requiered.
   d. $\forall n.\ \Gamma \vdash \text{disj}(S_s, U_n)$ and $S_d = U$:
      The result is immediate.

2. By induction on a derivation of $\Gamma \vdash S <: T$.
   - *Case* S-Refl: $S = T$
     If T is a type variable then so is S and the result is immediate. If T is a match type with $\Gamma \vdash T \rightleftharpoons X$, S is a match type and $\Gamma \vdash S \rightleftharpoons Y$, as required.
   - *Case* S-Trans: $\Gamma \vdash S <: U \quad \Gamma \vdash U <: T$
     If T is a type variable we use the IH on the right premise to get that U is either a match type with $\Gamma \vdash U \rightleftharpoons X$ or a type variable. In either case, the result follows from using the IH on the left premise.
     If T is a match type with $\Gamma \vdash T \rightleftharpoons X$ we can also use the the IH on the right premise to get to the same result.
   - *Case* S-TVar: $S = Y \quad Y <: X \in \Gamma$
     S is a type variable and the result is immediate.
   - *Case* S-Match1: $S = T_s\ \mathit{match}\{U_i \Rightarrow T_i\}\ \mathit{or}\ T_d \quad T = T_n$
     $\Gamma \vdash T_s <: U_n \quad \forall m < n.\ \Gamma \vdash \text{disj}(T_s, U_m)$
     If $T = X$, we use S-Match2 to obtain $\Gamma \vdash S \rightleftharpoons X$, as required. If T is a match type with $\Gamma \vdash T \rightleftharpoons X$, we use S-Match2 to get $\Gamma \vdash S \rightleftharpoons T$, and S-Trans to obtain $\Gamma \vdash S \rightleftharpoons X$, as required.
   - *Case* S-Match2: $S = T_n \quad T = T_s\ \mathit{match}\{U_i \Rightarrow T_i\}\ \mathit{or}\ T_d$
     $\Gamma \vdash T_s <: U_n \quad \forall m < n.\ \Gamma \vdash \text{disj}(T_s, U_m)$
     In this case the first premise of the statement ($\Gamma \vdash S <: X$) does not apply since T is a match type. Using the 1st part of the lemma on $\Gamma \vdash T \rightleftharpoons X$, we get 4 subcases:
     a. $\Gamma \vdash T_s <: U_n$, $\forall m < n.\ \Gamma \vdash \text{disj}(T_s, U_m)$ and T_n is a match type with $\Gamma \vdash T_n \rightleftharpoons X$:
        $S = T_n$ is a match type and $\Gamma \vdash S \rightleftharpoons X$, as required.
     b. $\Gamma \vdash T_s <: U_n$, $\forall m < n.\ \Gamma \vdash \text{disj}(T_s, U_m)$ and $T_n = X$:
        $S = T_n = X$ is a type variable and the result is immediate.
     c. $\forall m.\ \Gamma \vdash \text{disj}(T_s, U_m)$ and T_d is a match type with $\Gamma \vdash T_d \rightleftharpoons X$:
        This case cannot occur since $\Gamma \vdash \text{disj}(T_s, U_n)$ contradicts $\Gamma \vdash T_s <: U_n$ (Lemma 3.5).
     d. $\forall m.\ \Gamma \vdash \text{disj}(T_s, U_m)$ and $T_d = X$:
        This case cannot occur since $\Gamma \vdash \text{disj}(T_s, U_n)$ contradicts $\Gamma \vdash T_s <: U_n$ (Lemma 3.5).
   - *Case* S-Match3: $S = T_s\ \mathit{match}\{U_i \Rightarrow T_i\}\ \mathit{or}\ T_d \quad T = T_d$
     $\forall n.\ \Gamma \vdash \text{disj}(T_s, U_n)$
     If $T = X$, we use S-Match4 to obtain $\Gamma \vdash S \rightleftharpoons X$, as required. If T is a match type with $\Gamma \vdash T \rightleftharpoons X$, we use S-Match4 to get $\Gamma \vdash S \rightleftharpoons T$, and S-Trans to obtain $\Gamma \vdash S \rightleftharpoons X$, as required.
   - *Case* S-Match4: $S = T_d \quad T = T_s\ \mathit{match}\{U_i \Rightarrow T_i\}\ \mathit{or}\ T_d$
     $\forall n.\ \Gamma \vdash \text{disj}(T_s, U_n)$
     In this case the first premise of the statement ($\Gamma \vdash S <: X$) does not apply since T is a match type. Using the 1st part of the lemma on $\Gamma \vdash T \rightleftharpoons X$, we get 4 subcases:
     a. $\Gamma \vdash T_s <: U_n$, $\forall m < n.\ \Gamma \vdash \text{disj}(T_s, U_m)$ and T_n is a match type with $\Gamma \vdash T_n \rightleftharpoons X$:
        This case cannot occur since $\Gamma \vdash \text{disj}(T_s, U_n)$ contradicts $\Gamma \vdash T_s <: U_n$ (Lemma 3.5).
     b. $\Gamma \vdash T_s <: U_n$, $\forall m < n.\ \Gamma \vdash \text{disj}(T_s, U_m)$ and $T_n = X$:
        This case cannot occur since $\Gamma \vdash \text{disj}(T_s, U_n)$ contradicts $\Gamma \vdash T_s <: U_n$ (Lemma 3.5).

c. $\forall m.\ \Gamma \vdash \mathrm{disj}(\mathrm{T}_s, \mathrm{U}_m)$ and T_d is a match type with $\Gamma \vdash \mathrm{T}_d \rightleftharpoons \mathrm{X}$:
S is a match type and $\Gamma \vdash \mathrm{S} \rightleftharpoons \mathrm{X}$, as required.
d. $\forall m.\ \Gamma \vdash \mathrm{disj}(\mathrm{T}_s, \mathrm{U}_m)$ and $\mathrm{T}_d = \mathrm{X}$:
$\mathrm{S} = \mathrm{T}_d = \mathrm{X}$ is a type variable and the result is immediate.

– *Case* S-Match5: $\mathrm{S} = \mathrm{S}_s\ \mathit{match}\{\mathrm{U}_i \Rightarrow \mathrm{S}_i\}\ \mathit{or}\ \mathrm{S}_d \quad \mathrm{T} = \mathrm{T}_s\ \mathit{match}\{\mathrm{U}_i \Rightarrow \mathrm{T}_i\}\ \mathit{or}\ \mathrm{T}_d$
$\Gamma \vdash \mathrm{S}_s <: \mathrm{T}_s \quad \forall m.\ \Gamma \vdash \mathrm{S}_m <: \mathrm{T}_m \quad \Gamma \vdash \mathrm{S}_d <: \mathrm{T}_d$
In this case the first premise of the statement ($\Gamma \vdash \mathrm{S} <: \mathrm{X}$) does not apply since T is a match type. Using the 1st part of the lemma on $\Gamma \vdash \mathrm{T} \rightleftharpoons \mathrm{X}$, we get 4 subcases:
a. $\Gamma \vdash \mathrm{T}_s <: \mathrm{U}_n$, $\forall m < n.\ \Gamma \vdash \mathrm{disj}(\mathrm{T}_s, \mathrm{U}_m)$ and T_n is a match type with $\Gamma \vdash \mathrm{T}_n \rightleftharpoons \mathrm{X}$:
Using D-Sub on $\forall m.\ \Gamma \vdash \mathrm{S}_m <: \mathrm{T}_m$ and $\forall m < n.\ \Gamma \vdash \mathrm{disj}(\mathrm{T}_s, \mathrm{U}_m)$ we obtain $\forall m < n.\ \Gamma \vdash \mathrm{disj}(\mathrm{S}_s, \mathrm{U}_m)$. From S-Trans we get $\Gamma \vdash \mathrm{S}_s <: \mathrm{U}_n$. Using S-Match1/2 we obtain $\Gamma \vdash \mathrm{S} \rightleftharpoons \mathrm{S}_n$. Since $\Gamma \vdash \mathrm{S}_n <: \mathrm{T}_n$ and $\Gamma \vdash \mathrm{T}_n \rightleftharpoons \mathrm{X}$, we use the IH to get that either S_n is a match type with $\Gamma \vdash \mathrm{S}_n \rightleftharpoons \mathrm{Y}$, or S_n is a type variable. If S_n is a type variable, we already proved $\Gamma \vdash \mathrm{S} \rightleftharpoons \mathrm{S}_n$, as required. If $\Gamma \vdash \mathrm{S}_n \rightleftharpoons \mathrm{Y}$, then using S-Trans with $\Gamma \vdash \mathrm{S} \rightleftharpoons \mathrm{S}_n$ gives $\Gamma \vdash \mathrm{S} \rightleftharpoons \mathrm{Y}$, as required.
b. $\Gamma \vdash \mathrm{T}_s <: \mathrm{U}_n$, $\forall m < n.\ \Gamma \vdash \mathrm{disj}(\mathrm{T}_s, \mathrm{U}_m)$ and $\mathrm{T}_n = \mathrm{X}$:
Using D-Sub on $\forall m.\ \Gamma \vdash \mathrm{S}_m <: \mathrm{T}_m$ and $\forall m < n.\ \Gamma \vdash \mathrm{disj}(\mathrm{T}_s, \mathrm{U}_m)$ we obtain $\forall m < n.\ \Gamma \vdash \mathrm{disj}(\mathrm{S}_s, \mathrm{U}_m)$. From S-Trans we get $\Gamma \vdash \mathrm{S}_s <: \mathrm{U}_n$. Using S-Match1/2 we obtain $\Gamma \vdash \mathrm{S} \rightleftharpoons \mathrm{S}_n$. Since $\Gamma \vdash \mathrm{S}_n <: \mathrm{X}$, we use the IH to get that either S_n is a match type with $\Gamma \vdash \mathrm{S}_n \rightleftharpoons \mathrm{Y}$, or S_n is a type variable. If S_n is a type variable, we already proved $\Gamma \vdash \mathrm{S} \rightleftharpoons \mathrm{S}_n$, as required. If $\Gamma \vdash \mathrm{S}_n \rightleftharpoons \mathrm{Y}$, then using S-Trans with $\Gamma \vdash \mathrm{S} \rightleftharpoons \mathrm{S}_n$ gives $\Gamma \vdash \mathrm{S} \rightleftharpoons \mathrm{Y}$, as required.
c. $\forall m.\ \Gamma \vdash \mathrm{disj}(\mathrm{T}_s, \mathrm{U}_m)$ and T_d is a match type with $\Gamma \vdash \mathrm{T}_d \rightleftharpoons \mathrm{X}$:
Using D-Sub on $\forall m.\ \Gamma \vdash \mathrm{S}_m <: \mathrm{T}_m$ and $\forall m.\ \Gamma \vdash \mathrm{disj}(\mathrm{T}_s, \mathrm{U}_m)$ we obtain $\forall m.\ \Gamma \vdash \mathrm{disj}(\mathrm{S}_s, \mathrm{U}_m)$. Using S-Match3/4 we obtain $\Gamma \vdash \mathrm{S} \rightleftharpoons \mathrm{S}_d$. Since $\Gamma \vdash \mathrm{S}_d <: \mathrm{T}_d$ and $\Gamma \vdash \mathrm{T}_d \rightleftharpoons \mathrm{X}$, we use the IH to get that either S_d is a match type with $\Gamma \vdash \mathrm{S}_d \rightleftharpoons \mathrm{Y}$, or S_d is a type variable. If S_d is a type variable, we already proved $\Gamma \vdash \mathrm{S} \rightleftharpoons \mathrm{S}_d$, as required. If $\Gamma \vdash \mathrm{S}_d \rightleftharpoons \mathrm{Y}$, then using S-Trans with $\Gamma \vdash \mathrm{S} \rightleftharpoons \mathrm{S}_d$ gives $\Gamma \vdash \mathrm{S} \rightleftharpoons \mathrm{Y}$, as required.
d. $\forall m.\ \Gamma \vdash \mathrm{disj}(\mathrm{T}_s, \mathrm{U}_m)$ and $\mathrm{T}_d = \mathrm{X}$:
Using D-Sub on $\forall m.\ \Gamma \vdash \mathrm{S}_m <: \mathrm{T}_m$ and $\forall m.\ \Gamma \vdash \mathrm{disj}(\mathrm{T}_s, \mathrm{U}_m)$ we obtain $\forall m.\ \Gamma \vdash \mathrm{disj}(\mathrm{S}_s, \mathrm{U}_m)$. Using S-Match3/4 we obtain $\Gamma \vdash \mathrm{S} <: \mathrm{S}_d$. Since $\Gamma \vdash \mathrm{S}_d <: \mathrm{T}_d$ and $\Gamma \vdash \mathrm{T}_d \rightleftharpoons \mathrm{X}$, we use the IH to get that either S_d is a match type with $\Gamma \vdash \mathrm{S}_d \rightleftharpoons \mathrm{Y}$, or S_d is a type variable. If S_d is a type variable, we already proved $\Gamma \vdash \mathrm{S} \rightleftharpoons \mathrm{S}_d$, as required. If $\Gamma \vdash \mathrm{S}_d \rightleftharpoons \mathrm{Y}$, then using S-Trans with $\Gamma \vdash \mathrm{S} \rightleftharpoons \mathrm{S}_d$ gives $\Gamma \vdash \mathrm{S} \rightleftharpoons \mathrm{Y}$, as required.

– *Case* S-Top, S-Sin, S-Arrow, S-All, S-Psi:
In those cases, T is neither a type variable nor a match type and the result is immediate.

3. By induction on a derivation of $\Gamma \vdash \mathrm{S} <: \mathrm{T}$.
– *Case* S-Refl: $\mathrm{T} = \mathrm{S}$
If T is a match type with $\Gamma \vdash \mathrm{T} \rightleftharpoons \mathrm{T}_1 \to \mathrm{T}_2$ and the result follows directly from S-Refl. If $\mathrm{T} = \mathrm{T}_1 \to \mathrm{T}_2$, the result also follows directly from S-Refl.
– *Case* S-Trans: $\Gamma \vdash \mathrm{S} <: \mathrm{U} \quad \Gamma \vdash \mathrm{U} <: \mathrm{T}$
Using the IH on the right premise we get 4 subcases:
a. U is a match type with $\Gamma \vdash \mathrm{U} \rightleftharpoons \mathrm{U}_1 \to \mathrm{U}_2$, for some $\mathrm{U}_1, \mathrm{U}_2$ such that $\Gamma \vdash \mathrm{T}_1 <: \mathrm{U}_1$ and $\Gamma \vdash \mathrm{U}_2 <: \mathrm{T}_2$:
The result follows directly from using the IH on the left premise ($\Gamma \vdash \mathrm{T}_1 <: \mathrm{S}_1$ is obtained using S-Trans with $\Gamma \vdash \mathrm{T}_1 <: \mathrm{U}_1$ and $\Gamma \vdash \mathrm{U}_1 <: \mathrm{S}_1$, $\Gamma \vdash \mathrm{S}_2 <: \mathrm{T}_2$ is obtained analogously).
b. U is a match type with $\Gamma \vdash \mathrm{U} \rightleftharpoons \mathrm{X}$, for some X:

Using the 2nd part of the lemma on the left premise leads to the desired result.

c. U is a type variable:
The result follows from using the 2nd part of the lemma on the left premise.

d. U has the form $U_1 \to U_2$ with $\Gamma \vdash T_1 <: U_1$ and $\Gamma \vdash U_2 <: T_2$:
The result follows directly from using the IH on the left premise ($\Gamma \vdash T_1 <: S_1$ is obtained using S-Trans with $\Gamma \vdash T_1 <: U_1$ and $\Gamma \vdash U_1 <: S_1$, $\Gamma \vdash S_2 <: T_2$ is obtained analogously).

– *Case* S-Match1: $S = T_s \ \mathit{match}\{U_i \Rightarrow T_i\} \mathit{or}\ T_d \quad T = T_n \quad \Gamma \vdash T_s <: U_n \quad \forall m < n.\ \Gamma \vdash \mathrm{disj}(T_s, U_m)$

If $T = T_1 \to T_2$, we use S-Match2 to obtain $\Gamma \vdash S \rightleftharpoons T_1 \to T_2$, as required. If T is a match type with $\Gamma \vdash T \rightleftharpoons T_1 \to T_2$, we use S-Match2 to get $\Gamma \vdash S \rightleftharpoons T$, and S-Trans to obtain $\Gamma \vdash S \rightleftharpoons T_1 \to T_2$, and the result follows from S-Refl.

– *Case* S-Match2: $S = T_n \quad T = T_s \ \mathit{match}\{U_i \Rightarrow T_i\} \mathit{or}\ T_d \quad \Gamma \vdash T_s <: U_n \quad \forall m < n.\ \Gamma \vdash \mathrm{disj}(T_s, U_m)$

In this case the first premise of the statement ($\Gamma \vdash S <: T_1 \to T_2$) does not apply since T is a match type. Using the 1st part of the lemma on $\Gamma \vdash T \rightleftharpoons T_1 \to T_2$, we get 4 subcases:

a. $\Gamma \vdash T_s <: U_n$, $\forall m < n.\ \Gamma \vdash \mathrm{disj}(T_s, U_m)$ and T_n is a match type with $\Gamma \vdash T_n \rightleftharpoons T_1 \to T_2$:
$S = T_n$ is a match type and $\Gamma \vdash S \rightleftharpoons T_1 \to T_2$, and the result follows from S-Refl.

b. $\Gamma \vdash T_s <: U_n$, $\forall m < n.\ \Gamma \vdash \mathrm{disj}(T_s, U_m)$ and $T_n = T_1 \to T_2$:
$S = T_1 \to T_2$, $S_1 = T_1$, $S_2 = T_2$, and the result follows from S-Refl.

c. $\forall m.\ \Gamma \vdash \mathrm{disj}(T_s, U_m)$ and T_d is a match type with $\Gamma \vdash T_d \rightleftharpoons T_1 \to T_2$:
This case cannot occur since $\Gamma \vdash \mathrm{disj}(T_s, U_n)$ contradicts $\Gamma \vdash T_s <: U_n$ (Lemma 3.5).

d. $\forall m.\ \Gamma \vdash \mathrm{disj}(T_s, U_m)$ and $T_d = T_1 \to T_2$:
This case cannot occur since $\Gamma \vdash \mathrm{disj}(T_s, U_n)$ contradicts $\Gamma \vdash T_s <: U_n$ (Lemma 3.5).

– *Case* S-Match3: $S = T_s \ \mathit{match}\{U_i \Rightarrow T_i\} \mathit{or}\ T_d \quad T = T_d \quad \forall n.\ \Gamma \vdash \mathrm{disj}(T_s, U_n)$

If $T = T_1 \to T_2$, we use S-Match2 to obtain $\Gamma \vdash S \rightleftharpoons T_1 \to T_2$, and the result follows from S-Refl. If T is a match type with $\Gamma \vdash T \rightleftharpoons T_1 \to T_2$, we use S-Match2 to get $\Gamma \vdash S \rightleftharpoons T$, and S-Trans to obtain $\Gamma \vdash S \rightleftharpoons T_1 \to T_2$, and the result follows from S-Refl.

– *Case* S-Match4: $S = T_d \quad T = T_s \ \mathit{match}\{U_i \Rightarrow T_i\} \mathit{or}\ T_d \quad \forall n.\ \Gamma \vdash \mathrm{disj}(T_s, U_n)$

In this case the first premise of the statement ($\Gamma \vdash S <: T_1 \to T_2$) does not apply since T is a match type. Using the 1st part of the lemma on $\Gamma \vdash T \rightleftharpoons T_1 \to T_2$, we get 4 subcases:

a. $\Gamma \vdash T_s <: U_n$, $\forall m < n.\ \Gamma \vdash \mathrm{disj}(T_s, U_m)$ and T_n is a match type with $\Gamma \vdash T_n \rightleftharpoons T_1 \to T_2$:
This case cannot occur since $\Gamma \vdash \mathrm{disj}(T_s, U_n)$ contradicts $\Gamma \vdash T_s <: U_n$ (Lemma 3.5).

b. $\Gamma \vdash T_s <: U_n$, $\forall m < n.\ \Gamma \vdash \mathrm{disj}(T_s, U_m)$ and $T_n = T_1 \to T_2$:
This case cannot occur since $\Gamma \vdash \mathrm{disj}(T_s, U_n)$ contradicts $\Gamma \vdash T_s <: U_n$ (Lemma 3.5).

c. $\forall m.\ \Gamma \vdash \mathrm{disj}(T_s, U_m)$ and T_d is a match type with $\Gamma \vdash T_d \rightleftharpoons T_1 \to T_2$:
$S = T_d$ is a match type and $\Gamma \vdash S \rightleftharpoons T_1 \to T_2$, and the result follows from S-Refl.

d. $\forall m.\ \Gamma \vdash \mathrm{disj}(T_s, U_m)$ and $T_d = T_1 \to T_2$:
$S = T_1 \to T_2$, $S_1 = T_1$, $S_2 = T_2$, and the result follows from S-Refl.

– *Case* S-Match5: $S = S_s \ \mathit{match}\{U_i \Rightarrow S_i\} \mathit{or}\ S_d \quad T = T_s \ \mathit{match}\{U_i \Rightarrow T_i\} \mathit{or}\ T_d \quad \Gamma \vdash S_s <: T_s \quad \forall n.\ \Gamma \vdash S_n <: T_n \quad \Gamma \vdash S_d <: T_d$

In this case the first premise of the statement ($\Gamma \vdash S <: T_1 \to T_2$) does not apply since T is a match type. Using the 1st part of the lemma on $\Gamma \vdash T \rightleftharpoons T_1 \to T_2$, we get 4 subcases:

a. $\Gamma \vdash T_s <: U_n$, $\forall m < n.\ \Gamma \vdash \mathrm{disj}(T_s, U_m)$ and T_n is a match type with $\Gamma \vdash T_n \rightleftharpoons T_1 \to T_2$:
Using D-Sub on $\forall m.\ \Gamma \vdash S_m <: T_m$ and $\forall m < n.\ \Gamma \vdash \mathrm{disj}(T_s, U_m)$ we obtain $\forall m < n.\ \Gamma \vdash \mathrm{disj}(S_s, U_m)$. From S-Trans we get $\Gamma \vdash S_s <: U_n$. Using S-Match1/2 we obtain $\Gamma \vdash S \rightleftharpoons S_n$. Since $\Gamma \vdash S_n <: T_n$ and $\Gamma \vdash T_n \rightleftharpoons T_1 \to T_2$, the IH gives 4 subsubcases:
   (i) S_n is a match type with $\Gamma \vdash S_n \rightleftharpoons S_1 \to S_2$ for some S_1, S_2 such that $\Gamma \vdash T_1 <: S_1$ and $\Gamma \vdash S_2 <: T_2$:
   The result follows directly from S-Trans.
   (ii) S_n is a match type with $\Gamma \vdash S_n \rightleftharpoons X$:
   The result follows directly from S-Trans.
   (iii) S_n is a type variable:
   We already proved $\Gamma \vdash S \rightleftharpoons S_n$, as required.
   (iv) S_n has the form $S_1 \to S_2$ with $\Gamma \vdash T_1 <: S_1$ and $\Gamma \vdash S_2 <: T_2$:
   We already proved $\Gamma \vdash S \rightleftharpoons S_n$, as required.

b. $\Gamma \vdash T_s <: U_n$, $\forall m < n.\ \Gamma \vdash \mathrm{disj}(T_s, U_m)$ and $T_n = T_1 \to T_2$:
Using D-Sub on $\forall m.\ \Gamma \vdash S_m <: T_m$ and $\forall m < n.\ \Gamma \vdash \mathrm{disj}(T_s, U_m)$ we obtain $\forall m < n.\ \Gamma \vdash \mathrm{disj}(S_s, U_m)$. From S-Trans we get $\Gamma \vdash S_s <: U_n$. Using S-Match1/2 we obtain $\Gamma \vdash S \rightleftharpoons S_n$. Since $\Gamma \vdash S_n <: T_n$, the IH gives 4 subsubcases:
   (i) S_n is a match type with $\Gamma \vdash S_n \rightleftharpoons S_1 \to S_2$ for some S_1, S_2 such that $\Gamma \vdash T_1 <: S_1$ and $\Gamma \vdash S_2 <: T_2$:
   The result follows directly from S-Trans.
   (ii) S_n is a match type with $\Gamma \vdash S_n \rightleftharpoons X$:
   The result follows directly from S-Trans.
   (iii) S_n is a type variable:
   We already proved $\Gamma \vdash S \rightleftharpoons S_n$, as required.
   (iv) S_n has the form $S_1 \to S_2$ with $\Gamma \vdash T_1 <: S_1$ and $\Gamma \vdash S_2 <: T_2$:
   We already proved $\Gamma \vdash S \rightleftharpoons S_n$, as required.

c. $\forall n.\ \Gamma \vdash \mathrm{disj}(T_s, U_n)$ and T_d is a match type with $\Gamma \vdash T_d \rightleftharpoons T_1 \to T_2$:
Using D-Sub on $\forall n.\ \Gamma \vdash S_n <: T_n$ and $\forall n.\ \Gamma \vdash \mathrm{disj}(T_s, U_n)$ we obtain $\forall n.\ \Gamma \vdash \mathrm{disj}(S_s, U_n)$. From S-Trans we get $\Gamma \vdash S_s <: C_d$. Using S-Match3/4 we obtain $\Gamma \vdash S \rightleftharpoons S_d$. Since $\Gamma \vdash S_d <: T_d$ and $\Gamma \vdash T_d \rightleftharpoons T_1 \to T_2$, the IH gives 4 subsubcases:
   (i) S_d is a match type with $\Gamma \vdash S_d \rightleftharpoons S_1 \to S_2$ for some S_1, S_2 such that $\Gamma \vdash T_1 <: S_1$ and $\Gamma \vdash S_2 <: T_2$:
   The result follows directly from S-Trans.
   (ii) S_d is a match type with $\Gamma \vdash S_d \rightleftharpoons X$:
   The result follows directly from S-Trans.
   (iii) S_d is a type variable:
   We already proved $\Gamma \vdash S \rightleftharpoons S_d$, as required.
   (iv) S_d has the form $S_1 \to S_2$ with $\Gamma \vdash T_1 <: S_1$ and $\Gamma \vdash S_2 <: T_2$:
   We already proved $\Gamma \vdash S \rightleftharpoons S_d$, as required.

d. $\forall n.\ \Gamma \vdash \mathrm{disj}(T_s, U_n)$ and $T_d = T_1 \to T_2$:
Using D-Sub on $\forall n.\ \Gamma \vdash S_n <: T_n$ and $\forall n.\ \Gamma \vdash \mathrm{disj}(T_s, U_n)$ we obtain $\forall n.\ \Gamma \vdash \mathrm{disj}(S_s, U_n)$. From S-Trans we get $\Gamma \vdash S_s <: C_d$. Using S-Match3/4 we obtain $\Gamma \vdash S \rightleftharpoons S_d$. Since $\Gamma \vdash S_d <: T_d$, the IH gives 4 subsubcases:
   (i) S_d is a match type with $\Gamma \vdash S_d \rightleftharpoons S_1 \to S_2$ for some S_1, S_2 such that $\Gamma \vdash T_1 <: S_1$ and $\Gamma \vdash S_2 <: T_2$:
   The result follows directly from S-Trans.

(ii) S_d is a match type with $\Gamma \vdash S_d \rightleftharpoons X$:
The result follows directly from S-TRANS.
(iii) S_d is a type variable:
We already proved $\Gamma \vdash S \rightleftharpoons S_d$, as required.
(iv) S_d has the form $S_1 \to S_2$ with $\Gamma \vdash T_1 <: S_1$ and $\Gamma \vdash S_2 <: T_2$:
We already proved $\Gamma \vdash S \rightleftharpoons S_d$, as required.

- *Case* S-TVAR: $S = Y \quad Y <: T \in \Gamma$
  S is a type variable and the result is immediate.
- *Case* S-ARROW: $S = S_1 \to S_2 \quad T = T_1 \to T_2$
  $\Gamma \vdash T_1 <: S_1 \quad \Gamma \vdash S_2 <: T_2$
  S has the form $S_1 \to S_2$, with $\Gamma \vdash T_1 <: S_1$ and $\Gamma \vdash S_2 <: T_2$, as required.
- *Case* S-TOP, S-SIN, S-ALL, S-PSI:
  In those cases, T is neither a type variable nor a function type and the result is immediate.

4. By induction on a derivation of $\Gamma \vdash S <: T$.
   - *Case* S-REFL: $T = S$
     If T is a match type with $\Gamma \vdash T \rightleftharpoons \forall X <: U_1.\ T_2$ the result follows directly from S-REFL. If $T = \forall X <: U_1.\ T_2$, $S_2 = T_1$, and the result also follows from S-REFL.
   - *Case* S-TRANS: $\Gamma \vdash S <: U \quad \Gamma \vdash U <: T$
     Using the IH on the right premise we get 4 subcases:
     a. U is a match type with $\Gamma \vdash U \rightleftharpoons U_1 \to U_2$ for some U_2 such that $\Gamma, X <: U_1 \vdash U_2 <: T_2$:
        The result follows directly from using the IH on the left premise ($\Gamma, X <: U_1 \vdash S_2 <: T_2$ is obtained using S-TRANS with $\Gamma, X <: U_1 \vdash S_2 <: U_2$ and $\Gamma, X <: U_1 \vdash U_2 <: T_2$).
     b. U is a match type with $\Gamma \vdash U \rightleftharpoons X$:
        Using the 2nd part of the lemma on the left premise leads to the desired result.
     c. U is a type variable:
        The result follows from using the 2nd part of the lemma on the left premise.
     d. U has the form $\forall X <: U_1.\ U_2$ with $\Gamma, X <: U_1 \vdash U_2 <: T_2$:
        The result follows directly from using the IH on the left premise ($\Gamma, X <: U_1 \vdash S_2 <: T_2$ is obtained using S-TRANS with $\Gamma, X <: U_1 \vdash S_2 <: U_2$ and $\Gamma, X <: U_1 \vdash U_2 <: T_2$).
   - *Case* S-MATCH1: $S = T_s\ \mathit{match}\{U_i \Rightarrow T_i\}\ \mathit{or}\ T_d \quad T = T_n$
     $\Gamma \vdash T_s <: U_n \quad \forall m < n.\ \Gamma \vdash \mathrm{disj}(T_s, U_m)$
     If $T = \forall X <: U_1.\ T_2$, we use S-MATCH2 to obtain $\Gamma \vdash S \rightleftharpoons \forall X <: U_1.\ T_2$, as required, and the result follows from S-REFL. If T is a match type with $\Gamma \vdash T \rightleftharpoons \forall X <: U_1.\ T_2$, we use S-MATCH2 to get $\Gamma \vdash S \rightleftharpoons T$, and S-TRANS to obtain $\Gamma \vdash S \rightleftharpoons \forall X <: U_1.\ T_2$, and the result follows from S-REFL.
   - *Case* S-MATCH2: $S = T_n \quad T = T_s\ \mathit{match}\{U_i \Rightarrow T_i\}\ \mathit{or}\ T_d$
     $\Gamma \vdash T_s <: U_n \quad \forall m < n.\ \Gamma \vdash \mathrm{disj}(T_s, U_m)$
     In this case the first premise of the statement ($\Gamma \vdash S <: (\forall X <: U_1.\ T_2)$) does not apply since T is a match type. Using the 1st part of the lemma on $\Gamma \vdash T \rightleftharpoons \forall X <: U_1.\ T_2$, we get 4 subcases:
     a. $\Gamma \vdash T_s <: U_n$, $\forall m < n.\ \Gamma \vdash \mathrm{disj}(T_s, U_m)$ and T_n is a match type with $\Gamma \vdash T_n \rightleftharpoons \forall X <: U_1.\ T_2$:
        $S = T_n$ is a match type and $\Gamma \vdash S \rightleftharpoons \forall X <: U_1.\ T_2$, and the result follows from S-REFL.
     b. $\Gamma \vdash T_s <: U_n$, $\forall m < n.\ \Gamma \vdash \mathrm{disj}(T_s, U_m)$ and $T_n = \forall X <: U_1.\ T_2$:
        $S = \forall X <: U_1.\ T_2$, $S_2 = T_2$, and the result follows from S-REFL.
     c. $\forall m.\ \Gamma \vdash \mathrm{disj}(T_s, U_m)$ and T_d is a match type with $\Gamma \vdash T_d \rightleftharpoons \forall X <: U_1.\ T_2$:

This case cannot occur since $\Gamma \vdash \mathrm{disj}(\mathrm{T}_s, \mathrm{U}_n)$ contradicts $\Gamma \vdash \mathrm{T}_s <: \mathrm{U}_n$ (Lemma 3.5).

d. $\forall m.\ \Gamma \vdash \mathrm{disj}(\mathrm{T}_s, \mathrm{U}_m)$ and $\mathrm{T}_d = \forall \mathrm{X} <: \mathrm{U}_1.\ \mathrm{T}_2$:
This case cannot occur since $\Gamma \vdash \mathrm{disj}(\mathrm{T}_s, \mathrm{U}_n)$ contradicts $\Gamma \vdash \mathrm{T}_s <: \mathrm{U}_n$ (Lemma 3.5).

– *Case* S-Match3: $\mathrm{S} = \mathrm{T}_s\ \mathit{match}\{\mathrm{U}_i \Rightarrow \mathrm{T}_i\}\ \mathit{or}\ \mathrm{T}_d \quad \mathrm{T} = \mathrm{T}_d \quad \forall n.\ \Gamma \vdash \mathrm{disj}(\mathrm{T}_s, \mathrm{U}_n)$
If $\mathrm{T} = \forall \mathrm{X} <: \mathrm{U}_1.\ \mathrm{T}_2$, we use S-Match2 to obtain $\Gamma \vdash \mathrm{S} \rightleftharpoons \forall \mathrm{X} <: \mathrm{U}_1.\ \mathrm{T}_2$, and the result follows from S-Refl. If T is a match type with $\Gamma \vdash \mathrm{T} \rightleftharpoons \forall \mathrm{X} <: \mathrm{U}_1.\ \mathrm{T}_2$, we use S-Match2 to get $\Gamma \vdash \mathrm{S} \rightleftharpoons \mathrm{T}$, and S-Trans to obtain $\Gamma \vdash \mathrm{S} \rightleftharpoons \forall \mathrm{X} <: \mathrm{U}_1.\ \mathrm{T}_2$, and the result follows from S-Refl.

– *Case* S-Match4: $\mathrm{S} = \mathrm{T}_d \quad \mathrm{T} = \mathrm{T}_s\ \mathit{match}\{\mathrm{U}_i \Rightarrow \mathrm{T}_i\}\ \mathit{or}\ \mathrm{T}_d \quad \forall n.\ \Gamma \vdash \mathrm{disj}(\mathrm{T}_s, \mathrm{U}_n)$
In this case the first premise of the statement ($\Gamma \vdash \mathrm{S} <: (\forall \mathrm{X} <: \mathrm{U}_1.\ \mathrm{T}_2)$) does not apply since T is a match type. Using the 1st part of the lemma on $\Gamma \vdash \mathrm{T} \rightleftharpoons \forall \mathrm{X} <: \mathrm{U}_1.\ \mathrm{T}_2$, we get 4 subcases:

a. $\Gamma \vdash \mathrm{T}_s <: \mathrm{U}_n$, $\forall m < n.\ \Gamma \vdash \mathrm{disj}(\mathrm{T}_s, \mathrm{U}_m)$ and T_n is a match type with $\Gamma \vdash \mathrm{T}_n \rightleftharpoons \forall \mathrm{X} <: \mathrm{U}_1.\ \mathrm{T}_2$:
This case cannot occur since $\Gamma \vdash \mathrm{disj}(\mathrm{T}_s, \mathrm{U}_n)$ contradicts $\Gamma \vdash \mathrm{T}_s <: \mathrm{U}_n$ (Lemma 3.5).

b. $\Gamma \vdash \mathrm{T}_s <: \mathrm{U}_n$, $\forall m < n.\ \Gamma \vdash \mathrm{disj}(\mathrm{T}_s, \mathrm{U}_m)$ and $\mathrm{T}_n = \forall \mathrm{X} <: \mathrm{U}_1.\ \mathrm{T}_2$:
This case cannot occur since $\Gamma \vdash \mathrm{disj}(\mathrm{T}_s, \mathrm{U}_n)$ contradicts $\Gamma \vdash \mathrm{T}_s <: \mathrm{U}_n$ (Lemma 3.5).

c. $\forall m.\ \Gamma \vdash \mathrm{disj}(\mathrm{T}_s, \mathrm{U}_m)$ and T_d is a match type with $\Gamma \vdash \mathrm{T}_d \rightleftharpoons \forall \mathrm{X} <: \mathrm{U}_1.\ \mathrm{T}_2$:
$\mathrm{S} = \mathrm{T}_d$ is a match type and $\Gamma \vdash \mathrm{S} \rightleftharpoons \forall \mathrm{X} <: \mathrm{U}_1.\ \mathrm{T}_2$, and the result follows from S-Refl.

d. $\forall m.\ \Gamma \vdash \mathrm{disj}(\mathrm{T}_s, \mathrm{U}_m)$ and $\mathrm{T}_d = \forall \mathrm{X} <: \mathrm{U}_1.\ \mathrm{T}_2$:
$\mathrm{S} = \forall \mathrm{X} <: \mathrm{U}_1.\ \mathrm{T}_2$, $\mathrm{S}_2 = \mathrm{T}_2$, and the result follows from S-Refl.

– *Case* S-Match5: $\mathrm{S} = \mathrm{S}_s\ \mathit{match}\{\mathrm{U}_i \Rightarrow \mathrm{S}_i\}\ \mathit{or}\ \mathrm{S}_d \quad \mathrm{T} = \mathrm{T}_s\ \mathit{match}\{\mathrm{U}_i \Rightarrow \mathrm{T}_i\}\ \mathit{or}\ \mathrm{T}_d \quad \Gamma \vdash \mathrm{S}_s <: \mathrm{T}_s \quad \forall n.\ \Gamma \vdash \mathrm{S}_n <: \mathrm{T}_n \quad \Gamma \vdash \mathrm{S}_d <: \mathrm{T}_d$
In this case the first premise of the statement ($\Gamma \vdash \mathrm{S} <: (\forall \mathrm{X} <: \mathrm{U}_1.\ \mathrm{T}_2)$) does not apply since T is a match type. Using the 1st part of the lemma on $\Gamma \vdash \mathrm{T} \rightleftharpoons \forall \mathrm{X} <: \mathrm{U}_1.\ \mathrm{T}_2$, we get 4 subcases:

a. $\Gamma \vdash \mathrm{T}_s <: \mathrm{U}_n$, $\forall m < n.\ \Gamma \vdash \mathrm{disj}(\mathrm{T}_s, \mathrm{U}_m)$ and T_n is a match type with $\Gamma \vdash \mathrm{T}_n \rightleftharpoons \forall \mathrm{X} <: \mathrm{U}_1.\ \mathrm{T}_2$:
Using D-Sub on $\forall m.\ \Gamma \vdash \mathrm{S}_m <: \mathrm{T}_m$ and $\forall m < n.\ \Gamma \vdash \mathrm{disj}(\mathrm{T}_s, \mathrm{U}_m)$ we obtain $\forall m < n.\ \Gamma \vdash \mathrm{disj}(\mathrm{S}_s, \mathrm{U}_m)$. From S-Trans we get $\Gamma \vdash \mathrm{S}_s <: \mathrm{U}_n$. Using S-Match1/2 we obtain $\Gamma \vdash \mathrm{S} \rightleftharpoons \mathrm{S}_n$. Since $\Gamma \vdash \mathrm{S}_n <: \mathrm{T}_n$ and $\Gamma \vdash \mathrm{T}_n \rightleftharpoons \forall \mathrm{X} <: \mathrm{U}_1.\ \mathrm{T}_2$, the IH gives 4 subsubcases:

(i) S_n is a match type with $\Gamma \vdash \mathrm{S}_n \rightleftharpoons \forall \mathrm{X} <: \mathrm{U}_1.\ \mathrm{S}_2$ for some S_2 such that $\Gamma, \mathrm{X} <: \mathrm{U}_1 \vdash \mathrm{S}_2 <: \mathrm{T}_2$:
The result follows directly from S-Trans.

(ii) S_n is a match type with $\Gamma \vdash \mathrm{S}_n \rightleftharpoons \mathrm{X}$:
The result follows directly from S-Trans.

(iii) S_n is a type variable:
We already proved $\Gamma \vdash \mathrm{S} \rightleftharpoons \mathrm{S}_n$, as required.

(iv) S_n has the form $\forall \mathrm{X} <: \mathrm{U}_1.\ \mathrm{S}_2$ with $\Gamma, \mathrm{X} <: \mathrm{U}_1 \vdash \mathrm{S}_2 <: \mathrm{T}_2$:
We already proved $\Gamma \vdash \mathrm{S} \rightleftharpoons \mathrm{S}_n$, as required.

b. $\Gamma \vdash \mathrm{T}_s <: \mathrm{U}_n$, $\forall m < n.\ \Gamma \vdash \mathrm{disj}(\mathrm{T}_s, \mathrm{U}_m)$ and $\mathrm{T}_n = \forall \mathrm{X} <: \mathrm{U}_1.\ \mathrm{T}_2$:
Using D-Sub on $\forall m.\ \Gamma \vdash \mathrm{S}_m <: \mathrm{T}_m$ and $\forall m < n.\ \Gamma \vdash \mathrm{disj}(\mathrm{T}_s, \mathrm{U}_m)$ we obtain $\forall m < n.\ \Gamma \vdash \mathrm{disj}(\mathrm{S}_s, \mathrm{U}_m)$. From S-Trans we get $\Gamma \vdash \mathrm{S}_s <: \mathrm{U}_n$. Using S-Match1/2 we obtain $\Gamma \vdash \mathrm{S} \rightleftharpoons \mathrm{S}_n$. Since $\Gamma \vdash \mathrm{S}_n <: \mathrm{T}_n$, the IH, the IH gives 4 subsubcases:

(i) S_n is a match type with $\Gamma \vdash S_n \rightleftharpoons \forall X <: U_1.\ S_2$ for some S_2 such that $\Gamma, X <: U_1 \vdash S_2 <: T_2$:
The result follows directly from S-Trans.
(ii) S_n is a match type with $\Gamma \vdash S_n \rightleftharpoons X$:
The result follows directly from S-Trans.
(iii) S_n is a type variable:
We already proved $\Gamma \vdash S \rightleftharpoons S_n$, as required.
(iv) S_n has the form $\forall X <: U_1.\ S_2$ with $\Gamma, X <: U_1 \vdash S_2 <: T_2$:
We already proved $\Gamma \vdash S \rightleftharpoons S_n$, as required.

c. $\forall m.\ \Gamma \vdash \text{disj}(T_s, U_m)$ and T_d is a match type with $\Gamma \vdash T_d \rightleftharpoons \forall X <: U_1.\ T_2$:
Using D-Sub on $\forall m.\ \Gamma \vdash S_m <: T_m$ and $\forall m.\ \Gamma \vdash \text{disj}(T_s, U_m)$ we obtain $\forall m.\ \Gamma \vdash \text{disj}(S_s, U_m)$. From S-Trans we get $\Gamma \vdash S_s <: C_d$. Using S-Match3/4 we obtain $\Gamma \vdash S \rightleftharpoons S_d$. Since $\Gamma \vdash S_d <: T_d$ and $\Gamma \vdash T_d \rightleftharpoons \forall X <: U_1.\ T_2$, the IH gives 4 subsubcases:
(i) S_d is a match type with $\Gamma \vdash S_d \rightleftharpoons \forall X <: U_1.\ S_2$ for some S_2 such that $\Gamma, X <: U_1 \vdash S_2 <: T_2$::
The result follows directly from S-Trans.
(ii) S_d is a match type with $\Gamma \vdash S_d \rightleftharpoons X$:
The result follows directly from S-Trans.
(iii) S_d is a type variable:
We already proved $\Gamma \vdash S \rightleftharpoons S_d$, as required.
(iv) S_d has the form $\forall X <: U_1.\ S_2$ with $\Gamma, X <: U_1 \vdash S_2 <: T_2$:
We already proved $\Gamma \vdash S \rightleftharpoons S_d$, as required.

d. $\forall m.\ \Gamma \vdash \text{disj}(T_s, U_m)$ and $T_d = \forall X <: U_1.\ T_2$:
Using D-Sub on $\forall m.\ \Gamma \vdash S_m <: T_m$ and $\forall m.\ \Gamma \vdash \text{disj}(T_s, U_m)$ we obtain $\forall m.\ \Gamma \vdash \text{disj}(S_s, U_m)$. From S-Trans we get $\Gamma \vdash S_s <: C_d$. Using S-Match3/4 we obtain $\Gamma \vdash S \rightleftharpoons S_d$. Since $\Gamma \vdash S_d <: T_d$, the IH gives 4 subsubcases:
(i) S_d is a match type with $\Gamma \vdash S_d \rightleftharpoons \forall X <: U_1.\ S_2$ for some S_2 such that $\Gamma, X <: U_1 \vdash S_2 <: T_2$::
The result follows directly from S-Trans.
(ii) S_d is a match type with $\Gamma \vdash S_d \rightleftharpoons X$:
The result follows directly from S-Trans.
(iii) S_d is a type variable:
We already proved $\Gamma \vdash S \rightleftharpoons S_d$, as required.
(iv) S_d has the form $\forall X <: U_1.\ S_2$ with $\Gamma, X <: U_1 \vdash S_2 <: T_2$:
We already proved $\Gamma \vdash S \rightleftharpoons S_d$, as required.

– *Case* S-TVar: $S = Y \quad Y <: T \in \Gamma$
S is a type variable and the result is immediate.
– *Case* S-All: $S = \forall X <: U_1.\ S_2 \quad T = \forall X <: U_1.\ T_2 \quad \Gamma, X <: U_1 \vdash S_2 <: T_2$
S has the form $\forall X <: U_1.\ S_2$, with $\Gamma, X <: U_1 \vdash S_2 <: T_2$, as required.
– *Case* S-Top, S-Sin, S-Arrow, S-Psi:
In those cases, T is neither a type variable nor a universal type and the result is immediate.

□

Lemma 3.7 (Canonical forms).

1. *If* $\Gamma \vdash t : T$, *where either* T *is a type variable, or* T *is a match type with* $\Gamma \vdash T \rightleftharpoons X$, *then* t *is not a closed value.*

2. *If* v *is a closed value with* $\Gamma \vdash v : T$ *where either* $T = T_1 \to T_2$*, or* T *is a match type and* $\Gamma \vdash T \rightleftharpoons T_1 \to T_2$*, then* v *has the form* $\lambda x : S_1.\ t_2$.
3. *If* v *is a closed value with* $\Gamma \vdash v : T$ *where either* $T = \forall X <: U_1.\ T_2$*, or* T *is a match type and* $\Gamma \vdash T \rightleftharpoons \forall X <: U_1.\ T_2$*, then* v *has the form* $\lambda X <: U_1.\ t_2$.

Proof:

1. By induction on a derivation of $\Gamma \vdash t : T$
   - *Case* T-Var, T-TApp, T-App, T-Match:
     In those cases, t is not a value and the result is immediate.
   - *Case* T-Abs, T-TAbs, T-Class:
     In those cases, T is neither a match type nor a type variable and the result is immediate.
   - *Case* T-Sub: $\Gamma \vdash t : S \quad \Gamma \vdash S <: T$
     By Lemma 3.6, either S is a match type with $\Gamma \vdash S \rightleftharpoons Y$, or S is a type variable. In both cases, we use the IH to show that t is not a closed value, as required.
2. By induction on a derivation of $\Gamma \vdash t : T$.
   - *Case* T-Var, T-TApp, T-App, T-Match:
     In those cases, t is not a value and the result is immediate.
   - *Case* T-Abs: $t = \lambda x : T_1 t_2 \quad T = T_1 \to T_2 \quad \Gamma, x : T_1 \vdash t_2 : T_2$
     $t = \lambda x : T_1 t_2$ and the result is immediate.
   - *Case* T-TAbs, T-Class:
     T is neither a function type nor a match type and the result is immediate.
   - *Case* T-Sub: $\Gamma \vdash t : S \quad \Gamma \vdash S <: T$
     Using Lemma 3.6 we get 4 subcases:
     a. S is a match type with $\Gamma \vdash S \rightleftharpoons S_1 \to S_2$:
        The result follows from the IH.
     b. S is a match type with $\Gamma \vdash S \rightleftharpoons X$:
        This case cannot occure since the 1th part of the lemma would lead to a contradiction on the fact that v is a closed value.
     c. S is a type variable:
        Similarly, this case cannot occure since the 1st part of the lemma would lead to a contradiction.
     d. S has the form $S_1 \to S_2$ with $\Gamma \vdash T_1 <: S_1$ and $\Gamma \vdash S_2 <: T_2$:
        The result follows from the IH.
3. By induction on a derivation of $\Gamma \vdash t : T$.
   - *Case* T-Var, T-TApp, T-App, T-Match:
     In those cases, t is not a value and the result is immediate.
   - *Case* T-Abs, T-Class:
     T is neither a universal type nor a match type and the result is immediate.
   - *Case* T-TAbs: $t = \lambda Y <: T_1.\ t_2 \quad T = \forall Y <: T_1.\ T_2 \quad \Gamma, Y <: T_1 \vdash t_2 : T_2$
     $t = \lambda Y <: T_1.\ t_2$ and the result is immediate.
   - *Case* T-Sub: $\Gamma \vdash t : S \quad \Gamma \vdash S <: T$
     Using Lemma 3.6 we get 4 subcases:
     a. S is a match type with $\Gamma \vdash S \rightleftharpoons \forall X <: U_1.\ S_2$, The result follows from the IH.
     b. S is a match type with $\Gamma \vdash S \rightleftharpoons X$, This case cannot occure since the 1th part of the lemma would lead to a contradiction on the fact that v is a closed value.
     c. S is a type variable, Similarly, this case cannot occure since the 1st part of the lemma would lead to a contradiction.

d. S has the form $\forall X <: U_1.\ S_2$ and $\Gamma, X <: U_1 \vdash S_2 <: T_2$. The result follows from the IH.

□

Lemma 3.8 (Inversion of typing).

1. *If* $\Gamma \vdash \lambda x : S_1.\ s_2 : T$ *and* $\Gamma \vdash T <: U_1 \to U_2$, *then* $\Gamma \vdash U_1 <: S_1$ *and there is some* S_2 *such that* $\Gamma, x : S_1 \vdash s_2 : S_2$ *and* $\Gamma \vdash S_2 <: U_2$.
2. *If* $\Gamma \vdash \lambda X <: S_1.\ s_2 : T$ *and* $\Gamma \vdash T <: (\forall X <: U_1.\ U_2)$, *then* $U_1 = S_1$ *and there is some* S_2 *such that* $\Gamma, X <: S_1 \vdash s_2 : S_2$ *and* $\Gamma, X <: S_1 \vdash S_2 <: U_2$.

Proof:

1. By induction on a derivation of $\Gamma \vdash t : T$.
   - *Case* T-Abs: $t = \lambda x : S_1 s_2 \quad T = S_1 \to T_2 \quad \Gamma\, x : S_1 \vdash s_2 : T_2$
     Given $\Gamma \vdash S_1 \to T_2 <: U_1 \to U_2$, we use Lemma 3.6 to obtain $\Gamma \vdash U_1 <: S_1$ and $\Gamma \vdash T_2 <: U_2$. We pick S_2 to be T_2 to obtain the desired result.
   - *Case* T-Sub: $\Gamma \vdash t : S \quad \Gamma \vdash S <: T$
     Using S-Trans with $\Gamma \vdash S <: T$ and $\Gamma \vdash T <: U_1 \to U_2$ we get $\Gamma \vdash S <: U_1 \to U_2$. The results follows directly from the IH.
   - *Case* T-Var, T-App, T-TAbs, T-TApp, T-Class, T-Match:
     In those cases, t is not of the form $\lambda x : S_1.\ s_2$ and the result is immediate.
2. By induction on a derivation of $\Gamma \vdash t : T$.
   - *Case* T-TAbs: $t = \lambda X <: S_1.\ s_2 \quad T = \forall X <: S_1.\ T_2 \quad \Gamma, X <: S_1 \vdash s_2 : T_2$
     Given $\Gamma \vdash (\forall X <: S_1.\ T_2) <: (\forall X <: U_1.\ U_2)$, we use Lemma 3.6 to obtain $U_1 = S_1$ and $\Gamma, X <: S_1 \vdash T_2 <: U_2$. We pick S_2 to be T_2 to obtain the desired result.
   - *Case* T-Sub: $\Gamma \vdash t : S \quad \Gamma \vdash S <: T$
     Using S-Trans with $\Gamma \vdash S <: T$ and $\Gamma \vdash T <: (\forall X <: U_1.\ U_2)$ we get $\Gamma \vdash S <: (\forall X <: U_1.\ U_2)$. The results follows directly from the IH.
   - *Case* T-Var, T-Abs, T-App, T-TApp, T-Class, T-Match:
     In those cases, t is not of the form $\lambda X <: S_1.\ s_2 : T$ and the result is immediate.

□

Lemma 3.9 (Minimum types).

1. *If* $\Gamma \vdash \mathit{new}\, C : T$ *then* $\Gamma \vdash \{\mathit{new}\, C\} <: T$.
2. *If* $\Gamma \vdash \lambda x : T_1.\ t_2 : T$ *then there is some* T_2 *such that* $\Gamma \vdash T_1 \to T_2 <: T$.
3. *If* $\Gamma \vdash \lambda X <: U_1.\ t_2 : T$ *then there is some* T_2 *such that* $\Gamma \vdash \forall X <: U_1.\ T_2 <: T$.

Proof:

1. By induction on a derivation of $\Gamma \vdash t : T$.
   - *Case* T-Var, T-Abs, T-App, T-TAbs, T-TApp, T-Match:
     In those cases, t is not a constructor call and the result is immediate.
   - *Case* T-Class: $t = \mathit{new}\, C \quad T = \{\mathit{new}\, C\}$
     The result follows directly from S-Refl.
   - *Case* T-Sub: $\Gamma \vdash t : S \quad \Gamma \vdash S <: T$
     By the IH, $\Gamma \vdash \{\mathit{new}\, C\} <: S$. The result follows from S-Trans.
2. By induction on a derivation of $\Gamma \vdash t : T$.
   - *Case* T-Var, T-App, T-TAbs, T-TApp, T-Class, T-Match:
     In those cases, t is not an abstraction and the result is immediate
   - *Case* T-Abs: $T = T_1 \to S_2 \quad \Gamma, x : T_1 \vdash t_2 : S_2$
     $T_2 = S_2$ and the result is immediate using S-Refl.

- *Case* T-Sub: $\Gamma \vdash t : S \quad \Gamma \vdash S <: T$
  Using the IH, there is some T_2 such that $\Gamma \vdash T_1 \to T_2 <: S$. The result follows from S-Trans.

3. By induction on a derivation of $\Gamma \vdash t : T$.
   - *Case* T-Var, T-Abs, T-App, T-TApp, T-Class, T-Match:
     In those cases, t is not a type abstraction and the result is immediate
   - *Case* T-TAbs: $T = \forall X <: U_1 . S_2 \quad \Gamma, X <: U_1 \vdash t_2 : S_2$
     $T_2 = S_2$ and the result is immediate using S-Refl.
   - *Case* T-Sub: $\Gamma \vdash t : S \quad \Gamma \vdash S <: T$
     Using the IH, there is some T_2 such that $\Gamma \vdash \forall X <: U_1 . T_2 <: S$. The result follows from S-Trans.

□

Theorem 3.10 (Preservation).
If $\Gamma \vdash t : T$ *and* $t \longrightarrow t'$ *then* $\Gamma \vdash t' : T$.

Proof: By induction on a derivation of $\Gamma \vdash t : T$.

- *Case* T-Var, T-Abs, T-TAbs, T-Class:
  These cases cannot arise since we assume $t \longrightarrow t'$ but there are no evaluation rules for variables, abstractions, type abstractions and class instantiations.
- *Case* T-App: $t = t_1 t_2 \quad T = T_{12} \quad \Gamma \vdash t_1 : T_{11} \to T_{12} \quad \Gamma \vdash t_2 : T_{11}$
  By definition of the evaluation relation, there are 3 subcases:
  Subcase E-App1: $t_1 \longrightarrow t_1' \quad t' = t_1' t_2$
  By the IH and the 1st premise we get $\Gamma \vdash t_1' : T_{11} \to T_{12}$. We use T-App with that result and the 2nd premise to obtain $\Gamma \vdash t_1' t_2 : T_{12}$, as required.
  Subcase E-App2: $t_2 \longrightarrow t_2' \quad t_1 = v_1 \quad t' = v_1 t_2'$
  Analogously, the IH gives $\Gamma \vdash t_2' : T_{12}$ and the result follows from T-App.
  Subcase E-AppAbs: $t_1 = \lambda x : U_{11} . u_{12} \quad t' = [x \mapsto t_2]u_{12}$
  By Lemma 3.8 (with $S_1 = U_{11}$, $s_2 = u_{12}$, $U_1 = T_{11}$ and $U_2 = T_{12}$), there is some S_2 such that $\Gamma, x : U_{11} \vdash u_{12} : S_2$, $\Gamma \vdash S_2 <: T_{12}$ and $\Gamma \vdash T_{11} <: U_{11}$. Using T-Sub with $\Gamma \vdash t_2 : T_{11}$ and $\Gamma \vdash T_{11} <: U_{11}$ we obtain $\Gamma \vdash t_2 : U_{11}$. By Lemma 3.4 we get $\Gamma \vdash [x \mapsto t_2]u_{12} : S_2$. Using T-Sub we obtain $\Gamma \vdash [x \mapsto t_2]u_{12} : T_{12}$, as required.
- *Case* T-TApp: $t = t_1 T_2 \quad T = [X \mapsto T_2]T_{12}$
  $\Gamma \vdash t_1 : (\forall X <: U_1 . T_{12}) \quad \Gamma \vdash T_2 <: U_1$
  The proof for case T-TApp is analogous to the one for T-App. By definition of the evaluation relation, there are 2 subcases:
  Subcase E-TApp: $t_1 \longrightarrow t_1' \quad t' = t_1' T_2$
  By the IH and the 1st premise we get $\Gamma \vdash t_1' : (\forall X <: U_1 . T_{12})$. We use T-TApp with that result and the 2nd premise to obtain $\Gamma \vdash t_1' T_2 : [X \mapsto T_2]T_{12}$, as required.
  Subcase E-TAppTAbs: $t_1 = \lambda X <: U_{11} . u_{12} \quad t' = [X \mapsto T_2]u_{12}$
  By Lemma 3.8 (with $S_1 = U_{11}$, $s_2 = u_{12}$ and $U_2 = T_{12}$), there is some S_2 such that $\Gamma, X <: U_{11} \vdash u_{12} : S_2$, $U_1 = U_{11}$, and $\Gamma, X <: U_{11} \vdash S_2 <: T_{12}$. Since $\Gamma \vdash T_2 <: U_{11}$, we use Lemma 3.4 twice to get $\Gamma \vdash [X \mapsto T_2]u_{12} : [X \mapsto T_2]S_2$ and $\Gamma \vdash [X \mapsto T_2]S_2 <: [X \mapsto T_2]T_{12}$. By T-Sub $\Gamma \vdash [X \mapsto T_2]u_{12} : [X \mapsto T_2]T_{12}$, as required.
- *Case* T-Sub: $\Gamma \vdash t : S \quad \Gamma \vdash S <: T$
  By the IH, $\Gamma \vdash t' : S$. The result follows directly from T-Sub.
- *Case* T-Match: $t = t_s \ match \{ C_i \Rightarrow t_i \} or\ t_d \quad T = T_s \ match \{ C_i \Rightarrow T_i \} or\ T_d$
  $\Gamma \vdash t_s : T_s \quad \Gamma, x_i : C_i \vdash t_i : T_i \quad \Gamma \vdash t_d : T_d$
  By definition of the evaluation relation, there are 5 subcases:

Subcase E-Match1: $t_s \longrightarrow t'_s \quad t' = t'_s\ \mathit{match}\{x_i : C_i \Rightarrow t_i\}\mathit{or}\ t_d$
By the IH $\Gamma \vdash t'_s : T_s$. The result follows directly from T-Match.
Subcase E-Match2: $t_s = \mathit{new}\, C \quad (C, C_n) \in \Psi$
$\forall m < n.\ (C, C_m) \notin \Psi \quad t' = [x_n \mapsto \mathit{new}\, C]t_n$
By S-Psi, S-Sin and S-Trans, $\Gamma \vdash \{\mathit{new}\, C\} <: C_n$. From D-Psi, $\forall m < n.\ \Gamma \vdash \mathrm{disj}(\{\mathit{new}\, C\}, C_m)$. By Lemma 3.9 we get $\Gamma \vdash \{\mathit{new}\, C\} <: T_s$. Let T_1 be $\{\mathit{new}\, C\}\ \mathit{match}\{C_i \Rightarrow T_i\}\mathit{or}\ T_d$. From S-Match1, $\Gamma \vdash T_n <: T_1$. Using S-Match5, $\Gamma \vdash T_1 <: T$. By S-Trans, $\Gamma \vdash T_n <: T$. Using T-Class, T-Sub, S-Sin and S-Psi (with $(C, C_n) \in \Psi$) we get $\Gamma \vdash \mathit{new}\, C : C_n$. From the case premises, we have $\Gamma, x_n : C_n \vdash t_n : T_n$. By Lemma 3.4, we get $\Gamma \vdash [x_n \mapsto \mathit{new}\, C]t_n : T_n$. Finally, using T-Sub we get , $\Gamma \vdash [x_n \mapsto \mathit{new}\, C]t_n : T$, as required.
Subcase E-Match3: $t_s = \mathit{new}\, C \quad \forall n.\ (C, C_n) \notin \Psi \quad t' = t_d$
The proof for subcase E-Match3 is analogous to the one for E-Match2. From D-Psi, $\forall n.\ \Gamma \vdash \mathrm{disj}(\{\mathit{new}\ C\}, C_n)$. By Lemma 3.9 we get $\Gamma \vdash \{\mathit{new}\ C\} <: T_s$. Let T_1 be $\{\mathit{new}\, C\}\ \mathit{match}\{C_i \Rightarrow T_i\}\mathit{or}\, T_d$. From S-Match2, $\Gamma \vdash T_d <: T_1$. Using S-Match5, $\Gamma \vdash T_1 <: T$. By S-Trans $\Gamma \vdash T_d <: T$. Using T-Sub, $\Gamma \vdash t_d : T$, as required.
Subcase E-Match4: $t_s = \lambda x : U.\ u \quad t' = t_d$
By Lemma 3.9, there exists a V such that $\Gamma \vdash U \to V <: T_s$. Using D-Arrow, $\forall n.\ \Gamma \vdash \mathrm{disj}(U \to V, C_n)$. Let T_1 be $U \to V\ \mathit{match}\{C_i \Rightarrow T_i\}\mathit{or}\, T_d$. From S-Match2, $\Gamma \vdash T_d <: T_1$. Using S-Match5, $\Gamma \vdash T_1 <: T$. By S-Trans $\Gamma \vdash T_d <: T$. Using T-Sub, $\Gamma \vdash t_d : T$, as required.
Subcase E-Match5: $t_s = \forall X <: U.\ u \quad t' = t_d$
The proof for subcase E-Match5 is analogous to the one for E-Match4. By Lemma 3.9, there exists a V such that $\Gamma \vdash (\forall X <: U.\ V) <: T_s$. Using D-Arrow, we get $\forall n.\ \Gamma \vdash \mathrm{disj}(\forall X <: U.\ V, C_n)$. Let T_1 be $(\forall X <: U.\ V)\ \mathit{match}\{C_i \Rightarrow T_i\}\mathit{or}\, T_d$. From S-Match2, $\Gamma \vdash T_d <: T_1$. Using S-Match5, $\Gamma \vdash T_1 <: T$. By S-Trans $\Gamma \vdash T_d <: T$. Using T-Sub, $\Gamma \vdash t_d : T$, as required.

□

Theorem 3.11 (Progress).
If t *is a closed, well-typed term, then either* t *is a value or there is some* t′ *such that* $t \longrightarrow t'$.

Proof: By induction on a derivation of $\Gamma \vdash t : T$.

- *Case* T-Var: $t = x \quad x : T \in \Gamma$
  This case cannot occure because t is closed.
- *Case* T-Abs, T-TAbs, T-Class:
  In those cases, t is a value and the result is immediate.
- *Case* T-App: $t = t_1 t_2 \quad T = T_{12} \quad \Gamma \vdash t_1 : T_{11} \to T_{12} \quad \Gamma \vdash t_2 : T_{11}$
  By the IH, either t_1 is a value or t_1 can take a step (there is some t'_1 such that $t_1 \longrightarrow t'_1$). If t_1 can take a step, then E-App1 applies to t. If t_1 is a value, we use Lemma 3.7 to get that t_1 has the form $\lambda x : S_1.\ t_2$. Therefore, E-AppAbs applies to t, as required.
- *Case* T-TApp: $t = t_1 T_2 \quad T = [X \mapsto T_2]T_{12}$
  $\Gamma \vdash t_1 : (\forall X <: U_1.\ T_{12}) \quad \Gamma \vdash T_2 <: U_1$
  The proof for case T-TApp is analogous to the one for T-App.
  By the IH, either t_1 is a value or t_1 can take a step. If t_1 can take a step, then E-TApp applies to t. If t_1 is a value, we use Lemma 3.7 to get that t_1 has the form $\lambda X <: T_1.\ t_2$. Therefore, E-TAppTAbs applies to t, as required.
- *Case* T-Sub: $\Gamma \vdash t : S \quad \Gamma \vdash S <: T$
  The result follows directly from the IH.
- *Case* T-Match: $t = t_s\ \mathit{match}\{x_i : C_i \Rightarrow t_i\}\mathit{or}\, t_d \quad T = T_s\ \mathit{match}\{C_i \Rightarrow T_i\}\mathit{or}\, T_d$
  $\Gamma \vdash t_s : T_s \quad \Gamma, x_i : C_i \vdash t_i : T_i \quad \Gamma \vdash t_d : T_d$

By the IH, either t_s is a value or t_s can take a step. If t_s can take a step, then E-Match1 applies to t, as required. If t_s is a value, t_s can take 3 different forms:

Subcase t_s is of the form *new* C:
If $\forall m.\ (C, C_m) \notin \Psi$, then E-Match3 applies to t. Otherwise, let C_k be the first class such that $(C, C_k) \in \Psi$. By construction we know that $\forall m < k.\ (C, C_k) \notin \Psi$. Therefore E-Match2 applies to t, as required.

Subcase t_s is of the form $\lambda x : T_1 t_2$:
E-Match4 applies to t and the result is immediate.

Subcase t_s is of the form $\forall X <: U_1.\ T_2$:
E-Match5 applies to t and the result is immediate.

□

APPENDIX B: A TYPE SAFETY PROOF FOR SYSTEM FMB

LEMMA 3.1 (PERMUTATION). *If* Γ *and* Δ *are well-formed and* Δ *is a permutation of* Γ*, then:*

1. *If* $\Gamma \vdash \mathrm{disj}(S, T)$*, then* $\Delta \vdash \mathrm{disj}(S, T)$.
2. *If* $\Gamma \vdash S <: T$*, then* $\Delta \vdash S <: T$.
3. *If* $\Gamma \vdash t : T$*, then* $\Delta \vdash t : T$.

Proof: We prove 1. and 2. simultaneously by induction on two derivations of $\Gamma \vdash \mathrm{disj}(V, W)$ and $\Gamma \vdash S <: T$. More precisely, the induction is done on the cumulative depth of both derivation tree.

1. $\Gamma \vdash \mathrm{disj}(V, W)$
   - *Case* D-XI: $V = C_1 \quad W = C_2 \quad (C_1, C_2) \in \Xi$
     Using D-XI with context Δ directly leads to the desired result.
   - *Case* D-PSI: $V = \{new\, C_1\} \quad W = C_2 \quad (C_1, C_2) \notin \Psi$
     Using D-PSI with context Δ directly leads to the desired result.
   - *Case* D-SUB: $\Gamma \vdash V <: U \quad \Gamma \vdash \mathrm{disj}(U, W)$
     By the IH we get $\Delta \vdash \mathrm{disj}(U, W)$. Using the 2nd part of the lemma we obtain $\Delta \vdash V <: U$. The result follows from D-SUB.
   - *Case* D-ARROW: $V = V_1 \to V_2 \quad W = C$
     Using D-ARROW with context Δ directly leads to the desired result.
   - *Case* D-ALL: $V = \forall X <: V_1.\, V_2 \quad W = C$
     Using D-ALL with context Δ directly leads to the desired result.
2. $\Gamma \vdash S <: T$.
   - *Case* S-REFL: $T = S$
     Using S-REFL with context Δ directly leads to the desired result.
   - *Case* S-TRANS: $\Gamma \vdash S <: U \quad \Gamma \vdash U <: T$
     The result follows directly from the IH and S-TRANS.
   - *Case* S-TOP: $T = \mathrm{Top}$
     Using S-TOP with context Δ directly leads to the desired result.
   - *Case* S-SIN: $S = \{new\, C\} \quad T = C$
     Using S-SIN with context Δ directly leads to the desired result.
   - *Case* S-TVAR: $S = X \quad X <: T \in \Gamma$ Since Δ is a permutation of Γ, $X <: T \in \Delta$, and the result follows from S-TVAR.
   - *Case* S-ARROW: $S = S_1 \to S_2 \quad T = T_1 \to T_2$
     $\Gamma \vdash T_1 <: S_1 \quad \Gamma \vdash S_2 <: T_2$
     The result follows directly from the IH and S-ARROW.
   - *Case* S-ALL: $S = \forall X <: U_1.\, S_2 \quad T = \forall X <: U_1.\, T_2 \quad \Gamma, X <: U_1 \vdash S_2 <: T_2$
     If Δ is a permutation of Γ, then $\Delta, X <: U_1$ is a permutation of $\Gamma, X <: U_1$. Therefore, we can use the IH to get $\Delta, X <: U_1 \vdash S_2 <: T_2$. The result follows from S-ALL.
   - *Case* S-PSI: $S = C_1 \quad T = C_2 \quad (C_1, C_2) \in \Psi$
     Using S-PSI with context Δ directly leads to the desired result.
   - *Case* BS-MATCH1/2: $T_1 = T_n \quad T_2 = T_s \;\; match[X]\{S_i \Rightarrow T_i\}\, or\, T_d$
     $\Gamma, X <: U \vdash T_s <: S_n$
     $\forall m < n.\ \Gamma, X <: \mathrm{Top} \vdash \mathrm{disj}(T_s, S_m)$
     By the 1st part of the lemma we get $\forall m < n.\ \Delta, X <: \mathrm{Top} \vdash \mathrm{disj}(T_s, S_m)$. From the IH we obtain $\Delta, X <: U \vdash T_s <: S_n$. The result follows from BS-MATCH1/2.
   - *Case* BS-MATCH3/4: $S = T_d \quad T = T_s \;\; match[X]\{S_i \Rightarrow T_i\}\, or\, T_d$
     $\forall n.\ \Gamma, X <: \mathrm{Top} \vdash \mathrm{disj}(T_s, S_n)$
     The result follows from the 1st part of the lemma and BS-MATCH3/4.

- *Case* BS-Match5: $S = S_s\ \mathit{match}[X]\{U_i \Rightarrow S_i\}\mathit{or}\,S_d \quad T = T_s\ \mathit{match}[X]\{U_i \Rightarrow T_i\}\mathit{or}\,T_d$
  $\Gamma \vdash S_s <: T_s$
  $\forall n.\ \Gamma, X <: \mathrm{Top} \vdash S_n <: T_n$
  $\Gamma \vdash S_d <: T_d$
  The result follows directly from the IH and BS-Match5.

3. By induction on a derivation of $\Gamma \vdash t : T$
   - *Case* T-Var: $t = x \quad x : T \in \Gamma$
     Since Δ is a permutation of Γ, $x : T \in \Delta$, and the result follows from T-Var.
   - *Case* T-Abs: $t = \lambda x : T_1 t_2 \quad T = T_1 \to T_2 \quad \Gamma, x : T_1 \vdash t_2 : T_2$
     If Δ is a permutation of Γ, then $\Delta, x : T_1$ is a permutation of $\Gamma, x : T_1$. Therefore, we can use the IH to get $\Delta, x : T_1 \vdash t_2 : T_2$. The result follows from T-Abs.
   - *Case* T-App: $t = t_1 t_2 \quad T = T_{12} \quad \Gamma \vdash t_1 : T_{11} \to T_{12} \quad \Gamma \vdash t_2 : T_{11}$
     The result follows directly from the IH and T-App.
   - *Case* T-TAbs: $t = \lambda X <: U_1.\ t_2 \quad T = \forall X <: U_1.\ T_2 \quad \Gamma, X <: U_1 \vdash t_2 : T_2$
     If Δ is a permutation of Γ, then $\Delta, X <: U_1$ is a permutation of $\Gamma, X <: U_1$. Therefore, we can use the IH to get $\Delta, X <: U_1 \vdash t_2 : T_2$. The result follows from T-TAbs.
   - *Case* T-TApp: $t = t_1 T_2 \quad T = [X \mapsto T_2]T_{12}$
     $\Gamma \vdash t_1 : (\forall X <: U_1.\ T_{12}) \quad \Gamma \vdash T_2 <: U_1$
     By the IH we get $\Delta \vdash t_1 : (\forall X <: U_1.\ T_{12})$. Using the 2nd part of the lemma we obtain $\Delta \vdash T_2 <: U_1$. The result follows from T-TApp.
   - *Case* T-Sub: $\Gamma \vdash t : S \quad \Gamma \vdash S <: T$
     By the IH we get $\Delta \vdash t : S$. Using the 2nd part of the lemma we obtain $\Delta \vdash S <: T$. The result follows from T-Sub.
   - *Case* T-Class: $t = \mathit{new}\,C \quad T = \{\mathit{new}\,C\}$
     Using T-Class with context Δ directly leads to the desired result.
   - *Case* BT-Match: $t = t_s\ \mathit{match}[X]\{x_i : C_i \Rightarrow t_i\}\mathit{or}\,t_d \quad T = T_s\ \mathit{match}[X]\{C_i \Rightarrow T_i\}\mathit{or}\,T_d$
     $\Gamma \vdash t_s : T_s$
     $\Gamma, X <: \mathrm{Top}, x_i : C_i \vdash t_i : T_i$
     $\Gamma \vdash t_d : T_d$
     If Δ is a permutation of Γ, then $\Delta, X <: \mathrm{Top}, x_i : C_i$ is a permutation of $\Gamma, X <: \mathrm{Top}, x_i : C_i$. Therefore the result follows directly from the IH and BT-Match.

□

Lemma 3.2 (Weakening).

1. *If* $\Gamma \vdash \mathrm{disj}(S, T)$ *and* $\Gamma, X <: U$ *is well formed,*
   then $\Gamma, X <: U \vdash \mathrm{disj}(S, T)$.
2. *If* $\Gamma \vdash S <: T$ *and* $\Gamma, X <: U$ *is well formed,*
   then $\Gamma, X <: U \vdash S <: T$.
3. *If* $\Gamma \vdash S <: T$ *and* $\Gamma, x : U$ *is well formed,*
   then $\Gamma, x : U \vdash S <: T$.
4. *If* $\Gamma \vdash t : T$ *and* $\Gamma, x : U$ *is well formed,*
   then $\Gamma, x : U \vdash t : T$.
5. *If* $\Gamma \vdash t : T$ *and* $\Gamma, X <: U$ *is well formed,*
   then $\Gamma, X <: U \vdash t : T$.

Proof: We prove 1. and 2. simultaneously by induction on two derivations of $\Gamma \vdash \mathrm{disj}(V, W)$ and $\Gamma \vdash S <: T$. More precisely, the induction is done on the cumulative depth of both derivation tree.

1. $\Gamma \vdash \mathrm{disj}(V, W)$

- *Case* D-Xi: $V = C_1 \quad W = C_2 \quad (C_1, C_2) \in \Xi$
  Using D-Xi with context $\Gamma, X <: U$ directly leads to the desired result.
- *Case* D-Psi: $V = \{new\, C_1\} \quad W = C_2 \quad (C_1, C_2) \notin \Psi$
  Using D-Psi with context $\Gamma, X <: U$ directly leads to the desired result.
- *Case* D-Sub: $\Gamma \vdash V <: U \quad \Gamma \vdash \text{disj}(U, W)$
  By the IH we get $\Gamma \vdash \text{disj}(U, W)$. Using the 2nd part of the lemma we obtain $\Gamma, X <: U \vdash V <: U$. The result follows from D-Sub.
- *Case* D-Arrow: $V = V_1 \to V_2 \quad W = C$
  Using D-Arrow with context $\Gamma, X <: U$ directly leads to the desired result.
- *Case* D-All: $V = \forall X <: V_1.\, V_2 \quad W = C$
  Using D-All with context $\Gamma, X <: U$ directly leads to the desired result.

2. $\Gamma \vdash S <: T$.
   - *Case* S-Refl: $T = S$
     Using S-Refl with context $\Gamma, X <: U$ directly leads to the desired result.
   - *Case* S-Trans: $\Gamma \vdash S <: U \quad \Gamma \vdash U <: T$
     The result follows from the IH and S-Trans.
   - *Case* S-Top: $T = \text{Top}$
     Using S-Top with context $\Gamma, X <: U$ directly leads to the desired result.
   - *Case* S-Sin: $S = \{new\, C\} \quad T = C$
     Using S-Sin with context $\Gamma, X <: U$ directly leads to the desired result.
   - *Case* S-TVar: $S = Y \quad Y <: T \in \Gamma$
     If $Y <: T \in \Gamma$, then $Y <: T \in \Gamma, X <: U$ and the result follows from S-TVar.
   - *Case* S-Arrow: $S = S_1 \to S_2 \quad T = T_1 \to T_2$
     $\Gamma \vdash T_1 <: S_1 \quad \Gamma \vdash S_2 <: T_2$
     The result follows from the IH and S-Arrow.
   - *Case* S-All: $S = \forall Y <: U_1.\, S_2 \quad T = \forall Y <: U_1.\, T_2 \quad \Gamma, Y <: U_1 \vdash S_2 <: T_2$
     Using the IH with context $\Gamma_{IH} = \Gamma, Y <: U_1$, we get $\Gamma, Y <: U_1, X <: U \vdash S_2 <: T_2$. From Lemma 3.1, $\Gamma, X <: U, Y <: U_1 \vdash S_2 <: T_2$ The result follows from S-All.
   - *Case* S-Psi: $S = C_1 \quad T = C_2 \quad (C_1, C_2) \in \Psi$
     Using S-Psi with context $\Gamma, X <: U$ directly leads to the desired result.
   - *Case* BS-Match1/2: $T_1 = T_n \quad T_2 = T_s\ match[Y]\{S_i \Rightarrow T_i\}\ or\ T_d$
     $\Gamma, Y <: V \vdash T_s <: S_n$
     $\forall m < n.\ \Gamma, Y <: \text{Top} \vdash \text{disj}(T_s, S_m)$
     From the IH we get $\Gamma, Y <: V, X <: U \vdash T_s <: S_n$. Using the 1st part of the lemma we obtain $\forall m < n.\ \Gamma, Y <: \text{Top}, X <: U \vdash \text{disj}(T_s, S_m)$. The result follows from Lemma 3.1 and BS-Match1/2.
   - *Case* BS-Match3/4: $T_1 = T_d \quad T_2 = T_s\ match[Y]\{S_i \Rightarrow T_i\}\ or\ T_d$
     $\forall n.\ \Gamma, Y <: \text{Top} \vdash \text{disj}(T_s, S_n)$
     Using the 1st part of the lemma we get $\forall n.\ \Gamma, Y <: \text{Top}, X <: U \vdash \text{disj}(T_s, S_n)$. The result follows from Lemma 3.1 and BS-Match3/4.
   - *Case* BS-Match5: $S = S_s\ match[Y]\{S_i \Rightarrow T_i\}\ or\ T_d \quad T = T_s\ match[Y]\{S_i \Rightarrow U_i\}\ or\ U_d$
     $\Gamma, \vdash S_s <: T_s$
     $\forall n.\ \Gamma, Y <: \text{Top} \vdash T_n <: U_n$
     $\Gamma \vdash T_d <: U_d$
     We use the IH on each premise and the result follows directly from Lemma 3.1 and BS-Match5.

3. By inspection of the subtyping rules, it is clear that typing assumbtions play no role in subtyping derivations.

4. By induction on a derivation of $\Gamma \vdash t : T$.
   - *Case* T-Var: $t = y \quad y : T \in \Gamma$
     If $y : T \in \Gamma$ then $y : T \in \Gamma, x : U$ and the result follows from T-Var.
   - *Case* T-Abs: $t = \lambda y : T_1 . t_2 \quad T = T_1 \to T_2 \quad \Gamma, y : T_1 \vdash t_2 : T_2$
     Using the IH with $\Gamma_{IH} = \Gamma, y : T_1$ we get $\Gamma, y : T_1, x : U \vdash t_2 : T_2$. From Lemma 3.1, $\Gamma, x : U, y : T_1 \vdash t_2 : T_2$. The result follows from T-Abs.
   - *Case* T-App: $t = t_1 t_2 \quad T = T_{12} \quad \Gamma \vdash t_1 : T_{11} \to T_{12} \quad \Gamma \vdash t_2 : T_{12}$
     We use the IH on each premise and the result follows from T-App.
   - *Case* T-TAbs: $t = \lambda X <: T_1 . t_2 \quad T = \forall X <: T_1 . T_2 \quad \Gamma, X <: T_1 \vdash t_2 : T_2$
     Using the IH with $\Gamma_{IH} = \Gamma, X <: T_1$ we get $\Gamma, X <: T_1, x : U \vdash t_2 : T_2$. From Lemma 3.1, $\Gamma, x : U, X <: T_1 \vdash t_2 : T_2$ The result follows from T-TAbs.
   - *Case* T-TApp: $t = t_1 T_2 \quad T = [X \mapsto T_2]T_{12}$
     $\Gamma \vdash t_1 : (\forall X <: T_{11} . T_{12}) \quad \Gamma \vdash T_2 <: T_{11}$
     Using the IH on the left premise we get $\Gamma, x : U \vdash t_1 : (\forall X <: T_{11} . T_{12})$. Using the 3rd part of the lemma on the right premise we obtain $\Gamma, x : U \vdash T_2 <: T_{11}$. The result follows from T-TApp.
   - *Case* T-Sub: $\Gamma \vdash t : S \quad \Gamma \vdash S <: T$
     Using the IH on the left premise we get $\Gamma, x : U \vdash t : S$. Using the 3rd part of the lemma on the right premise we obtain $\Gamma, x : U \vdash S <: T$. The result follows from T-Sub.
   - *Case* T-Class: $t = new\, C \quad T = \{new\, C\}$
     Using T-Class with context $\Gamma, x : U$ directly leads to the desired result.
   - *Case* BT-Match: $t = t_s\ match[Y]\{x_i : C_i \Rightarrow t_i\}or\, t_d \quad T = T_s\ match[Y]\{C_i \Rightarrow T_i\}or\, T_d$
     $\Gamma \vdash t_s : T_s \quad \Gamma, Y <: Top, x_i : C_i \vdash t_i : T_i \quad \Gamma \vdash t_d : T_d$
     We use the IH on each premise and the result follows directly from Lemma 3.1 and BT-Match.
5. By induction on a derivation of $\Gamma \vdash t : T$.
   - *Case* T-Var: $t = x \quad x : T \in \Gamma$
     If $y : T \in \Gamma$ then $y : T \in \Gamma, X <: U$ and the result follows from T-Var.
   - *Case* T-Abs: $t = \lambda x : T_1 . t_2 \quad T = T_1 \to T_2 \quad \Gamma, x : T_1 \vdash t_2 : T_2$
     Using the IH with $\Gamma_{IH} = \Gamma, x : T_1$ we get $\Gamma, x : T_1, X <: U \vdash t_2 : T_2$. From Lemma 3.1, $\Gamma, X <: U, x : T_1 \vdash t_2 : T_2$. The result follows from T-Abs.
   - *Case* T-App: $t = t_1 t_2 \quad T = T_{12} \quad \Gamma \vdash t_1 : T_{11} \to T_{12} \quad \Gamma \vdash t_2 : T_{12}$
     We use the IH on each premise and the result follows from T-App.
   - *Case* T-TAbs: $t = \lambda Y <: T_1 . t_2 \quad T = \forall Y <: T_1 . T_2 \quad \Gamma, Y <: T_1 \vdash t_2 : T_2$
     Using the IH with $\Gamma_{IH} = \Gamma, Y <: T_1$ we get $\Gamma, Y <: T_1, X <: U \vdash t_2 : T_2$. From Lemma 3.1, $\Gamma, X <: U, Y <: T_1, \vdash t_2 : T_2$. The result follows from T-TAbs.
   - *Case* T-TApp: $t = t_1 T_2 \quad T = [Y \mapsto T_2]T_{12}$
     $\Gamma \vdash t_1 : (\forall Y <: T_{11} . T_{12}) \quad \Gamma \vdash T_2 <: T_{11}$
     Using the IH on the left premise we get $\Gamma, X <: U \vdash t_1 : (\forall X <: T_{11} . T_{12})$. Using the 2nd part of the lemma on the right premise we obtain $\Gamma, X <: U \vdash T_2 <: T_{11}$. The result follows from T-TApp.
   - *Case* T-Sub: $\Gamma \vdash t : S \quad \Gamma \vdash S <: T$
     Using the IH on the left premise we get $\Gamma, S <: U \vdash t : S$. Using the 2nd part of the lemma on the right premise we obtain $\Gamma, X <: U \vdash S <: T$. The result follows from T-Sub.
   - *Case* T-Class: $t = new\, C \quad T = \{new\, C\}$
     Using T-Class with context $\Gamma, X <: U$ directly leads to the desired result.
   - *Case* BT-Match: $t = t_s\ match[Y]\{x_i : C_i \Rightarrow t_i\}or\, t_d \quad T = T_s\ match[Y]\{C_i \Rightarrow T_i\}or\, T_d$
     $\Gamma \vdash t_s : T_s \quad \Gamma, Y <: Top, x_i : C_i \vdash t_i : T_i \quad \Gamma \vdash t_d : T_d$

We use the IH on each premise and the result follows directly from Lemma 3.1 and BT-Match.

□

Lemma 3.3 (Strengthening). *If* $\Gamma, x : T, \Delta \vdash S <: T$, *then* $\Gamma, \Delta \vdash S <: T$.

Proof: By inspection of the subtyping rules, it is clear that typing assumbtions play no role in subtyping derivations.

□

Lemma 3.4 (Narrowing).

1. *If* $\Gamma, X <: Q, \Delta \vdash \text{disj}(S, T)$ *and* $\Gamma \vdash P <: Q$, *then* $\Gamma, X <: P, \Delta \vdash \text{disj}(S, T)$.
2. *If* $\Gamma, X <: Q, \Delta \vdash S <: T$ *and* $\Gamma \vdash P <: Q$, *then* $\Gamma, X <: P, \Delta \vdash S <: T$.
3. *If* $\Gamma, X <: Q, \Delta \vdash t : T$ *and* $\Gamma \vdash P <: Q$, *then* $\Gamma, X <: P, \Delta \vdash t : T$.

Proof: We prove 1. and 2. simultaneously by induction on two derivations of $\Gamma, X <: Q, \Delta \vdash \text{disj}(V, W)$ and $\Gamma, X <: Q, \Delta \vdash S <: T$. More precisely, the induction is done on the cumulative depth of both derivation tree.

1. $\Gamma, X <: Q, \Delta \vdash \text{disj}(V, W)$
   - *Case* D-Xi: $V = C_1 \quad W = C_2 \quad (C_1, C_1) \in \Xi$
     Using D-Xi with context $\Gamma, X <: P, \Delta$ directly leads to the desired result.
   - *Case* D-Psi: $V = C_1 \quad W = C_2 \quad (C_1, C_1) \notin \Psi$
     Using D-Psi with context $\Gamma, X <: P, \Delta$ directly leads to the desired result.
   - *Case* D-Sub: $\Gamma, X <: Q, \Delta \vdash V <: U \quad \Gamma, X <: Q, \Delta \vdash \text{disj}(U, W)$
     By the 2nd part of the lemma we get $\Gamma, X <: P, \Delta \vdash V <: U$. From the IH we obtain $\Gamma, X <: P, \Delta \vdash \text{disj}(U, W)$. The result follows from D-Sub.
   - *Case* D-Arrow: $V = T_1 \to T_2 \quad W = C$
     Using D-Arrow with context $\Gamma, X <: P, \Delta$ directly leads to the desired result.
   - *Case* D-All: $V = \forall X <: V_1.\, V_2 \quad W = C$
     Using D-All with context $\Gamma, X <: P, \Delta$ directly leads to the desired result.
2. $\Gamma, X <: Q, \Delta \vdash S <: T$
   - *Case* S-Refl: $T = S$
     Using S-Refl with context $\Gamma, X <: P, \Delta$ directly leads to the desired result.
   - *Case* S-Top: $T = \text{Top}$
     Using S-Top with context $\Gamma, X <: P, \Delta$ directly leads to the desired result.
   - *Case* S-Sin: $S = \{new\, C\} \quad T = C$
     Using S-Sin with context $\Gamma, X <: P, \Delta$ directly leads to the desired result.
   - *Case* S-Psi: $S = C_1 \quad T = C_2 \quad (C_1, C_2) \in \Psi$
     Using S-Psi with context $\Gamma, X <: P, \Delta$ directly leads to the desired result.
   - *Case* S-Trans: $\Gamma, X <: Q, \Delta \vdash S <: U \quad \Gamma, X <: Q, \Delta \vdash U <: T$
     The result follows directly from the IH and S-Trans.
   - *Case* S-Arrow: $S = S_1 \to S_2 \quad T = T_1 \to T_2$
     $\Gamma, X <: Q, \Delta \vdash T_1 <: S_1 \quad \Gamma, X <: Q, \Delta \vdash S_2 <: T_2$
     The result follows directly from the IH and S-Arrow.
   - *Case* S-TVar: $S = Y \quad Y <: T \in (\Gamma, X <: Q, \Delta)$
     By context well-formedness, $Y <: T \in (\Gamma, X <: Q, \Delta)$ can be decomposed into 3 subcases:
     Subcase $Y <: T \in \Gamma$:
     $Y <: T \in (\Gamma, X <: P, \Delta)$ and the result follows from S-TVar.
     Subcase $Y <: T \in \Delta$:
     Ditto.

Subcase $Y <: T = X <: Q$ (*i.e.* $Y = X$ and $T = Q$):
$Y <: P \in (\Gamma, Y <: P, \Delta)$. Using S-TVar we obtain $\Gamma, Y <: P, \Delta \vdash Y <: P$. Using $\Gamma \vdash P <: Q$ and Lemma 3.2 we get $\Gamma, Y <: P, \Delta \vdash P <: Q$. From S-Trans we obtain $\Gamma, Y <: P, \Delta \vdash Y <: Q$. Finally, we use the subcase assumption to substitute Y by X, Y by S and Q by T to obtain the desired result.

- *Case* S-All: $S = \forall Y <: U_1.\ S_2 \quad T = \forall Y <: U_1.\ T_2 \quad \Gamma, X <: Q, \Delta, Y <: U_1 \vdash S_2 <: T_2$
  We instantiate the induction hypothesis with $\Delta_{IH} = (\Delta, Y <: U_1)$ to obtain $\Gamma, X <: P, \Delta, Y <: U_1 \vdash S_2 <: T_2$. The result follows from S-All.
- *Case* BS-Match1/2: $T_1 = [Y \mapsto U]T_n \quad T_2 = T_s\ \mathit{match}[Y]\{S_i \Rightarrow T_i\}\mathit{or}\ T_d$
  $\Gamma, X <: Q, \Delta, Y <: U \vdash T_s <: S_n$
  $\forall m < n.\ \Gamma, X <: Q, \Delta, Y <: \mathrm{Top} \vdash \mathrm{disj}(T_s, S_m)$
  By the 1st part of the lemma (with $\Delta_{1st} = \Delta, Y <: \mathrm{Top}$) we get $\forall m < n.\ \Gamma, X <: P, \Delta, Y <: \mathrm{Top} \vdash \mathrm{disj}(T_s, S_m)$. From the IH (with $\Delta_{IH} = \Delta, Y <: U$) we obtain $\Gamma, X <: P, \Delta, Y <: U \vdash T_s <: S_n$. The result follows from BS-Match1/2.
- *Case* BS-Match3/4: $T_1 = T_d \quad T_2 = T_s\ \mathit{match}[Y]\{S_i \Rightarrow T_i\}\mathit{or}\ T_d$
  $\forall n.\ \Gamma, X <: Q, \Delta, Y <: \mathrm{Top} \vdash \mathrm{disj}(T_s, S_n)$
  The result follows from the 1st part of the lemma (with $\Delta_{1st} = \Delta, Y <: \mathrm{Top}$) and BS-Match3/4.
- *Case* BS-Match5: $S = S_s\ \mathit{match}[Y]\{U_i \Rightarrow S_i\}\mathit{or}\ S_d \quad T = T_s\ \mathit{match}[Y]\{U_i \Rightarrow T_i\}\mathit{or}\ T_d$
  $\Gamma, X <: Q, \Delta \vdash S_s <: T_s$
  $\forall n.\ \Gamma, X <: Q, \Delta, Y <: \mathrm{Top} \vdash S_n <: T_n$
  $\Gamma, X <: Q, \Delta \vdash S_d <: T_d$
  The result follows directly from the IH (with $\Delta_{IH} = \Delta, Y <: \mathrm{Top}$) and BS-Match5.

3. By induction on a derivation of $\Gamma, X <: Q, \Delta \vdash t : T$.
   - *Case* T-Abs: $t = \lambda x : T_1 t_2 \quad T = T_1 \to T_2 \quad \Gamma, X <: Q, \Delta, x : T_1 \vdash t_2 : T_2$
     We instantiate the IH with $\Delta_{IH} = (\Delta, x : T_1)$ to obtain $\Gamma, X <: P, \Delta, x : T_1 \vdash t_2 : T_2$. The result follows from S-All.
   - *Case* T-App: $t = t_1 t_2 \quad T = T_{12} \quad \Gamma, X <: Q, \Delta \vdash t_1 : T_{11} \to T_{12} \quad \Gamma, X <: Q, \Delta \vdash t_2 : T_{11}$
     The result follows directly from the IH and T-App.
   - *Case* T-TAbs: $t = \lambda Y <: T_1.\ t_2 \quad T = \forall Y <: T_1.\ T_2 \quad \Gamma, X <: Q, \Delta, Y <: T_1 \vdash t_2 : T_2$
     We instantiate the IH with $\Delta_{IH} = (\Delta, Y <: T_1)$ to obtain $\Gamma, X <: P, \Delta, Y <: T_1 \vdash t_2 : T_2$. The result follows from T-TAbs.
   - *Case* T-TApp: $t = t_1 T_2 \quad T = [Y \mapsto T_2]T_{12}$
     $\Gamma, X <: Q, \Delta \vdash t_1 : (\forall Y <: T_{11}.\ T_{12}) \quad \Gamma, X <: Q, \Delta \vdash T_2 <: T_{11}$
     The result follows directly from the IH and T-TApp.
   - *Case* T-Var: $t = x \quad x : T \in \Gamma, X <: Q, \Delta$
     Either $x : T \in \Gamma$ or $x : T \in \Delta$. In both cases we have $x : T \in \Gamma, X <: P, \Delta$. The result follows from T-Var.
   - *Case* T-Sub: $\Gamma, X <: Q, \Delta \vdash t : S \quad \Gamma, X <: Q, \Delta \vdash S <: T$
     By the second part of the lemma we get $\Gamma, X <: P, \Delta \vdash S <: T$. From the IH we obtain $\Gamma, X <: P, \Delta \vdash t : S$. The result follows from T-Sub.
   - *Case* T-Class: $t = \mathit{new}\ C \quad T = \{\mathit{new}\ C\}$
     Using T-Class with context $\Gamma, X <: P, \Delta$ directly leads to the desired result.
   - *Case* BT-Match: $t = t_s\ \mathit{match}[Y]\{x_i : C_i \Rightarrow t_i\}\mathit{or}\ t_d \quad T = T_s\ \mathit{match}[Y]\{C_i \Rightarrow T_i\}\mathit{or}\ T_d$
     $\Gamma, X <: Q, \Delta \vdash t_s : T_s$
     $\Gamma, X <: Q, \Delta, Y <: \mathrm{Top}, x_i : C_i \vdash t_i : T_i$
     $\Gamma, X <: Q, \Delta \vdash t_d : T_d$

The result follows directly from the IH and BT-Match.

□

Lemma 3.5 (Substitution).

1. *If* $\Gamma, X <: Q, \Delta \vdash \mathrm{disj}(S, T)$ *and* $\Gamma \vdash P <: Q$,
   then $\Gamma, [X \mapsto P]\Delta \vdash \mathrm{disj}([X \mapsto P]S, [X \mapsto P]T)$.
2. *If* $\Gamma, X <: Q, \Delta \vdash S <: T$ *and* $\Gamma \vdash P <: Q$,
   then $\Gamma, [X \mapsto P]\Delta \vdash [X \mapsto P]S <: [X \mapsto P]T$.
3. *If* $\Gamma, X <: Q, \Delta \vdash t : T$ *and* $\Gamma \vdash P <: Q$,
   then $\Gamma, [X \mapsto P]\Delta \vdash [X \mapsto P]t : [X \mapsto P]T$.
4. *If* $\Gamma, x : Q, \Delta \vdash t : T$ *and* $\Gamma \vdash q : Q$,
   then $\Gamma, \Delta \vdash [x \mapsto q]t : T$.

Proof: We prove 1. and 2. simultaneously by induction on two derivations of $\Gamma, X <: Q, \Delta \vdash \mathrm{disj}(V, W)$ and $\Gamma, X <: Q, \Delta \vdash S <: T$. More precisely, the induction is done on the cumulative depth of both derivation tree.

1. $\Gamma, X <: Q, \Delta \vdash \mathrm{disj}(V, W)$
   - *Case* D-Xi: $V = C_1 \quad W = C_2 \quad (C_1, C_2) \in \Xi$
     Since $[X \mapsto P]C_1 = C_1$ and $[X \mapsto P]C_2 = C_2$, we can use D-Xi with context $\Gamma, [X \mapsto P]\Delta$ to obtain the desired result.
   - *Case* D-Psi: $V = \{new\, C_1\} \quad W = C_2 \quad (C_1, C_2) \notin \Psi$
     Since $[X \mapsto P]\{new\, C_1\} = \{new\, C_1\}$ and $[X \mapsto P]C_2 = C_2$, we can use D-Psi with context $\Gamma, [X \mapsto P]\Delta$ to obtain the desired result.
   - *Case* D-Sub: $\Gamma, X <: Q, \Delta \vdash V <: U \quad \Gamma, X <: Q, \Delta \vdash \mathrm{disj}(U, W)$
     By the IH we get $\Gamma, [X \mapsto P]\Delta \vdash \mathrm{disj}([X \mapsto P]U, [X \mapsto P]W)$. Using the 2nd part of the lemma we obtain $\Gamma, [X \mapsto P]\Delta \vdash [X \mapsto P]V <: [X \mapsto P]U$. The result follows from D-Sub.
   - *Case* D-Arrow: $V = V_1 \to V_2 \quad W = C$
     Since $[X \mapsto P](V_1 \to V_2) = [X \mapsto P]V_1 \to [X \mapsto P]V_2$ and $[X \mapsto P]C = C$, we can use D-Arrow with context $\Gamma, [X \mapsto P]\Delta$ to obtain the desired result.
   - *Case* D-All: $V = \forall Y <: V_1.\, V_2 \quad W = C$
     Since $[X \mapsto P](\forall Y <: V_1.\, V_2) = \forall Y <: [X \mapsto P]V_1.\, [X \mapsto P]V_2$ and $[X \mapsto P]C = C$, we can use D-All with context $\Gamma, [X \mapsto P]\Delta$ to obtain the desired result.
2. $\Gamma, X <: Q, \Delta \vdash S <: T$.
   - *Case* S-Refl: $T = S$
     The result follows directly from S-Refl.
   - *Case* S-Trans: $\Gamma, X <: Q, \Delta \vdash S <: U \quad \Gamma, X <: Q, \Delta \vdash U <: T$
     The result follows directly from the IH and S-Trans.
   - *Case* S-Top: $T = \mathrm{Top}$
     $[X \mapsto P]\mathrm{Top} = \mathrm{Top}$ and the result follows from S-Top.
   - *Case* S-Sin: $S = \{new\, C\} \quad T = C$
     $[X \mapsto P]\{new\, C\} = \{new\, C\}$, $[X \mapsto P]C = C$, and the result follows from S-Sin.
   - *Case* S-TVar: $S = Y \quad Y <: T \in (\Gamma, X <: Q, \Delta)$
     By context well-formedness, $Y <: T \in (\Gamma, X <: Q, \Delta)$ can be decomposed into 3 subcases:
     Subcase $Y <: T \in \Gamma$:
     By context well-formedness X does not appear in Γ consequently is also absent from Y and T. Hence $Y = [X \mapsto P]Y$, $T = [X \mapsto P]T$ and $[X \mapsto P]Y <: [X \mapsto P]T \in (\Gamma, [X \mapsto P]\Delta)$. The result follows from S-TVar.
     Subcase $Y <: T \in \Delta$:

$[X \mapsto P]Y <: [X \mapsto P]T \in [X \mapsto P]\Delta$ and $[X \mapsto P]Y <: [X \mapsto P]T \in (\Gamma, [X \mapsto P]\Delta)$. The result follows from S-TVar.

Subcase $Y <: T = X <: Q$ (*i.e.* $Y = X$ and $T = Q$):

By context well-formedness, X doesn't appear in Q, and $[X \mapsto P]T = [X \mapsto P]Q = Q$. Also $[X \mapsto P]S = [X \mapsto P]X = P$ and $[X \mapsto P]T = Q$. As a result, $\Gamma \vdash P <: Q$ implies $\Gamma \vdash [X \mapsto P]S <: [X \mapsto P]T$. Using Lemma 3.2 we get $\Gamma, [X \mapsto P]\Delta \vdash [X \mapsto P]S <: [X \mapsto P]T$, as required.

- *Case* S-Arrow: $S = S_1 \to S_2 \quad T = T_1 \to T_2 \quad \Gamma, X <: Q, \Delta \vdash T_1 <: S_1 \quad \Gamma, X <: Q, \Delta \vdash S_2 <: T_2$

  $[X \mapsto P](S_1 \to S_2) = [X \mapsto P]S_1 \to [X \mapsto P]S_2$ and $[X \mapsto P](T_1 \to T_2) = [X \mapsto P]T_1 \to [X \mapsto P]T_2$. The result follows from the IH and S-Arrow.
- *Case* S-All: $S = \forall Y <: U_1.\, S_2 \quad T = \forall Y <: U_1.\, T_2 \quad \Gamma, X <: Q, \Delta, Y <: U_1 \vdash S_2 <: T_2$

  We instantiate the IH with $\Delta_{IH} = (\Delta, Y <: U_1)$ to obtain $\Gamma, [X \mapsto P]\Delta, Y <: [X \mapsto P]U_1 \vdash [X \mapsto P]S_2 <: [X \mapsto P]T_2$. Using S-All, we get $\Gamma, [X \mapsto P]\Delta \vdash (\forall Y <: [X \mapsto P]U_1.\, [X \mapsto P]S_2) <: (\forall Y <: [X \mapsto P]U_1.\, [X \mapsto P]T_2)$, that is, $\Gamma, [X \mapsto P]\Delta \vdash [X \mapsto P](\forall Y <: U_1.\, S_2) <: [X \mapsto P](\forall Y <: U_1.\, T_2)$, as required.
- *Case* S-Psi: $S = C_1 \quad T = C_2 \quad (C_1, C_2) \in \Psi$

  $[X \mapsto P]C_1 = C_1$ and $[X \mapsto P]C_2 = C_2$. The result follows from S-Psi.
- *Case* BS-Match1/2: $T_1 = T_n \quad T_2 = T_s\ \mathit{match}[Y]\{S_i \Rightarrow T_i\}\mathit{or}\ T_d \quad \Gamma, X <: Q, \Delta, Y <: U \vdash T_s <: S_n \quad \forall m < n.\ \Gamma, X <: Q, \Delta, Y <: \mathrm{Top} \vdash \mathrm{disj}(T_s, S_m)$

  Using the 1st part of the lemma we get $\forall m < n.\ \Gamma, [X \mapsto P]\Delta, Y <: [X \mapsto P]U \vdash \mathrm{disj}([X \mapsto P]T_s, [X \mapsto P]S_m)$. By IH, $\Gamma, [X \mapsto P]\Delta, Y <: [X \mapsto P]U \vdash [X \mapsto P]T_s <: S_n$. $[X \mapsto P]T = [X \mapsto P]T_s\ \mathit{match}[Y]\{[X \mapsto P]S_i \Rightarrow [X \mapsto P]T_i\}\mathit{or}\ [X \mapsto P]T_d$. The result follows from BS-Match1/2.
- *Case* BS-Match3/4: $S = T_d \quad T = T_s\ \mathit{match}[Y]\{S_i \Rightarrow T_i\}\mathit{or}\ T_d \quad \forall n.\ \Gamma, X <: Q, \Delta \vdash \mathrm{disj}(T_s, S_n)$

  By the 1st part of the lemma we get $\forall n.\ \Gamma, [X \mapsto P]\Delta, Y <: \mathrm{Top} \vdash \mathrm{disj}([X \mapsto P]T_s, [X \mapsto P]S_n)$ and the result follows from BS-Match3/4.
- *Case* BS-Match5: $S = S_s\ \mathit{match}[Y]\{U_i \Rightarrow S_i\}\mathit{or}\ S_d \quad T = T_s\ \mathit{match}[Y]\{U_i \Rightarrow T_i\}\mathit{or}\ T_d \quad \Gamma, X <: Q, \Delta \vdash S_s <: T_s \quad \forall n.\ \Gamma, X <: Q, \Delta, Y <: \mathrm{Top} \vdash S_n <: T_n \quad \Gamma, X <: Q, \Delta \vdash S_d <: T_d$

  The result follows directly from the IH.

3. By induction on a derivation of $\Gamma, X <: Q, \Delta \vdash t : T$.
   - *Case* T-Var: $t = x \quad x : T \in \Gamma, X <: Q, \Delta$

     $[X \mapsto P]x = x$. T-Var's premise can be divided in two subcases:

     Subcase $x : T \in \Gamma$:

     Context well-formedness impiles that there is no occurence of X in T and $[X \mapsto P]T = T$. Also, $x : T \in \Gamma, [X \mapsto P]\Delta$ and the result follows from T-Var.

     Subcase $x : T \in \Delta$:

     Context well-formedness impiles that there is a unique occurence of $x : T$ in Δ, that is, there exists Δ_1, Δ_2 such that $\Delta = \Delta_1, x : T, \Delta_2$, $x \notin \Delta_1$ and $x \notin \Delta_2$. As a result, $[X \mapsto P]\Delta = [X \mapsto P]\Delta_1, x : [X \mapsto P]T, [X \mapsto P]\Delta_2$ and $x : [X \mapsto P]T \in \Gamma, [X \mapsto P]\Delta$. The result follows from T-Var.
   - *Case* T-Abs: $t = \lambda x : T_1 t_2 \quad T = T_1 \to T_2 \quad \Gamma, X <: Q, \Delta, x : T_1 \vdash t_2 : T_2$

     $[X \mapsto P](\lambda x : T_1 t_2) = \lambda x : [X \mapsto P]T_1 [X \mapsto P]t_2$ and $[X \mapsto P](T_1 \to T_2) = [X \mapsto P]T_1 \to [X \mapsto P]T_2$. We instantiate the IH with $\Delta_{IH} = (\Delta, x : T_1)$ to obtain $\Gamma, [X \mapsto P]\Delta, x : [X \mapsto P]T_1 \vdash [X \mapsto P]t_2 : [X \mapsto P]T_2$. The result follows from T-Abs.

- *Case* T-App: $t = t_1 t_2 \quad T = T_{12} \quad \Gamma, X <: Q, \Delta \vdash t_1 : T_{11} \to T_{12} \quad \Gamma, X <: Q, \Delta \vdash t_2 : T_{11}$
  $[X \mapsto P](t_1 t_2) = [X \mapsto P]t_1 [X \mapsto P]t_2$ and $[X \mapsto P]T_{11} \to T_{12} = [X \mapsto P]T_{11} \to [X \mapsto P]T_{12}$. The result follows directly from the IH and T-App.
- *Case* T-TAbs: $t = \lambda Y <: T_1.\ t_2 \quad T = \forall Y <: T_1.\ T_2 \quad \Gamma, X <: Q, \Delta, Y <: T_1 \vdash t_2 : T_2$
  $[X \mapsto P](\lambda Y <: T_1.\ t_2) = \lambda Y <: [X \mapsto P]T_1.\ [X \mapsto P]t_2$ and $[X \mapsto P](\forall Y <: T_1.\ T_2) = \forall Y <: [X \mapsto P]T_1.\ [X \mapsto P]T_2$. We instantiate the IH with $\Delta_{IH} = (\Delta, Y <: T_1)$ to obtain $\Gamma, [X \mapsto P]\Delta, Y <: [X \mapsto P]T_1 \vdash [X \mapsto P]t_2 : [X \mapsto P]T_2$. The result follows from T-TAbs.
- *Case* T-TApp: $t = t_1 T_2 \quad T = [Y \mapsto T_2]T_{12}$
  $\Gamma, X <: Q, \Delta \vdash t_1 : (\forall Y <: T_{11}.\ T_{12}) \quad \Gamma, X <: Q, \Delta \vdash T_2 <: T_{11}$
  By the IH, $\Gamma, [X \mapsto P]\Delta \vdash [X \mapsto P]t_1 : [X \mapsto P](\forall Y <: T_{11}.\ T_{12})$, i.e., $\Gamma, [X \mapsto P]\Delta \vdash [X \mapsto P]t_1 : \forall Y <: [X \mapsto P]T_{11}.\ [X \mapsto P]T_{12}$. Using the second part of the lemma we get, $\Gamma, [X \mapsto P]\Delta \vdash [X \mapsto P]T_2 <: [X \mapsto P]T_{11}$. By T-TApp we get, $\Gamma, [X \mapsto P]\Delta \vdash [X \mapsto P]t_1 [X \mapsto P]T_2 : [Y \mapsto [X \mapsto P]T_2][X \mapsto P]T_{12}$, i.e., $\Gamma, [X \mapsto P]\Delta \vdash [X \mapsto P](t_1 T_2) : [X \mapsto P]([Y \mapsto T_2]T_{12})$, as required.
- *Case* T-Sub: $\Gamma, X <: Q, \Delta \vdash t : S \quad \Gamma, X <: Q, \Delta \vdash S <: T$
  By the IH, $\Gamma, [X \mapsto P]\Delta \vdash [X \mapsto P]t : [X \mapsto P]S$. Using the second part of the lemma we get, $\Gamma, [X \mapsto P]\Delta \vdash [X \mapsto P]S <: [X \mapsto P]T$. The result follows from T-Sub.
- *Case* T-Class: $t = new\ C \quad T = \{new\ C\}$
  $[X \mapsto P]new\ C = new\ C$ and using T-Class with context $\Gamma, [X \mapsto P]\Delta$ directly leads to the desired result.
- *Case* BT-Match: $t = t_s\ match[Y]\{x_i : C_i \Rightarrow t_i\}or\ t_d \quad T = T_s\ match[Y]\{C_i \Rightarrow T_i\}or\ T_d$
  $\Gamma, X <: Q, \Delta \vdash t_s : T_s$
  $\Gamma, X <: Q, \Delta, Y <: Top, x_i : C_i \vdash t_i : T_i$
  $\Gamma, X <: Q, \Delta \vdash t_d : T_d$
  $[X \mapsto P](t_s\ match[Y]\{x_i : C_i \Rightarrow t_i\}or\ t_d) = [X \mapsto P]t_s\ match[Y]\{x_i : C_i \Rightarrow [X \mapsto P]t_i\}or\ [X \mapsto P]t_d$. $[X \mapsto P](T_s\ match[Y]\{C_i \Rightarrow T_i\}or\ T_d) = [X \mapsto P]T_s\ match[Y]\{C_i \Rightarrow [X \mapsto P]T_i\}or\ [X \mapsto P]T_d$. The result follows directly from the IH and T-App.

4. By induction on a derivation of $\Gamma, x : Q, \Delta \vdash t : T$.
   - *Case* T-Var: $t = y \quad y : T \in \Gamma, x : Q, \Delta$
     By context well-formedness, the premise can be decomposed into 3 subcases:
     Subcase $y : T \in \Gamma$:
     $[x \mapsto q]y = y$ and $y : T \in \Gamma, \Delta$. The result follows from T-Var.
     Subcase $y : T \in \Delta$:
     Ditto.
     Subcase $y : T = x : Q$ (i.e. $y = x$ and $T = Q$):
     Using $\Gamma \vdash q : Q$ and Lemma 3.2 we get $\Gamma, \Delta \vdash q : Q$. Since $[x \mapsto q]y = q$ and $T = Q$, $\Gamma, \Delta \vdash [x \mapsto q]y : T$, as required.
   - *Case* T-Abs: $t = \lambda y : T_1 t_2 \quad T = T_1 \to T_2 \quad \Gamma, x : Q, \Delta, y : T_1 \vdash t_2 : T_2$
     We instantiate the IH with $\Delta_{IH} = (\Delta, y : T_1)$ to obtain $\Gamma, \Delta, y : T_1 \vdash [x \mapsto q]t_2 : T_2$. $[x \mapsto q](\lambda y : T_1 t_2) = \lambda y : T_1 [x \mapsto q]t_2$ and the result follows from T-Abs.
   - *Case* T-App: $t = t_1 t_2 \quad T = T_{12} \quad \Gamma, x : Q, \Delta \vdash t_1 : T_{11} \to T_{12} \quad \Gamma, x : Q, \Delta \vdash t_2 : T_{11}$
     $[x \mapsto q](t_1 t_2) = ([x \mapsto q]t_1)([x \mapsto q]t_2)$ and the result follows from the IH and T-App.
   - *Case* T-TAbs: $t = \lambda X <: U_1.\ t_2 \quad T = \forall X <: U_1.\ T_2 \quad \Gamma, x : Q, \Delta, X <: U_1 \vdash t_2 : T_2$
     We instantiate the IH with $\Delta_{IH} = (\Delta, X <: U_1)$ to obtain $\Gamma, \Delta, X <: U_1 \vdash [x \mapsto q]t_2 : T_2$. $[x \mapsto q](\lambda X <: U_1.\ t_2) = \lambda X <: U_1.\ [x \mapsto q]t_2$ and the result follows from T-TAbs.
   - *Case* T-TApp: $t = t_1 T_2 \quad T = [X \mapsto T_2]T_{12}$
     $\Gamma, x : Q, \Delta \vdash t_1 : (\forall X <: U_1.\ T_{12}) \quad \Gamma, x : Q, \Delta \vdash T_2 <: U_1$

By Lemma 3.3, $\Gamma, \Delta \vdash T_2 <: U_1$. From the IH we get $\Gamma, \Delta \vdash [x \mapsto q]t_1 : (\forall X <: U_1.\ T_{12})$. $[x \mapsto q](t_1 T_2) = [x \mapsto q]t_1 T_2$ and the result follows from T-TApp.

- *Case* T-Sub: $\Gamma, x : Q, \Delta \vdash t : S \quad \Gamma, x : Q, \Delta \vdash S <: T$
  By Lemma 3.3, $\Gamma, \Delta \vdash S <: T$. The result follows from the IH and T-Sub.
- *Case* T-Class: $t = \mathit{new}\, C \quad T = \{\mathit{new}\, C\}$
  Since $[x \mapsto q]\mathit{new}\, C = \mathit{new}\, C$, using T-Class with context Γ, Δ directly leads to the desired result.
- *Case* BT-Match: $t = t_s\ \mathit{match}[Y]\{x_i : C_i \Rightarrow t_i\}\mathit{or}\, t_d \quad T = T_s\ \mathit{match}[Y]\{C_i \Rightarrow T_i\}\mathit{or}\, T_d$
  $\Gamma, x : Q, \Delta \vdash t_s : T_s$
  $\Gamma, x : Q, \Delta, Y <: \mathrm{Top}, x_i : C_i \vdash t_i : T_i$
  $\Gamma, x : Q, \Delta \vdash t_d : T_d$
  $[x \mapsto q](t_s\ \mathit{match}[Y]\{x_i : C_i \Rightarrow t_i\}\mathit{or}\, t_d) = [x \mapsto q]t_s\ \mathit{match}[Y]\{x_i : C_i \Rightarrow [x \mapsto q]t_i\}\mathit{or}\, [x \mapsto q]t_d$, and the result follows from the IH and BT-Match.

□

Lemma 3.6 (Disjointness/subtyping exclusivity).
The type disjointness and subtyping relations are mutually exclusive.

Proof: We prove mutual exclusivity of type disjointness and subtyping by first defining $[\![\cdot]\!]_\Gamma$, a mapping from System FM types (in a given context Γ) into non-empty subsets of a newly defined set P. We then show that the subtyping relation in FM corresponds to a subset relation in P, and that the type disjointness relation in FM (disj) corresponds to set disjointness relation in P.

This set-theoretical view of subtyping and disjointness renders the proof trivial. Indeed, suppose there exist two types S and T with $\Gamma \vdash S <: T$ and $\Gamma \vdash \mathrm{disj}(S, T)$. $[\![S]\!]_\Gamma$ and $[\![T]\!]_\Gamma$ are two non-empty sets which are both intersecting and disjoint, a contradiction.

We first define P as $P = \{\Lambda, V\} \cup C$. Elements of P can be understood as equivalence classes for System FM values: Λ corresponds to all abstraction values, V corresponds to type abstraction value, and elements of C correspond to their respective constructors. The definition of $[\![\cdot]\!]_\Gamma$ makes this correspondence apparent.

We define $[\![\cdot]\!]_\Gamma$, a mapping of System FM types into subsets of P in a given context Γ:

$$
\begin{aligned}
[\![\mathrm{Top}]\!]_\Gamma &= P \\
[\![X]\!]_\Gamma &= [\![T]\!]_\Gamma \quad \text{where } X <: T \in \Gamma \\
[\![T_1 \to T_2]\!]_\Gamma &= \{\Lambda\} \\
[\![\forall X <: U_1.\ T_2]\!]_\Gamma &= \{V\} \\
[\![\{\mathit{new}\, C_1\}]\!]_\Gamma &= \{C_1\} \\
[\![C_1]\!]_\Gamma &= \{c \in C \mid (c, C_1) \in \Psi\} \\
[\![T_s\ \mathit{match}[X]\{S_i \Rightarrow T_i\}\mathit{or}\, T_d]\!]_\Gamma &= \begin{cases} [\![[X \mapsto U]T_n]\!]_\Gamma & \text{if } [\![T_s]\!]_{\Gamma, X <: U} \subset [\![S_n]\!]_{\Gamma, X <: U} \\ & \quad \text{and } \forall m < n.\ [\![T_s]\!]_{\Gamma, X <: \mathrm{Top}} \cap \\ & \qquad [\![S_m]\!]_{\Gamma, X <: \mathrm{Top}} = \{\} \\ [\![T_d]\!]_\Gamma & \text{if } \forall m.\ [\![T_s]\!]_{\Gamma, X <: U} \cap [\![S_m]\!]_{\Gamma, X <: \mathrm{Top}} = \{\} \\ P & \text{otherwise} \end{cases}
\end{aligned}
$$

We show that the subtyping relation in FM corresponds to a subset relation in P, and that the type disjointness relation in FM (disj) corresponds to set disjointness in P. In other words, we prove the follows statements:

1. $\Gamma \vdash S <: T$ implies $[\![S]\!]_\Gamma \subset [\![T]\!]_\Gamma$,
2. $\Gamma \vdash \mathrm{disj}(S, T)$ implies $[\![S]\!]_\Gamma \cap [\![T]\!]_\Gamma = \{\}$,

Both statements are proved simultaneously by induction on derivations of $\Gamma \vdash \mathrm{disj}(V, W)$ and $\Gamma \vdash S <: T$. More precisely, the induction is done on the cumulative depth of both derivation tree.

1. ($\Gamma \vdash S <: T$ implies $[\![S]\!]_\Gamma \subset [\![T]\!]_\Gamma$)
   - *Case* S-Refl: $T = S$
     $[\![S]\!]_\Gamma = [\![T]\!]_\Gamma$ and the result is immediate.
   - *Case* S-Trans: $\Gamma \vdash S <: U \quad \Gamma \vdash U <: T$
     By the IH, $[\![S]\!]_\Gamma \subset [\![U]\!]_\Gamma$ and $[\![U]\!]_\Gamma \subset [\![T]\!]_\Gamma$. From subset transitivity, $[\![S]\!]_\Gamma \subset [\![T]\!]_\Gamma$, as required.
   - *Case* S-Top: $T = \mathrm{Top}$
     $[\![\mathrm{Top}]\!]_\Gamma = P$ coincides with the codomain of $[\![\cdot]\!]_\Gamma$. Therefore, for any type T, $[\![T]\!]_\Gamma \subset [\![\mathrm{Top}]\!]_\Gamma$, as required.
   - *Case* S-Sin: $S = \{\mathit{new}\, C_1\} \quad T = C_1$
     $[\![\{\mathit{new}\, C_1\}]\!]_\Gamma = \{C_1\}$ and $[\![C_1]\!]_\Gamma = \{c \in C \mid (c, C_1) \in \Psi\}$. The result follows by reflexivity of Ψ.
   - *Case* S-TVar: $S = X \quad X <: T \in \Gamma$
     By definition $[\![X]\!]_\Gamma = [\![T]\!]_\Gamma$, and the result is immediate.
   - *Case* S-Arrow: $S = S_1 \to S_2 \quad T = T_1 \to T_2$
     $\Gamma \vdash T_1 <: S_1 \quad \Gamma \vdash S_2 <: T_2$
     $[\![S_1 \to S_2]\!]_\Gamma = [\![T_1 \to T_2]\!]_\Gamma = \{\Lambda\}$ and the result is immediate.
   - *Case* S-All: $S = \forall X <: U_1.\, S_2 \quad T = \forall X <: U_1.\, T_2 \quad \Gamma, X <: U_1 \vdash S_2 <: T_2$
     $[\![\forall X <: U_1.\, S_2]\!]_\Gamma = [\![\forall X <: U_1.\, T_2]\!]_\Gamma = \{V\}$ and the result is immediate.
   - *Case* S-Psi: $S = C_1 \quad T = C_2 \quad (C_1, C_2) \in \Psi$
     $[\![C_1]\!]_\Gamma = \{c \in C \mid (c, C_1) \in \Psi\}$ and $[\![C_2]\!]_\Gamma = \{c \in C \mid (c, C_2) \in \Psi\}$. By transitivity of Ψ, $[\![C_1]\!]_\Gamma \subset [\![C_2]\!]_\Gamma$, as required.
   - *Case* BS-Match1/2: $T_1 = [X \mapsto U]T_n \quad T_2 = T_s\ \mathit{match}[X]\{S_i \Rightarrow T_i\}\mathit{or}\, T_d$
     $\Gamma, X <: U \vdash T_s <: S_n$
     $\forall m < n.\ \Gamma, X <: \mathrm{Top} \vdash \mathrm{disj}(T_s, S_m)$
     Using the IH we get $[\![T_s]\!]_{\Gamma, X <: U} \subset [\![S_n]\!]_{\Gamma, X <: U}$, $\forall m < n.\ [\![T_s]\!]_{\Gamma, X <: \mathrm{Top}} \cap [\![S_m]\!]_{\Gamma, X <: \mathrm{Top}} = \{\}$ and $[\![T_s\ \mathit{match}[X]\{S_i \Rightarrow T_i\}\mathit{or}\, T_d]\!]_\Gamma = [\![[X \mapsto U]T_n]\!]_\Gamma$ (by definition of $[\![\cdot]\!]$). This last equality implies both $[\![T_1]\!]_\Gamma \subset [\![T_2]\!]_\Gamma$ and $[\![T_2]\!]_\Gamma \subset [\![T_1]\!]_\Gamma$, as required.
   - *Case* BS-Match3/4: $T_1 = T_d \quad T_2 = T_s\ \mathit{match}[X]\{S_i \Rightarrow T_i\}\mathit{or}\, T_d$
     $\forall n.\ \Gamma, X <: \mathrm{Top} \vdash \mathrm{disj}(T_s, S_n)$
     By the IH $\forall n.\ [\![T_s]\!]_{\Gamma, X <: \mathrm{Top}} \cap [\![S_n]\!]_{\Gamma, X <: \mathrm{Top}} = \{\}$. By definition of $[\![\cdot]\!]$, $[\![T_s\ \mathit{match}[X]\{S_i \Rightarrow T_i\}\mathit{or}\, T_d]\!]_\Gamma = [\![T_d]\!]_\Gamma$. This equality implies both $[\![T_1]\!]_\Gamma \subset [\![T_2]\!]_\Gamma$ and $[\![T_2]\!]_\Gamma \subset [\![T_1]\!]_\Gamma$, the desired result for BS-Match3 and BS-Match4 respectively.
   - *Case* BS-Match5: $S = S_s\ \mathit{match}[X]\{U_i \Rightarrow S_i\}\mathit{or}\, S_d \quad T = T_s\ \mathit{match}[X]\{U_i \Rightarrow T_i\}\mathit{or}\, T_d$
     $\Gamma \vdash S_s <: T_s$
     $\forall n.\ \Gamma, X <: \mathrm{Top} \vdash S_n <: T_n$
     $\Gamma \vdash S_d <: T_d$
     By case analysis on $[\![T]\!]_\Gamma$:
     a. $[\![T]\!]_\Gamma = [\![[X \mapsto U]T_n]\!]_\Gamma$ and $[\![T_s]\!]_{\Gamma, X <: U} \subset [\![U_n]\!]_{\Gamma, X <: U}$
        and $\forall m < n.\ [\![T_s]\!]_{\Gamma, X <: \mathrm{Top}} \cap [\![U_m]\!]_{\Gamma, X <: \mathrm{Top}} = \{\}$:
        The IH gives $[\![S_s]\!]_{\Gamma, X <: \mathrm{Top}} \subset [\![T_s]\!]_{\Gamma, X <: \mathrm{Top}}$ (Lemma 3.2). Using $\forall m < n.\ [\![T_s]\!]_{\Gamma, X <: \mathrm{Top}} \cap [\![U_m]\!]_{\Gamma, X <: \mathrm{Top}} = \{\}$, we get $\forall m < n.\ [\![S_s]\!]_{\Gamma, X <: \mathrm{Top}} \cap [\![U_m]\!]_{\Gamma, X <: \mathrm{Top}} = \{\}$. By

$$\frac{\vdots}{\Gamma \vdash S =:= T}\ (\text{S-Match1/2}) \qquad \frac{\vdots}{\Gamma \vdash S =:= T}\ (\text{S-Match3/4}) \qquad \frac{\Gamma \vdash S \rightleftharpoons U \quad \Gamma \vdash U \rightleftharpoons T}{\Gamma \vdash S \rightleftharpoons T}$$

Fig. 1. Definition of the auxiliary relation $\rightleftharpoons$, used to state inversion of subtyping.

$[\![T_s]\!]_{\Gamma, X <: U} \subset [\![U_n]\!]_{\Gamma, X <: U}$, we get $[\![S_s]\!]_{\Gamma, X <: U} \subset [\![U_n]\!]_{\Gamma, X <: U}$ (by subset transitivity). Therefore, $[\![S]\!]_\Gamma = [\![S_n]\!]_{\Gamma, X <: U}$, and the result follows from the IH.

b. $[\![T]\!]_\Gamma = [\![T_d]\!]_\Gamma$ and $\forall n.\ [\![T_s]\!]_{\Gamma, X <: \text{Top}} \cap [\![U_n]\!]_{\Gamma, X <: \text{Top}} = \{\}$:
The IH gives $[\![S_s]\!]_{\Gamma, X <: \text{Top}} \subset [\![T_s]\!]_{\Gamma, X <: \text{Top}}$ (using Lemma 3.2). Using $\forall n.\ [\![T_s]\!]_{\Gamma, X <: \text{Top}} \cap [\![U_n]\!]_{\Gamma, X <: \text{Top}} = \{\}$ we get $\forall n.\ [\![S_s]\!]_{\Gamma, X <: \text{Top}} \cap [\![U_n]\!]_{\Gamma, X <: \text{Top}} = \{\}$. Therefore, $[\![S]\!]_\Gamma = [\![S_d]\!]_\Gamma$, and the result follows from the IH.

c. $[\![T]\!]_\Gamma = P$:
P is the codomain of $[\![\cdot]\!]_\Gamma$, therefore $[\![S]\!]_\Gamma \subset [\![T]\!]_\Gamma$, as required.

2. ($\Gamma \vdash \text{disj}(S, T)$ implies $[\![S]\!]_\Gamma \cap [\![T]\!]_\Gamma = \{\}$)
   - *Case* D-Xi: $S = C_1 \quad T = C_2 \quad (C_1, C_2) \in \Xi$
     $[\![C_1]\!]_\Gamma = \{c \in C \mid (c, C_1) \in \Psi\}$ and $[\![C_2]\!]_\Gamma = \{c \in C \mid (c, C_2) \in \Psi\}$. By definition of Ξ, there is no class c such that both $(c, C_1) \in \Psi$ and $(c, C_2) \in \Psi$, and $[\![C_1]\!]_\Gamma \cap [\![C_2]\!]_\Gamma = \{\}$.
   - *Case* D-Psi: $S = \{\mathit{new}\ C_1\} \quad T = C_2 \quad (C_1, C_2) \notin \Psi$
     $[\![\{\mathit{new}\ C_1\}]\!]_\Gamma = C_1$ and $[\![C_2]\!]_\Gamma = \{c \in C \mid (c, C_2) \in \Psi\}$. Since $(C_1, C_2) \notin \Psi$, $C_1 \notin [\![C_2]\!]_\Gamma$ and $[\![C_1]\!]_\Gamma \cap [\![C_2]\!]_\Gamma = \{\}$.
   - *Case* D-Sub: $\Gamma \vdash S <: U \quad \Gamma \vdash \text{disj}(U, T)$
     By the IH, $[\![S]\!]_\Gamma \subset [\![U]\!]_\Gamma$ and $[\![U]\!]_\Gamma \cap [\![T]\!]_\Gamma = \{\}$. $[\![S]\!]_\Gamma \cap [\![T]\!]_\Gamma = \{\}$ follows directly.
   - *Case* D-Arrow: $S = S_1 \to S_2 \quad T = C_1$
     $[\![C_1]\!]_\Gamma = \{c \in C \mid (c, C_1) \in \Psi\}$, $[\![S_1 \to S_2]\!]_\Gamma = \{\Lambda\}$ and $[\![C_1]\!]_\Gamma \cap [\![S_1 \to S_2]\!]_\Gamma = \{\}$, as required.
   - *Case* D-All: $S = \forall X <: S_1.\ S_2 \quad T = C_1$
     $[\![C_1]\!]_\Gamma = \{c \in C \mid (c, C_1) \in \Psi\}$, $[\![\forall X <: S_1.\ S_2]\!]_\Gamma = \{S\}$ and $[\![C_1]\!]_\Gamma \cap [\![\forall X <: S_1.\ S_2]\!]_\Gamma = \{\}$, as required.

□

Definition. $\Gamma \vdash S \rightleftharpoons T$ *(defined in Figure 1) represents evidence of the mutual subtyping between a match type* S *and a type* T *with the additional constraint that this evidence was exclusively constructed using pairwise applications of S-Match1/2, S-Match3/4, and S-Trans in both directions.*

Lemma 3.7 (Inversion of subtyping).

1. *If* $\Gamma \vdash S_s$ *match*$[X]\{U_i \Rightarrow S_i\}$ *or* $S_d \rightleftharpoons T$*, then either:*
   a. $\Gamma, X <: U \vdash S_s <: U_n$, $\forall m < n.\ \Gamma, X <: \text{Top} \vdash \text{disj}(S_s, U_m)$ *and* $[X \mapsto U]S_n$ *is a match type with* $\Gamma \vdash [X \mapsto U]S_n \rightleftharpoons T$*,*
   b. $\Gamma, X <: U \vdash S_s <: U_n$, $\forall m < n.\ \Gamma, X <: \text{Top} \vdash \text{disj}(S_s, U_m)$ *and* $[X \mapsto U]S_n = T$*,*
   c. $\forall n.\ \Gamma, X <: \text{Top} \vdash \text{disj}(S_s, U_n)$ *and* S_d *is a match type with* $\Gamma \vdash S_d \rightleftharpoons T$*,*
   d. $\forall n.\ \Gamma, X <: \text{Top} \vdash \text{disj}(S_s, U_n)$ *and* $S_d = T$*.*
2. *If* $\Gamma \vdash S <: X$*, or* $\Gamma \vdash S <: T$ *where* T *is a match type with* $\Gamma \vdash T \rightleftharpoons X$*, then either*
   a. S *is a match type with* $\Gamma \vdash S \rightleftharpoons Y$*, for some* Y*,*
   b. S *is a type variable.*
3. *If* $\Gamma \vdash S <: T_1 \to T_2$*, or* $\Gamma \vdash S <: T$ *where* T *is a match type with* $\Gamma \vdash T \rightleftharpoons T_1 \to T_2$*, then either*
   a. S *is a match type with* $\Gamma \vdash S \rightleftharpoons S_1 \to S_2$ *for some* S_1, S_2 *such that* $\Gamma \vdash T_1 <: S_1$ *and* $\Gamma \vdash S_2 <: T_2$*,*

   b. S *is a match type with* $\Gamma \vdash S \rightleftharpoons X$, *for some* X,
   c. S *is a type variable,*
   d. S *has the form* $S_1 \to S_2$ *with* $\Gamma \vdash T_1 <: S_1$ *and* $\Gamma \vdash S_2 <: T_2$.
4. *If* $\Gamma \vdash S <: \forall X <: U_1.\ T_2$, *or* $\Gamma \vdash S <: T$ *where* T *is a match type with* $\Gamma \vdash T \rightleftharpoons \forall X <: U_1.\ T_2$, *then either*
   a. S *is a match type with* $\Gamma \vdash S \rightleftharpoons \forall X <: U_1.\ S_2$ *for some* S_2 *such that* $\Gamma, X <: U_1 \vdash S_2 <: T_2$,
   b. S *is a match type with* $\Gamma \vdash S \rightleftharpoons X$, *for some* X,
   c. S *is a type variable,*
   d. S *has the form* $\forall X <: U_1.\ S_2$ *and* $\Gamma, X <: U_1 \vdash S_2 <: T_2$.

Proof:

1. By induction on a derivation on $\Gamma \vdash S \rightleftharpoons T$.
   - *Case* BS-Match1/2: $S = S_s$ *match*$[X]\{U_i \Rightarrow S_i\}$*or* S_d $\quad T = [X \mapsto U]S_n$
     $\Gamma, X <: U \vdash S_s <: U_n$
     $\forall m < n.\ \Gamma, X <: \mathrm{Top} \vdash \mathrm{disj}(S_s, U_m)$
     $[X \mapsto U]S_n = T$ and the result follows directly from the premises.
   - *Case* BS-Match3/4: $S = S_s$ *match*$[X]\{U_i \Rightarrow S_i\}$*or* S_d $\quad T = S_d$
     $\forall n.\ \Gamma, X <: \mathrm{Top} \vdash \mathrm{disj}(S_s, U_n)$
     $S_d = T$ and the result follows directly from the premises.
   - *Case* S-Trans: $\quad \Gamma \vdash S \rightleftharpoons U \quad \Gamma \vdash U \rightleftharpoons T$
     By definition of the $\rightleftharpoons$ relation, $\Gamma \vdash U \rightleftharpoons T$ implies that U is a match type. Using the IH on the left premise we get 4 subcases:
     a. $\Gamma, X <: V \vdash S_s <: U_n$, $\forall m < n.\ \Gamma, X <: \mathrm{Top} \vdash \mathrm{disj}(S_s, U_m)$ and $[X \mapsto V]S_n$ is a match type with $\Gamma \vdash [X \mapsto V]S_n \rightleftharpoons U$:
        Using S-Trans we get $\Gamma \vdash [X \mapsto V]S_n \rightleftharpoons T$, as requiered.
     b. $\Gamma, X <: V \ \vdash S_s <: U_n$, $\forall m < n.\ \Gamma, X <: \mathrm{Top} \vdash \mathrm{disj}(S_s, U_m)$ and $[X \mapsto V]S_n = U$:
        The result is immediate.
     c. $\forall n.\ \Gamma, X <: \mathrm{Top} \vdash \mathrm{disj}(S_s, U_n)$ and S_d is a match type with $\Gamma \vdash S_d \rightleftharpoons U$:
        Using S-Trans we get $\Gamma \vdash S_d \rightleftharpoons T$, as requiered.
     d. $\forall n.\ \Gamma, X <: \mathrm{Top} \vdash \mathrm{disj}(S_s, U_n)$ and $S_d = U$:
        The result is immediate.
2. By induction on a derivation of $\Gamma \vdash S <: T$.
   - *Case* S-Refl: $\quad S = T$
     If T is a type variable then so is S and the result is immediate. If T is a match type with $\Gamma \vdash T \rightleftharpoons X$, S is a match type and $\Gamma \vdash S \rightleftharpoons Y$, as required.
   - *Case* S-Trans: $\quad \Gamma \vdash S <: U \quad \Gamma \vdash U <: T$
     If T is a type variable we use the IH on the right premise to get that U is either a match type with $\Gamma \vdash U \rightleftharpoons X$ or a type variable. In either case, the result follows from using the IH on the left premise.
     If T is a match type with $\Gamma \vdash T \rightleftharpoons X$ we can also use the the IH on the right premise to get to the same result.
   - *Case* S-TVar: $\quad S = Y \quad Y <: X \in \Gamma$
     S is a type variable and the result is immediate.
   - *Case* BS-Match1: $S = T_s$ *match*$[Y]\{U_i \Rightarrow T_i\}$*or* T_d $\quad T = [Y \mapsto U]T_n$
     $\Gamma, Y <: U \vdash T_s <: U_n$
     $\forall m < n.\ \Gamma, Y <: \mathrm{Top} \vdash \mathrm{disj}(T_s, U_m)$

If $T = X$, we use BS-Match2 to obtain $\Gamma \vdash S \rightleftharpoons X$, as required. If T is a match type with $\Gamma \vdash T \rightleftharpoons X$, we use BS-Match2 to get $\Gamma \vdash S \rightleftharpoons T$, and S-Trans to obtain $\Gamma \vdash S \rightleftharpoons X$, as required.

- *Case* BS-Match2: $S = [Y \mapsto U]T_n \quad T = T_s\ \mathit{match}[Y]\{U_i \Rightarrow T_i\}\mathit{or}\ T_d$
  $\Gamma, Y <: U \vdash T_s <: U_n$
  $\forall m < n.\ \Gamma, Y <: \mathrm{Top} \vdash \mathrm{disj}(T_s, U_m)$

  In this case the first premise of the statement ($\Gamma \vdash S <: X$) does not apply since T is a match type. Using the 1st part of the lemma on $\Gamma \vdash T \rightleftharpoons X$, we get 4 subcases:
  a. $\Gamma, Y <: U \vdash T_s <: U_n$, $\forall m < n.\ \Gamma, Y <: \mathrm{Top} \vdash \mathrm{disj}(T_s, U_m)$ and $[Y \mapsto U]T_n$ is a match type with $\Gamma \vdash [Y \mapsto U]T_n \rightleftharpoons X$:
     $S = [Y \mapsto U]T_n$ is a match type and $\Gamma \vdash S \rightleftharpoons X$, as required.
  b. $\Gamma, Y <: U \vdash T_s <: U_n$, $\forall m < n.\ \Gamma, Y <: \mathrm{Top} \vdash \mathrm{disj}(T_s, U_m)$ and $[Y \mapsto U]T_n = X$:
     $S = [Y \mapsto U]T_n = X$ is a type variable and the result is immediate.
  c. $\forall m.\ \Gamma, Y <: \mathrm{Top} \vdash \mathrm{disj}(T_s, U_m)$ and T_d is a match type with $\Gamma \vdash T_d \rightleftharpoons X$:
     This case cannot occur since $\Gamma, Y <: \mathrm{Top} \vdash \mathrm{disj}(T_s, U_n)$ implies $\Gamma, Y <: U \vdash \mathrm{disj}(T_s, U_n)$ (using Lemma 3.4) which contradicts $\Gamma, Y <: U \vdash T_s <: U_n$ (Lemma 3.6).
  d. $\forall m.\ \Gamma, Y <: \mathrm{Top} \vdash \mathrm{disj}(T_s, U_m)$ and $T_d = X$:
     This case cannot occur since $\Gamma, Y <: \mathrm{Top} \vdash \mathrm{disj}(T_s, U_n)$ implies $\Gamma, Y <: U \vdash \mathrm{disj}(T_s, U_n)$ (using Lemma 3.4) which contradicts $\Gamma, Y <: U \vdash T_s <: U_n$ (Lemma 3.6).
- *Case* BS-Match3: $S = T_s\ \mathit{match}[Y]\{U_i \Rightarrow T_i\}\mathit{or}\ T_d \quad T = T_d$
  $\forall n.\ \Gamma, Y <: \mathrm{Top} \vdash \mathrm{disj}(T_s, U_n)$

  If $T = X$, we use BS-Match4 to obtain $\Gamma \vdash S \rightleftharpoons X$, as required. If T is a match type with $\Gamma \vdash T \rightleftharpoons X$, we use BS-Match4 to get $\Gamma \vdash S \rightleftharpoons T$, and S-Trans to obtain $\Gamma \vdash S \rightleftharpoons X$, as required.
- *Case* BS-Match4: $S = T_d \quad T = T_s\ \mathit{match}[Y]\{U_i \Rightarrow T_i\}\mathit{or}\ T_d$
  $\forall n.\ \Gamma, Y <: \mathrm{Top} \vdash \mathrm{disj}(T_s, U_n)$

  In this case the first premise of the statement ($\Gamma \vdash S <: X$) does not apply since T is a match type. Using the 1st part of the lemma on $\Gamma \vdash T \rightleftharpoons X$, we get 4 subcases:
  a. $\Gamma, Y <: U \vdash T_s <: U_n$, $\forall m < n.\ \Gamma, Y <: \mathrm{Top} \vdash \mathrm{disj}(T_s, U_m)$ and $[Y \mapsto U]T_n$ is a match type with $\Gamma \vdash [Y \mapsto U]T_n \rightleftharpoons X$:
     This case cannot occur since $\Gamma, Y <: \mathrm{Top} \vdash \mathrm{disj}(T_s, U_n)$ implies $\Gamma, Y <: U \vdash \mathrm{disj}(T_s, U_n)$ (using Lemma 3.4) which contradicts $\Gamma, Y <: U \vdash T_s <: U_n$ (Lemma 3.6).
  b. $\Gamma, Y <: U \vdash T_s <: U_n$, $\forall m < n.\ \Gamma, Y <: \mathrm{Top} \vdash \mathrm{disj}(T_s, U_m)$ and $[Y \mapsto U]T_n = X$:
     This case cannot occur since $\Gamma, Y <: \mathrm{Top} \vdash \mathrm{disj}(T_s, U_n)$ implies $\Gamma, Y <: U \vdash \mathrm{disj}(T_s, U_n)$ (using Lemma 3.4) which contradicts $\Gamma, Y <: U \vdash T_s <: U_n$ (Lemma 3.6).
  c. $\forall m.\ \Gamma, Y <: \mathrm{Top} \vdash \mathrm{disj}(T_s, U_m)$ and T_d is a match type with $\Gamma \vdash T_d \rightleftharpoons X$:
     S is a match type and $\Gamma \vdash S \rightleftharpoons X$, as required.
  d. $\forall m.\ \Gamma, Y <: \mathrm{Top} \vdash \mathrm{disj}(T_s, U_m)$ and $T_d = X$:
     $S = T_d = X$ is a type variable and the result is immediate.
- *Case* BS-Match5: $S = S_s\ \mathit{match}[Y]\{U_i \Rightarrow S_i\}\mathit{or}\ S_d \quad T = T_s\ \mathit{match}[Y]\{U_i \Rightarrow T_i\}\mathit{or}\ T_d$
  $\Gamma \vdash S_s <: T_s$
  $\forall m.\ \Gamma, Y <: \mathrm{Top} \vdash S_m <: T_m$
  $\Gamma \vdash S_d <: T_d$

  In this case the first premise of the statement ($\Gamma \vdash S <: X$) does not apply since T is a match type. Using the 1st part of the lemma on $\Gamma \vdash T \rightleftharpoons X$, we get 4 subcases:
  a. $\Gamma, Y <: U \vdash T_s <: U_n$, $\forall m < n.\ \Gamma, Y <: \mathrm{Top} \vdash \mathrm{disj}(T_s, U_m)$ and $[Y \mapsto U]T_n$ is a match type with $\Gamma \vdash [Y \mapsto U]T_n \rightleftharpoons X$:

Using D-Sub on $\forall m.\ \Gamma, Y <: Top \vdash S_m <: T_m$ and $\forall m < n.\ \Gamma, Y <: Top \vdash disj(T_s, U_m)$ we obtain $\forall m < n.\ \Gamma, Y <: Top \vdash disj(S_s, U_m)$. From S-Trans we get $\Gamma, Y <: Top \vdash S_s <: U_n$. Using BS-Match1/2 we obtain $\Gamma \vdash S \rightleftharpoons [Y \mapsto U]S_n$. Since $\Gamma \vdash [Y \mapsto U]S_n <: T_n$ and $\Gamma \vdash [Y \mapsto U]T_n \rightleftharpoons X$, we use the IH to get that either $[Y \mapsto U]S_n$ is a match type with $\Gamma \vdash [Y \mapsto U]S_n \rightleftharpoons Z$, or $[Y \mapsto U]S_n$ is a type variable. If $[Y \mapsto U]S_n$ is a type variable, we already proved $\Gamma \vdash S \rightleftharpoons [Y \mapsto U]S_n$, as required. If $\Gamma \vdash [Y \mapsto U]S_n \rightleftharpoons Y$, then using S-Trans with $\Gamma \vdash S \rightleftharpoons [Y \mapsto U]S_n$ gives $\Gamma \vdash S \rightleftharpoons Y$, as required.

b. $\Gamma, Y <: U \vdash T_s <: U_n$, $\forall m < n.\ \Gamma, Y <: Top \vdash disj(T_s, U_m)$ and $[Y \mapsto U]T_n = X$:
Using D-Sub on $\forall m.\ \Gamma, Y <: Top \vdash S_m <: T_m$ and $\forall m < n.\ \Gamma, Y <: Top \vdash disj(T_s, U_m)$ we obtain $\forall m < n.\ \Gamma, Y <: Top \vdash disj(S_s, U_m)$. From S-Trans we get $\Gamma, Y <: Top \vdash S_s <: U_n$. Using BS-Match1/2 we obtain $\Gamma \vdash S \rightleftharpoons [Y \mapsto U]S_n$. Since $\Gamma \vdash [Y \mapsto U]S_n <: X$, we use the IH to get that either $[Y \mapsto U]S_n$ is a match type with $\Gamma \vdash [Y \mapsto U]S_n \rightleftharpoons Y$, or $[Y \mapsto U]S_n$ is a type variable. If $[Y \mapsto U]S_n$ is a type variable, we already proved $\Gamma \vdash S \rightleftharpoons [Y \mapsto U]S_n$, as required. If $\Gamma \vdash [Y \mapsto U]S_n \rightleftharpoons Y$, then using S-Trans with $\Gamma \vdash S \rightleftharpoons [Y \mapsto U]S_n$ gives $\Gamma \vdash S \rightleftharpoons Y$, as required.

c. $\forall m.\ \Gamma, Y <: Top \vdash disj(T_s, U_m)$ and T_d is a match type with $\Gamma \vdash T_d \rightleftharpoons X$:
Using D-Sub on $\forall m.\ \Gamma, Y <: Top \vdash S_m <: T_m$ and $\forall m.\ \Gamma, Y <: Top \vdash disj(T_s, U_m)$ we obtain $\forall m.\ \Gamma, Y <: Top \vdash disj(S_s, U_m)$. Using BS-Match3/4 we obtain $\Gamma \vdash S \rightleftharpoons S_d$. Since $\Gamma \vdash S_d <: T_d$ and $\Gamma \vdash T_d \rightleftharpoons X$, we use the IH to get that either S_d is a match type with $\Gamma \vdash S_d \rightleftharpoons Y$, or S_d is a type variable. If S_d is a type variable, we already proved $\Gamma \vdash S \rightleftharpoons S_d$, as required. If $\Gamma \vdash S_d \rightleftharpoons Y$, then using S-Trans with $\Gamma \vdash S \rightleftharpoons S_d$ gives $\Gamma \vdash S \rightleftharpoons Y$, as required.

d. $\forall m.\ \Gamma, Y <: Top \vdash disj(T_s, U_m)$ and $T_d = X$:
Using D-Sub on $\forall m.\ \Gamma, Y <: Top \vdash S_m <: T_m$ and $\forall m.\ \Gamma, Y <: Top \vdash disj(T_s, U_m)$ we obtain $\forall m.\ \Gamma, Y <: Top \vdash disj(S_s, U_m)$. Using BS-Match3/4 we obtain $\Gamma \vdash S \rightleftharpoons S_d$. Since $\Gamma \vdash S_d <: T_d$ and $T_d = X$, we use the IH to get that either S_d is a match type with $\Gamma \vdash S_d \rightleftharpoons Y$, or S_d is a type variable. If S_d is a type variable, we already proved $\Gamma \vdash S \rightleftharpoons S_d$, as required. If $\Gamma \vdash S_d \rightleftharpoons Y$, then using S-Trans with $\Gamma \vdash S \rightleftharpoons S_d$ gives $\Gamma \vdash S \rightleftharpoons Y$, as required.

– *Case* S-Top, S-Sin, S-Arrow, S-All, S-Psi:
In those cases, T is neither a type variable nor a match type and the result is immediate.

3. By induction on a derivation of $\Gamma \vdash S <: T$.
– *Case* S-Refl: $T = S$
If T is a match type with $\Gamma \vdash T \rightleftharpoons T_1 \to T_2$, the result follows directly from S-Refl. If $T = T_1 \to T_2$, the result also follows directly from S-Refl.
– *Case* S-Trans: $\Gamma \vdash S <: U$ $\Gamma \vdash U <: T$
Using the IH on the right premise we get 4 subcases:
a. U is a match type with $\Gamma \vdash U \rightleftharpoons U_1 \to U_2$, for some U_1, U_2 such that $\Gamma \vdash T_1 <: U_1$ and $\Gamma \vdash U_2 <: T_2$:
The result follows directly from using the IH on the left premise ($\Gamma \vdash T_1 <: S_1$ is obtained using S-Trans with $\Gamma \vdash T_1 <: U_1$ and $\Gamma \vdash U_1 <: S_1$, $\Gamma \vdash S_2 <: T_2$ is obtained analogously).
b. U is a match type with $\Gamma \vdash U \rightleftharpoons X$, for some X:
Using the 2nd part of the lemma on the left premise leads to the desired result.
c. U is a type variable:
The result follows from using the 2nd part of the lemma on the left premise.
d. U has the form $U_1 \to U_2$ with $\Gamma \vdash T_1 <: U_1$ and $\Gamma \vdash U_2 <: T_2$:

The result follows directly from using the IH on the left premise ($\Gamma \vdash T_1 <: S_1$ is obtained using S-Trans with $\Gamma \vdash T_1 <: U_1$ and $\Gamma \vdash U_1 <: S_1$, $\Gamma \vdash S_2 <: T_2$ is obtained analogously).

- *Case* BS-Match1: $S = T_s[Z]\ \mathit{match}\{U_i \Rightarrow T_i\}\mathit{or}\ T_d \quad T = [Z \mapsto U]T_n$
  $\Gamma, Z <: U \vdash T_s <: U_n$
  $\forall m < n.\ \Gamma, Z <: \mathrm{Top} \vdash \mathrm{disj}(T_s, U_m)$

  If $T = T_1 \to T_2$, we use BS-Match2 to obtain $\Gamma \vdash S \rightleftharpoons T_1 \to T_2$, as required. If T is a match type with $\Gamma \vdash T \rightleftharpoons T_1 \to T_2$, we use BS-Match2 to get $\Gamma \vdash S \rightleftharpoons T$, and S-Trans to obtain $\Gamma \vdash S \rightleftharpoons T_1 \to T_2$, and the result follows from S-Refl.
- *Case* BS-Match2: $S = [Z \mapsto U]T_n \quad T = T_s\ \mathit{match}[Z]\{U_i \Rightarrow T_i\}\mathit{or}\ T_d$
  $\Gamma, Z <: U \vdash T_s <: U_n$
  $\forall m < n.\ \Gamma, Z <: \mathrm{Top} \vdash \mathrm{disj}(T_s, U_m)$

  In this case the first premise of the statement ($\Gamma \vdash S <: T_1 \to T_2$) does not apply since T is a match type. Using the 1st part of the lemma on $\Gamma \vdash T \rightleftharpoons T_1 \to T_2$, we get 4 subcases:
  a. $\Gamma, Z <: U \vdash T_s <: U_n$, $\forall m < n.\ \Gamma, Z <: \mathrm{Top} \vdash \mathrm{disj}(T_s, U_m)$ and $[Z \mapsto U]T_n$ is a match type with $\Gamma \vdash [Z \mapsto U]T_n \rightleftharpoons T_1 \to T_2$:
     $S = [Z \mapsto U]T_n$ is a match type and $\Gamma \vdash S \rightleftharpoons T_1 \to T_2$, and the result follows from S-Refl.
  b. $\Gamma, Z <: U \vdash T_s <: U_n$, $\forall m < n.\ \Gamma, Z <: \mathrm{Top} \vdash \mathrm{disj}(T_s, U_m)$ and $[Z \mapsto U]T_n = T_1 \to T_2$:
     $S = T_1 \to T_2$, $S_1 = T_1$, $S_2 = T_2$, and the result follows from S-Refl.
  c. $\forall m.\ \Gamma, Z <: \mathrm{Top} \vdash \mathrm{disj}(T_s, U_m)$ and T_d is a match type with $\Gamma \vdash T_d \rightleftharpoons T_1 \to T_2$:
     This case cannot occur since $\Gamma, Z <: \mathrm{Top} \vdash \mathrm{disj}(T_s, U_n)$ contradicts $\Gamma, Z <: U \vdash T_s <: U_n$ (Lemma 3.4 and Lemma 3.6).
  d. $\forall m.\ \Gamma, Z <: \mathrm{Top} \vdash \mathrm{disj}(T_s, U_m)$ and $T_d = T_1 \to T_2$:
     This case cannot occur since $\Gamma, Z <: \mathrm{Top} \vdash \mathrm{disj}(T_s, U_n)$ contradicts $\Gamma, Z <: U \vdash T_s <: U_n$ (Lemma 3.4 and Lemma 3.6).
- *Case* BS-Match3: $S = T_s\ \mathit{match}[Z]\{U_i \Rightarrow T_i\}\mathit{or}\ T_d \quad T = T_d$
  $\forall n.\ \Gamma, Z <: \mathrm{Top}\ \vdash \mathrm{disj}(T_s, U_n)$

  If $T = T_1 \to T_2$, we use BS-Match2 to obtain $\Gamma \vdash S \rightleftharpoons T_1 \to T_2$, and the result follows from S-Refl. If T is a match type with $\Gamma \vdash T \rightleftharpoons T_1 \to T_2$, we use BS-Match2 to get $\Gamma \vdash S \rightleftharpoons T$, and S-Trans to obtain $\Gamma \vdash S \rightleftharpoons T_1 \to T_2$, and the result follows from S-Refl.
- *Case* BS-Match4: $S = T_d \quad T = T_s\ \mathit{match}[Z]\{U_i \Rightarrow T_i\}\mathit{or}\ T_d$
  $\forall n.\ \Gamma, Z <: \mathrm{Top} \vdash \mathrm{disj}(T_s, U_n)$

  In this case the first premise of the statement ($\Gamma \vdash S <: T_1 \to T_2$) does not apply since T is a match type. Using the 1st part of the lemma on $\Gamma \vdash T \rightleftharpoons T_1 \to T_2$, we get 4 subcases:
  a. $\Gamma, Z <: U \vdash T_s <: U_n$, $\forall m < n.\ \Gamma, Z <: \mathrm{Top} \vdash \mathrm{disj}(T_s, U_m)$ and T_n is a match type with $\Gamma \vdash T_n \rightleftharpoons T_1 \to T_2$:
     This case cannot occur since $\Gamma, Z <: \mathrm{Top} \vdash \mathrm{disj}(T_s, U_n)$ contradicts $\Gamma, Z <: U \vdash T_s <: U_n$ (Lemma 3.4 and Lemma 3.6).
  b. $\Gamma, Z <: U \vdash T_s <: U_n$, $\forall m < n.\ \Gamma, Z <: \mathrm{Top} \vdash \mathrm{disj}(T_s, U_m)$ and $T_n = T_1 \to T_2$:
     This case cannot occur since $\Gamma, Z <: \mathrm{Top} \vdash \mathrm{disj}(T_s, U_n)$ contradicts $\Gamma, Z <: U \vdash T_s <: U_n$ (Lemma 3.4 and Lemma 3.6).
  c. $\forall m.\ \Gamma, Z <: \mathrm{Top} \vdash \mathrm{disj}(T_s, U_m)$ and T_d is a match type with $\Gamma \vdash T_d \rightleftharpoons T_1 \to T_2$:
     $S = T_d$ is a match type and $\Gamma \vdash S \rightleftharpoons T_1 \to T_2$, and the result follows from S-Refl.
  d. $\forall m.\ \Gamma, Z <: \mathrm{Top} \vdash \mathrm{disj}(T_s, U_m)$ and $T_d = T_1 \to T_2$:
     $S = T_1 \to T_2$, $S_1 = T_1$, $S_2 = T_2$, and the result follows from S-Refl.

– *Case* BS-Match5: $S = S_s[Z]\ match\{U_i \Rightarrow S_i\}\ or\ S_d \quad T = T_s[Z]\ match\{U_i \Rightarrow T_i\}\ or\ T_d$
$\Gamma \vdash S_s <: T_s$
$\forall n.\ \Gamma, Z <: Top \vdash S_n <: T_n$
$\Gamma \vdash S_d <: T_d$

In this case the first premise of the statement ($\Gamma \vdash S <: T_1 \to T_2$) does not apply since T is a match type. Using the 1st part of the lemma on $\Gamma \vdash T \rightleftharpoons T_1 \to T_2$, we get 4 subcases:

a. $\Gamma, Z <: U \vdash T_s <: U_n$, $\forall m < n.\ \Gamma, Z <: Top \vdash disj(T_s, U_m)$ and $[Z \mapsto U]T_n$ is a match type with $\Gamma \vdash [Z \mapsto U]T_n \rightleftharpoons T_1 \to T_2$:
Using D-Sub on $\forall m.\ \Gamma, Z <: Top \vdash S_m <: T_m$ and $\forall m < n.\ \Gamma, Z <: Top \vdash disj(T_s, U_m)$ we obtain $\forall m < n.\ \Gamma, Z <: Top \vdash disj(S_s, U_m)$. From Lemma 3.2 and S-Trans we get $\Gamma, Z <: U \vdash S_s <: U_n$. Using BS-Match1/2 we obtain $\Gamma \vdash S \rightleftharpoons [Z \mapsto U]S_n$. Since $\Gamma \vdash [Z \mapsto U]S_n <: [Z \mapsto U]T_n$ and $\Gamma \vdash [Z \mapsto U]T_n \rightleftharpoons T_1 \to T_2$, the IH gives 4 subsubcases:
   (i) $[Z \mapsto U]S_n$ is a match type with $\Gamma \vdash [Z \mapsto U]S_n \rightleftharpoons S_1 \to S_2$ for some S_1, S_2 such that $\Gamma \vdash T_1 <: S_1$ and $\Gamma \vdash S_2 <: T_2$:
   The result follows directly from S-Trans.
   (ii) $[Z \mapsto U]S_n$ is a match type with $\Gamma \vdash [Z \mapsto U]S_n \rightleftharpoons X$:
   The result follows directly from S-Trans.
   (iii) $[Z \mapsto U]S_n$ is a type variable:
   We already proved $\Gamma \vdash S \rightleftharpoons S_n$, as required.
   (iv) $[Z \mapsto U]S_n$ has the form $S_1 \to S_2$ with $\Gamma \vdash T_1 <: S_1$ and $\Gamma \vdash S_2 <: T_2$:
   We already proved $\Gamma \vdash S \rightleftharpoons [Z \mapsto U]S_n$, as required.

b. $\Gamma, Z <: U \vdash T_s <: U_n$, $\forall m < n.\ \Gamma, Z <: Top \vdash disj(T_s, U_m)$ and $[Z \mapsto U]T_n = T_1 \to T_2$:
Using D-Sub on $\forall m.\ \Gamma, Z <: Top \vdash S_m <: T_m$ and $\forall m < n.\ \Gamma, Z <: Top \vdash disj(T_s, U_m)$ we obtain $\forall m < n.\ \Gamma, Z <: Top \vdash disj(S_s, U_m)$. From Lemma 3.2 and S-Trans we get $\Gamma, Z <: U \vdash S_s <: U_n$. Using BS-Match1/2 we obtain $\Gamma \vdash S \rightleftharpoons [Z \mapsto U]S_n$. Since $\Gamma \vdash [Z \mapsto U]S_n <: [Z \mapsto U]T_n$, the IH gives 4 subsubcases:
   (i) $[Z \mapsto U]S_n$ is a match type with $\Gamma \vdash [Z \mapsto U]S_n \rightleftharpoons S_1 \to S_2$, for some S_1, S_2 such that $\Gamma \vdash T_1 <: S_1$ and $\Gamma \vdash S_2 <: T_2$:
   The result follows directly from S-Trans.
   (ii) $[Z \mapsto U]S_n$ is a match type with $\Gamma \vdash [Z \mapsto U]S_n \rightleftharpoons X$:
   The result follows directly from S-Trans.
   (iii) $[Z \mapsto U]S_n$ is a type variable:
   We already proved $\Gamma \vdash S \rightleftharpoons [Z \mapsto U]S_n$, as required.
   (iv) $[Z \mapsto U]S_n$ has the form $S_1 \to S_2$ with $\Gamma \vdash T_1 <: S_1$ and $\Gamma \vdash S_2 <: T_2$:
   We already proved $\Gamma \vdash S \rightleftharpoons [Z \mapsto U]S_n$, as required.

c. $\forall n.\ \Gamma, Z <: Top \vdash disj(T_s, U_n)$ and T_d is a match type with $\Gamma \vdash T_d \rightleftharpoons T_1 \to T_2$:
Using D-Sub on $\forall n.\ \Gamma, Z <: Top \vdash S_n <: T_n$ and $\forall n.\ \Gamma, Z <: Top \vdash disj(T_s, U_n)$ we obtain $\forall n.\ \Gamma, Z <: Top \vdash disj(S_s, U_n)$. From S-Trans we get $\Gamma \vdash S_s <: C_d$. Using BS-Match3/4 we obtain $\Gamma \vdash S \rightleftharpoons S_d$. Since $\Gamma \vdash S_d <: T_d$ and $\Gamma \vdash T_d \rightleftharpoons T_1 \to T_2$, the IH gives 4 subsubcases:
   (i) S_d is a match type with $\Gamma \vdash S_d \rightleftharpoons S_1 \to S_2$ for some S_1, S_2 such that $\Gamma \vdash T_1 <: S_1$ and $\Gamma \vdash S_2 <: T_2$:
   The result follows directly from S-Trans.
   (ii) S_d is a match type with $\Gamma \vdash S_d \rightleftharpoons X$:
   The result follows directly from S-Trans.
   (iii) S_d is a type variable:

We already proved $\Gamma \vdash S \rightleftharpoons S_d$, as required.

(iv) S_d has the form $S_1 \to S_2$ with $\Gamma \vdash T_1 <: S_1$ and $\Gamma \vdash S_2 <: T_2$:
We already proved $\Gamma \vdash S \rightleftharpoons S_d$, as required.

d. $\forall n.\ \Gamma, Z <: \mathrm{Top} \vdash \mathrm{disj}(T_s, U_n)$ and $T_d = T_1 \to T_2$:
Using D-Sub on $\forall n.\ \Gamma, Z <: \mathrm{Top} \vdash S_n <: T_n$ and $\forall n.\ \Gamma, Z <: \mathrm{Top} \vdash \mathrm{disj}(T_s, U_n)$ we obtain $\forall n.\ \Gamma, Z <: \mathrm{Top} \vdash \mathrm{disj}(S_s, U_n)$. From S-Trans we get $\Gamma \vdash S_s <: C_d$. Using BS-Match3/4 we obtain $\Gamma \vdash S \rightleftharpoons S_d$. Since $\Gamma \vdash S_d <: T_d$, the IH gives 4 subsubcases:

(i) S_d is a match type with $\Gamma \vdash S_d \rightleftharpoons S_1 \to S_2$ for some S_1, S_2 such that $\Gamma \vdash T_1 <: S_1$ and $\Gamma \vdash S_2 <: T_2$:
The result follows directly from S-Trans.

(ii) S_d is a match type with $\Gamma \vdash S_d \rightleftharpoons X$:
The result follows directly from S-Trans.

(iii) S_d is a type variable:
We already proved $\Gamma \vdash S \rightleftharpoons S_d$, as required.

(iv) S_d has the form $S_1 \to S_2$ with $\Gamma \vdash T_1 <: S_1$ and $\Gamma \vdash S_2 <: T_2$:
We already proved $\Gamma \vdash S \rightleftharpoons S_d$, as required.

– *Case* S-TVar: $S = Y \quad Y <: T \in \Gamma$
S is a type variable and the result is immediate.

– *Case* S-Arrow: $S = S_1 \to S_2 \quad T = T_1 \to T_2 \quad \Gamma \vdash T_1 <: S_1 \quad \Gamma \vdash S_2 <: T_2$
S has the form $S_1 \to S_2$, with $\Gamma \vdash T_1 <: S_1$ and $\Gamma \vdash S_2 <: T_2$, as required.

– *Case* S-Top, S-Sin, S-All, S-Psi:
In those cases, T is neither a type variable nor a function type and the result is immediate.

4. By induction on a derivation of $\Gamma \vdash S <: T$.

– *Case* S-Refl: $T = S$
If T is a match type with $\Gamma \vdash T \rightleftharpoons \forall X <: U_1.\ T_2$ the result follows directly from S-Refl. If $T = \forall X <: U_1.\ T_2$, $S_2 = T_1$, and the result also follows from S-Refl.

– *Case* S-Trans: $\Gamma \vdash S <: U \quad \Gamma \vdash U <: T$
Using the IH on the right premise we get 4 subcases:

a. U is a match type with $\Gamma \vdash U \rightleftharpoons \forall X <: U_1.\ U_2$ for some U_2 such that $\Gamma, X <: U_1 \vdash U_2 <: T_2$:
The result follows directly from using the IH on the left premise ($\Gamma, X <: U_1 \vdash S_2 <: T_2$ is obtained using S-Trans with $\Gamma, X <: U_1 \vdash S_2 <: U_2$ and $\Gamma, X <: U_1 \vdash U_2 <: T_2$).

b. U is a match type with $\Gamma \vdash U \rightleftharpoons X$, for some X:
Using the 2nd part of the lemma on the left premise leads to the desired result.

c. U is a type variable:
The result follows from using the 2nd part of the lemma on the left premise.

d. U has the form $\forall X <: U_1.\ U_2$ with $\Gamma, X <: U_1 \vdash U_2 <: T_2$:
The result follows directly from using the IH on the left premise ($\Gamma, X <: U_1 \vdash S_2 <: T_2$ is obtained using S-Trans with $\Gamma, X <: U_1 \vdash S_2 <: U_2$ and $\Gamma, X <: U_1 \vdash U_2 <: T_2$).

– *Case* BS-Match1: $S = T_s\ \mathit{match}[Z]\{U_i \Rightarrow T_i\}\ \mathit{or}\ T_d \quad T = [Z \mapsto U]T_n \quad \Gamma, Z <: U \vdash T_s <: U_n \quad \forall m < n.\ \Gamma, Z <: \mathrm{Top} \vdash \mathrm{disj}(T_s, U_m)$
If $T = \forall X <: U_1.\ T_2$, we use BS-Match2 to obtain $\Gamma \vdash S \rightleftharpoons \forall X <: U_1.\ T_2$, as required, and the result follows from S-Refl. If T is a match type with $\Gamma \vdash T \rightleftharpoons \forall X <: U_1.\ T_2$, we use BS-Match2 to get $\Gamma \vdash S \rightleftharpoons T$, and S-Trans to obtain $\Gamma \vdash S \rightleftharpoons \forall X <: U_1.\ T_2$, and the result follows from S-Refl.

- *Case* BS-Match2: $S = [Z \mapsto U]T_n \quad T = T_s\ \mathit{match}[Z]\{U_i \Rightarrow T_i\}\mathit{or}\ T_d$
  $\Gamma, Z <: U \vdash T_s <: U_n$
  $\forall m < n.\ \Gamma, Z <: \mathrm{Top} \vdash \mathrm{disj}(T_s, U_m)$

  In this case the first premise of the statement ($\Gamma \vdash S <: (\forall X <: U_1.\ T_2)$) does not apply since T is a match type. Using the 1st part of the lemma on $\Gamma \vdash T \rightleftharpoons \forall X <: U_1.\ T_2$, we get 4 subcases:
  a. $\Gamma, Z <: U \vdash T_s <: U_n$, $\forall m < n.\ \Gamma, Z <: \mathrm{Top} \vdash \mathrm{disj}(T_s, U_m)$ and $[Z \mapsto U]T_n$ is a match type with $\Gamma \vdash [Z \mapsto U]T_n \rightleftharpoons \forall X <: U_1.\ T_2$:
     $S = [Z \mapsto U]T_n$ is a match type and $\Gamma \vdash S \rightleftharpoons \forall X <: U_1.\ T_2$, and the result follows from S-Refl.
  b. $\Gamma, Z <: U \vdash T_s <: U_n$, $\forall m < n.\ \Gamma, Z <: \mathrm{Top} \vdash \mathrm{disj}(T_s, U_m)$ and $[Z \mapsto U]T_n = \forall X <: U_1.\ T_2$:
     $S = \forall X <: U_1.\ T_2$, $S_2 = T_2$, and the result follows from S-Refl.
  c. $\forall m.\ \Gamma, Z <: \mathrm{Top} \vdash \mathrm{disj}(T_s, U_m)$ and T_d is a match type with $\Gamma \vdash T_d \rightleftharpoons \forall X <: U_1.\ T_2$:
     This case cannot occur since $\Gamma, Z <: \mathrm{Top} \vdash \mathrm{disj}(T_s, U_n)$ contradicts $\Gamma, Z <: U \vdash T_s <: U_n$ (Lemma 3.4 and Lemma 3.6).
  d. $\forall m.\ \Gamma, Z <: \mathrm{Top} \vdash \mathrm{disj}(T_s, U_m)$ and $T_d = \forall X <: U_1.\ T_2$:
     This case cannot occur since $\Gamma, Z <: \mathrm{Top} \vdash \mathrm{disj}(T_s, U_n)$ contradicts $\Gamma, Z <: U \vdash T_s <: U_n$ (Lemma 3.4 and Lemma 3.6).
- *Case* BS-Match3: $S = T_s\ \mathit{match}[Z]\{U_i \Rightarrow T_i\}\mathit{or}\ T_d \quad T = T_d$
  $\forall n.\ \Gamma, Z <: \mathrm{Top} \vdash \mathrm{disj}(T_s, U_n)$

  If $T = \forall X <: U_1.\ T_2$, we use BS-Match2 to obtain $\Gamma \vdash S \rightleftharpoons \forall X <: U_1.\ T_2$, and the result follows from S-Refl. If T is a match type with $\Gamma \vdash T \rightleftharpoons \forall X <: U_1.\ T_2$, we use BS-Match2 to get $\Gamma \vdash S \rightleftharpoons T$, and S-Trans to obtain $\Gamma \vdash S \rightleftharpoons \forall X <: U_1.\ T_2$, and the result follows from S-Refl.
- *Case* BS-Match4: $S = T_d \quad T = T_s\ \mathit{match}[Z]\{U_i \Rightarrow T_i\}\mathit{or}\ T_d$
  $\forall n.\ \Gamma, Z <: \mathrm{Top} \vdash \mathrm{disj}(T_s, U_n)$

  In this case the first premise of the statement ($\Gamma \vdash S <: (\forall X <: U_1.\ T_2)$) does not apply since T is a match type. Using the 1st part of the lemma on $\Gamma \vdash T \rightleftharpoons \forall X <: U_1.\ T_2$, we get 4 subcases:
  a. $\Gamma, Z <: U \vdash T_s <: U_n$, $\forall m < n.\ \Gamma, Z <: \mathrm{Top} \vdash \mathrm{disj}(T_s, U_m)$ and T_n is a match type with $\Gamma \vdash T_n \rightleftharpoons \forall X <: U_1.\ T_2$:
     This case cannot occur since $\Gamma, Z <: \mathrm{Top} \vdash \mathrm{disj}(T_s, U_n)$ contradicts $\Gamma, Z <: U \vdash T_s <: U_n$ (Lemma 3.4 and Lemma 3.6).
  b. $\Gamma, Z <: U \vdash T_s <: U_n$, $\forall m < n.\ \Gamma, Z <: \mathrm{Top} \vdash \mathrm{disj}(T_s, U_m)$ and $T_n = \forall X <: U_1.\ T_2$:
     This case cannot occur since $\Gamma, Z <: \mathrm{Top} \vdash \mathrm{disj}(T_s, U_n)$ contradicts $\Gamma, Z <: U \vdash T_s <: U_n$ (Lemma 3.4 and Lemma 3.6).
  c. $\forall m.\ \Gamma, Z <: \mathrm{Top} \vdash \mathrm{disj}(T_s, U_m)$ and T_d is a match type with $\Gamma \vdash T_d \rightleftharpoons \forall X <: U_1.\ T_2$:
     $S = T_d$ is a match type and $\Gamma \vdash S \rightleftharpoons \forall X <: U_1.\ T_2$, and the result follows from S-Refl.
  d. $\forall m.\ \Gamma, Z <: \mathrm{Top} \vdash \mathrm{disj}(T_s, U_m)$ and $T_d = \forall X <: U_1.\ T_2$:
     $S = \forall X <: U_1.\ T_2$, $S_2 = T_2$, and the result follows from S-Refl.
- *Case* BS-Match5: $S = S_s\ \mathit{match}[Z]\{U_i \Rightarrow S_i\}\mathit{or}\ S_d \quad T = T_s\ \mathit{match}[Z]\{U_i \Rightarrow T_i\}\mathit{or}\ T_d$
  $\Gamma \vdash S_s <: T_s$
  $\forall n.\ \Gamma, Z <: \mathrm{Top} \vdash S_n <: T_n$
  $\Gamma \vdash S_d <: T_d$

In this case the first premise of the statement ($\Gamma \vdash S <: (\forall X <: U_1.\ T_2)$) does not apply since T is a match type. Using the 1st part of the lemma on $\Gamma \vdash T \rightleftharpoons \forall X <: U_1.\ T_2$, we get 4 subcases:

a. $\Gamma, Z <: U \vdash T_s <: U_n$, $\forall m < n.\ \Gamma, Z <: \mathrm{Top} \vdash \mathrm{disj}(T_s, U_m)$ and $[Z \mapsto U]T_n$ is a match type with $\Gamma \vdash [Z \mapsto U]T_n \rightleftharpoons \forall X <: U_1.\ T_2$:
Using D-Sub on $\forall m.\ \Gamma, Z <: \mathrm{Top} \vdash S_m <: T_m$ and $\forall m < n.\ \Gamma, Z <: \mathrm{Top} \vdash \mathrm{disj}(T_s, U_m)$ we obtain $\forall m < n.\ \Gamma, Z <: \mathrm{Top} \vdash \mathrm{disj}(S_s, U_m)$. From Lemma 3.2 and S-Trans we get $\Gamma, Z <: U \vdash S_s <: U_n$. Using BS-Match1/2 we obtain $\Gamma \vdash S \rightleftharpoons [Z \mapsto U]S_n$. Since $\Gamma \vdash [Z \mapsto U]S_n <: [Z \mapsto U]T_n$ and $\Gamma \vdash [Z \mapsto U]T_n \rightleftharpoons \forall X <: U_1.\ T_2$, the IH gives 4 subsubcases:
   (i) $[Z \mapsto U]S_n$ is a match type with $\Gamma \vdash [Z \mapsto U]S_n \rightleftharpoons \forall X <: U_1.\ S_2$ for some S_2 such that $\Gamma, X <: U_1 \vdash S_2 <: T_2$:
   The result follows directly from S-Trans.
   (ii) $[Z \mapsto U]S_n$ is a match type with $\Gamma \vdash [Z \mapsto U]S_n \rightleftharpoons X$:
   The result follows directly from S-Trans.
   (iii) $[Z \mapsto U]S_n$ is a type variable:
   We already proved $\Gamma \vdash S \rightleftharpoons [Z \mapsto U]S_n$, as required.
   (iv) $[Z \mapsto U]S_n$ has the form $\forall X <: U_1.\ S_2$ with $\Gamma, X <: U_1 \vdash S_2 <: T_2$:
   We already proved $\Gamma \vdash S \rightleftharpoons [Z \mapsto U]S_n$, as required.

b. $\Gamma, Z <: U \vdash T_s <: U_n$, $\forall m < n.\ \Gamma, Z <: \mathrm{Top} \vdash \mathrm{disj}(T_s, U_m)$ and $T_n = \forall X <: U_1.\ T_2$:
Using D-Sub on $\forall m.\ \Gamma, Z <: \mathrm{Top} \vdash S_m <: T_m$ and $\forall m < n.\ \Gamma, Z <: \mathrm{Top} \vdash \mathrm{disj}(T_s, U_m)$ we obtain $\forall m < n.\ \Gamma, Z <: \mathrm{Top} \vdash \mathrm{disj}(S_s, U_m)$. From Lemma 3.2 and S-Trans we get $\Gamma, Z <: U \vdash S_s <: U_n$. Using BS-Match1/2 we obtain $\Gamma \vdash S \rightleftharpoons [Z \mapsto U]S_n$. Since $\Gamma \vdash [Z \mapsto U]S_n <: [Z \mapsto U]T_n$, the IH, the IH gives 4 subsubcases:
   (i) $[Z \mapsto U]S_n$ is a match type with $\Gamma \vdash [Z \mapsto U]S_n \rightleftharpoons \forall X <: U_1.\ S_2$ for some S_2 such that $\Gamma, X <: U_1 \vdash S_2 <: T_2$:
   The result follows directly from S-Trans.
   (ii) $[Z \mapsto U]S_n$ is a match type with $\Gamma \vdash [Z \mapsto U]S_n \rightleftharpoons X$:
   The result follows directly from S-Trans.
   (iii) $[Z \mapsto U]S_n$ is a type variable:
   We already proved $\Gamma \vdash S \rightleftharpoons [Z \mapsto U]S_n$, as required.
   (iv) $[Z \mapsto U]S_n$ has the form $\forall X <: U_1.\ S_2$ with $\Gamma, X <: U_1 \vdash S_2 <: T_2$:
   We already proved $\Gamma \vdash S \rightleftharpoons [Z \mapsto U]S_n$, as required.

c. $\forall m.\ \Gamma, Z <: \mathrm{Top} \vdash \mathrm{disj}(T_s, U_m)$ and T_d is a match type with $\Gamma \vdash T_d \rightleftharpoons \forall X <: U_1.\ T_2$:
Using D-Sub on $\forall m.\ \Gamma, Z <: \mathrm{Top} \vdash S_m <: T_m$ and $\forall m.\ \Gamma, Z <: \mathrm{Top} \vdash \mathrm{disj}(T_s, U_m)$ we obtain $\forall m.\ \Gamma, Z <: \mathrm{Top} \vdash \mathrm{disj}(S_s, U_m)$. From S-Trans we get $\Gamma \vdash S_s <: C_d$. Using BS-Match3 and BS-Match4 we obtain $\Gamma \vdash S \rightleftharpoons S_d$. Since $\Gamma \vdash S_d <: T_d$ and $\Gamma \vdash T_d \rightleftharpoons \forall X <: U_1.\ T_2$, the IH gives 4 subsubcases:
   (i) S_d is a match type with $\Gamma \vdash S_d \rightleftharpoons \forall X <: U_1.\ S_2$ for some S_2 such that $\Gamma, X <: U_1 \vdash S_2 <: T_2$:
   The result follows directly from S-Trans.
   (ii) S_d is a match type with $\Gamma \vdash S_d \rightleftharpoons X$:
   The result follows directly from S-Trans.
   (iii) S_d is a type variable:
   We already proved $\Gamma \vdash S \rightleftharpoons S_d$, as required.
   (iv) S_d has the form $\forall X <: U_1.\ S_2$ with $\Gamma, X <: U_1 \vdash S_2 <: T_2$:
   We already proved $\Gamma \vdash S \rightleftharpoons S_d$, as required.

d. $\forall m.\ \Gamma, Z <: \mathrm{Top} \vdash \mathrm{disj}(T_s, U_m)$ and $T_d = \forall X <: U_1.\ T_2$:

Using D-Sub on $\forall m.\ \Gamma, Z <: \mathrm{Top} \vdash S_m <: T_m$ and $\forall m.\ \Gamma, Z <: \mathrm{Top} \vdash \mathrm{disj}(T_s, U_m)$ we obtain $\forall m.\ \Gamma, Z <: \mathrm{Top} \vdash \mathrm{disj}(S_s, U_m)$. From S-Trans we get $\Gamma \vdash S_s <: C_d$. Using BS-Match3 and BS-Match4 we obtain $\Gamma \vdash S \rightleftharpoons S_d$. Since $\Gamma \vdash S_d <: T_d$, the IH gives 4 subsubcases:

(i) S_d is a match type with $\Gamma \vdash S_d \rightleftharpoons \forall X <: U_1.\ S_2$ for some S_2 such that $\Gamma, X <: U_1 \vdash S_2 <: T_2$:
The result follows directly from S-Trans.

(ii) S_d is a match type with $\Gamma \vdash S_d \rightleftharpoons X$:
The result follows directly from S-Trans.

(iii) S_d is a type variable:
We already proved $\Gamma \vdash S \rightleftharpoons S_d$, as required.

(iv) S_d has the form $\forall X <: U_1.\ S_2$ with $\Gamma, X <: U_1 \vdash S_2 <: T_2$:
We already proved $\Gamma \vdash S \rightleftharpoons S_d$, as required.

- *Case* S-TVar: $S = Y \quad Y <: T \in \Gamma$
  S is a type variable and the result is immediate.
- *Case* S-All: $S = \forall X <: U_1.\ S_2 \quad T = \forall X <: U_1.\ T_2 \quad \Gamma, X <: U_1 \vdash S_2 <: T_2$
  S has the form $\forall X <: U_1.\ S_2$, with $\Gamma, X <: U_1 \vdash S_2 <: T_2$, as required.
- *Case* S-Top, S-Sin, S-Arrow, S-Psi:
  In those cases, T is neither a type variable nor a universal type and the result is immediate.

□

Lemma 3.8 (Canonical forms).

1. *If* $\Gamma \vdash t : T$, *where either* T *is a type variable, or* T *is a match type with* $\Gamma \vdash T \rightleftharpoons X$, *then* t *is not a closed value.*
2. *If* v *is a closed value with* $\Gamma \vdash v : T$ *where either* $T = T_1 \to T_2$, *or* T *is a match type and* $\Gamma \vdash T \rightleftharpoons T_1 \to T_2$, *then* v *has the form* $\lambda x : S_1.\ t_2$.
3. *If* v *is a closed value with* $\Gamma \vdash v : T$ *where either* $T = \forall X <: U_1.\ T_2$, *or* T *is a match type and* $\Gamma \vdash T \rightleftharpoons \forall X <: U_1.\ T_2$, *then* v *has the form* $\lambda X <: U_1.\ t_2$.

Proof:

1. By induction on a derivation of $\Gamma \vdash t : T$
   - *Case* T-Var, T-TApp, T-App, BT-Match:
     In those cases, t is not a value and the result is immediate.
   - *Case* T-Abs, T-TAbs, T-Class:
     In those cases, T is neither a match type nor a type variable and the result is immediate.
   - *Case* T-Sub: $\Gamma \vdash t : S \quad \Gamma \vdash S <: T$
     By Lemma 3.7, either S is a match type with $\Gamma \vdash S \rightleftharpoons Y$, or S is a type variable. In both cases, we use the IH to show that t is not a closed value, as required.
2. By induction on a derivation of $\Gamma \vdash t : T$.
   - *Case* T-Var, T-TApp, T-App, BT-Match:
     In those cases, t is not a value and the result is immediate.
   - *Case* T-Abs: $t = \lambda x : T_1 t_2 \quad T = T_1 \to T_2 \quad \Gamma, x : T_1 \vdash t_2 : T_2$
     $t = \lambda x : T_1 t_2$ and the result is immediate.
   - *Case* T-TAbs, T-Class:
     T is neither a function type nor a match type and the result is immediate.
   - *Case* T-Sub: $\Gamma \vdash t : S \quad \Gamma \vdash S <: T$
     Using Lemma 3.7 we get 4 subcases:

   a. S is a match type with $\Gamma \vdash S \rightleftharpoons S_1 \to S_2$:
      The result follows from the IH.
   b. S is a match type with $\Gamma \vdash S \rightleftharpoons X$:
      This case cannot occure since the 1th part of the lemma would lead to a contradiction on the fact that v is a closed value.
   c. S is a type variable:
      Similarly, this case cannot occure since the 1st part of the lemma would lead to a contradiction.
   d. S has the form $S_1 \to S_2$ with $\Gamma \vdash T_1 <: S_1$ and $\Gamma \vdash S_2 <: T_2$:
      The result follows from the IH.
3. By induction on a derivation of $\Gamma \vdash t : T$.
   - *Case* T-Var, T-TApp, T-App, BT-Match:
     In those cases, t is not a value and the result is immediate.
   - *Case* T-Abs, T-Class:
     T is neither a universal type nor a match type and the result is immediate.
   - *Case* T-TAbs: $t = \lambda Y <: T_1.\ t_2 \quad T = \forall Y <: T_1.\ T_2 \quad \Gamma, Y <: T_1 \vdash t_2 : T_2$
     $t = \lambda Y <: T_1.\ t_2$ and the result is immediate.
   - *Case* T-Sub: $\Gamma \vdash t : S \quad \Gamma \vdash S <: T$
     Using Lemma 3.7 we get 4 subcases:
     a. S is a match type with $\Gamma \vdash S \rightleftharpoons \forall X <: U_1.\ S_2$, The result follows from the IH.
     b. S is a match type with $\Gamma \vdash S \rightleftharpoons X$, This case cannot occure since the 1th part of the lemma would lead to a contradiction on the fact that v is a closed value.
     c. S is a type variable, Similarly, this case cannot occure since the 1st part of the lemma would lead to a contradiction.
     d. S has the form $\forall X <: U_1.\ S_2$ and $\Gamma, X <: U_1 \vdash S_2 <: T_2$. The result follows from the IH.

□

Lemma 3.9 (Inversion of typing).

1. *If* $\Gamma \vdash \lambda x : S_1.\ s_2 : T$ *and* $\Gamma \vdash T <: U_1 \to U_2$, *then* $\Gamma \vdash U_1 <: S_1$ *and there is some* S_2 *such that* $\Gamma, x : S_1 \vdash s_2 : S_2$ *and* $\Gamma \vdash S_2 <: U_2$.
2. *If* $\Gamma \vdash \lambda X <: S_1.\ s_2 : T$ *and* $\Gamma \vdash T <: (\forall X <: U_1.\ U_2)$, *then* $U_1 = S_1$ *and there is some* S_2 *such that* $\Gamma, X <: S_1 \vdash s_2 : S_2$ *and* $\Gamma, X <: S_1 \vdash S_2 <: U_2$.

Proof:

1. By induction on a derivation of $\Gamma \vdash t : T$.
   - *Case* T-Abs: $t = \lambda x : S_1 s_2 \quad T = S_1 \to T_2 \quad \Gamma\, x : S_1 \vdash s_2 : T_2$
     Given $\Gamma \vdash S_1 \to T_2 <: U_1 \to U_2$, we use Lemma 3.7 to obtain $\Gamma \vdash U_1 <: S_1$ and $\Gamma \vdash T_2 <: U_2$. We pick S_2 to be T_2 to obtain the desired result.
   - *Case* T-Sub: $\Gamma \vdash t : S \quad \Gamma \vdash S <: T$
     Using S-Trans with $\Gamma \vdash S <: T$ and $\Gamma \vdash T <: U_1 \to U_2$ we get $\Gamma \vdash S <: U_1 \to U_2$. The results follows directly from the IH.
   - *Case* T-Var, T-App, T-TAbs, T-TApp, T-Class, BT-Match:
     In those cases, t is not of the form $\lambda x : S_1.\ s_2$ and the result is immediate.
2. By induction on a derivation of $\Gamma \vdash t : T$.
   - *Case* T-TAbs: $t = \lambda X <: S_1.\ s_2 \quad T = \forall X <: S_1.\ T_2 \quad \Gamma, X <: S_1 \vdash s_2 : T_2$
     Given $\Gamma \vdash (\forall X <: S_1.\ T_2) <: (\forall X <: U_1.\ U_2)$, we use Lemma 3.7 to obtain $U_1 = S_1$ and $\Gamma, X <: S_1 \vdash T_2 <: U_2$. We pick S_2 to be T_2 to obtain the desired result.

- *Case* T-Sub: $\Gamma \vdash t : S \quad \Gamma \vdash S <: T$
  Using S-Trans with $\Gamma \vdash S <: T$ and $\Gamma \vdash T <: (\forall X <: U_1.\ U_2)$ we get $\Gamma \vdash S <: (\forall X <: U_1.\ U_2)$.
  The results follows directly from the IH.
- *Case* T-Var, T-Abs, T-App, T-TApp, T-Class, BT-Match:
  In those cases, t is not of the form $\lambda X <: S_1.\ s_2 : T$ and the result is immediate.

□

Lemma 3.10 (Minimum types).

1. *If* $\Gamma \vdash new\ C : T$ *then* $\Gamma \vdash \{new\ C\} <: T$.
2. *If* $\Gamma \vdash \lambda x : T_1.\ t_2 : T$ *then there is some* T_2 *such that* $\Gamma \vdash T_1 \to T_2 <: T$.
3. *If* $\Gamma \vdash \lambda X <: U_1.\ t_2 : T$ *then there is some* T_2 *such that* $\Gamma \vdash \forall X <: U_1.\ T_2 <: T$.

Proof:

1. By induction on a derivation of $\Gamma \vdash t : T$.
   - *Case* T-Var, T-Abs, T-App, T-TAbs, T-TApp, BT-Match:
     In those cases, t is not a constructor call and the result is immediate.
   - *Case* T-Class: $t = new\ C \quad T = \{new\ C\}$
     The result follows directly from S-Refl.
   - *Case* T-Sub: $\Gamma \vdash t : S \quad \Gamma \vdash S <: T$
     By the IH, $\Gamma \vdash \{new\ C\} <: S$. The result follows from S-Trans.
2. By induction on a derivation of $\Gamma \vdash t : T$.
   - *Case* T-Var, T-App, T-TAbs, T-TApp, T-Class, BT-Match:
     In those cases, t is not an abstraction and the result is immediate
   - *Case* T-Abs: $T = T_1 \to S_2 \quad \Gamma, x : T_1 \vdash t_2 : S_2$
     $T_2 = S_2$ and the result is immediate using S-Refl.
   - *Case* T-Sub: $\Gamma \vdash t : S \quad \Gamma \vdash S <: T$
     Using the IH, there is some T_2 such that $\Gamma \vdash T_1 \to T_2 <: S$. The result follows from S-Trans.
3. By induction on a derivation of $\Gamma \vdash t : T$.
   - *Case* T-Var, T-Abs, T-App, T-TApp, T-Class, BT-Match:
     In those cases, t is not a type abstraction and the result is immediate
   - *Case* T-TAbs: $T = \forall X <: U_1.\ S_2 \quad \Gamma, X <: U_1 \vdash t_2 : S_2$
     $T_2 = S_2$ and the result is immediate using S-Refl.
   - *Case* T-Sub: $\Gamma \vdash t : S \quad \Gamma \vdash S <: T$
     Using the IH, there is some T_2 such that $\Gamma \vdash \forall X <: U_1.\ T_2 <: S$. The result follows from S-Trans.

□

Theorem 3.11 (Preservation).
If $\Gamma \vdash t : T$ *and* $t \longrightarrow t'$ *then* $\Gamma \vdash t' : T$.

Proof: By induction on a derivation of $\Gamma \vdash t : T$.

- *Case* T-Var, T-Abs, T-TAbs, T-Class:
  These cases cannot arise since we assume $t \longrightarrow t'$ but there are no evaluation rules for variables, abstractions, type abstractions and class instantiations.
- *Case* T-App: $t = t_1 t_2 \quad T = T_{12} \quad \Gamma \vdash t_1 : T_{11} \to T_{12} \quad \Gamma \vdash t_2 : T_{11}$
  By definition of the evaluation relation, there are 3 subcases:
  Subcase E-App1: $t_1 \longrightarrow t_1' \quad t' = t_1' t_2$

By the IH and the 1st premise we get $\Gamma \vdash t_1' : T_{11} \to T_{12}$. We use T-App with that result and the 2nd premise to obtain $\Gamma \vdash t_1' t_2 : T_{12}$, as required.

Subcase E-App2: $t_2 \longrightarrow t_2' \quad t_1 = v_1 \quad t' = v_1 t_2'$

Analogously, the IH gives $\Gamma \vdash t_2' : T_{12}$ and the result follows from T-App.

Subcase E-AppAbs: $t_1 = \lambda x : U_{11}.\ u_{12} \quad t' = [x \mapsto t_2]u_{12}$

By Lemma 3.9 (with $S_1 = U_{11}$, $s_2 = u_{12}$, $U_1 = T_{11}$ and $U_2 = T_{12}$), there is some S_2 such that $\Gamma, x : U_{11} \vdash u_{12} : S_2$, $\Gamma \vdash S_2 <: T_{12}$ and $\Gamma \vdash T_{11} <: U_{11}$. Using T-Sub with $\Gamma \vdash t_2 : T_{11}$ and $\Gamma \vdash T_{11} <: U_{11}$ we obtain $\Gamma \vdash t_2 : U_{11}$. By Lemma 3.5 we get $\Gamma \vdash [x \mapsto t_2]u_{12} : S_2$. Using T-Sub we obtain $\Gamma \vdash [x \mapsto t_2]u_{12} : T_{12}$, as required.

– *Case* T-TApp: $t = t_1 T_2 \quad T = [X \mapsto T_2]T_{12}$
$\Gamma \vdash t_1 : (\forall X <: U_1.\ T_{12}) \quad \Gamma \vdash T_2 <: U_1$

The proof for case T-TApp is analogous to the one for T-App. By definition of the evaluation relation, there are 2 subcases:

Subcase E-TApp: $t_1 \longrightarrow t_1' \quad t' = t_1' T_2$

By the IH and the 1st premise we get $\Gamma \vdash t_1' : (\forall X <: U_1.\ T_{12})$. We use T-TApp with that result and the 2nd premise to obtain $\Gamma \vdash t_1' T_2 : [X \mapsto T_2]T_{12}$, as required.

Subcase E-TAppTAbs: $t_1 = \lambda X <: U_{11}.\ u_{12} \quad t' = [X \mapsto T_2]u_{12}$

By Lemma 3.9 (with $S_1 = U_{11}$, $s_2 = u_{12}$ and $U_2 = T_{12}$), there is some S_2 such that $\Gamma, X <: U_{11} \vdash u_{12} : S_2$, $U_1 = U_{11}$, and $\Gamma, X <: U_{11} \vdash S_2 <: T_{12}$. Since $\Gamma \vdash T_2 <: U_{11}$, we use Lemma 3.5 twice to get $\Gamma \vdash [X \mapsto T_2]u_{12} : [X \mapsto T_2]S_2$ and $\Gamma \vdash [X \mapsto T_2]S_2 <: [X \mapsto T_2]T_{12}$. By T-Sub $\Gamma \vdash [X \mapsto T_2]u_{12} : [X \mapsto T_2]T_{12}$, as required.

– *Case* T-Sub: $\Gamma \vdash t : S \quad \Gamma \vdash S <: T$

By the IH, $\Gamma \vdash t' : S$. The result follows directly from T-Sub.

– *Case* BT-Match: $t = t_s\ \mathit{match}[X]\{x_i : C_i \Rightarrow t_i\}\mathit{or}\ t_d \quad T = T_s\ \mathit{match}[X]\{C_i \Rightarrow T_i\}\mathit{or}\ T_d$
$\Gamma \vdash t_s : T_s$
$\Gamma, X <: \mathrm{Top}, x_i : C_i \vdash t_i : T_i$
$\Gamma \vdash t_d : T_d$

By definition of the evaluation relation, there are 5 subcases:

Subcase E-Match1: $t_s \longrightarrow t_s' \quad t' = t_s'\ \mathit{match}[X]\{x_i : C_i \Rightarrow t_i\}\mathit{or}\ t_d$

By the IH $\Gamma \vdash t_s' : T_s$. The result follows directly from BT-Match.

Subcase E-Match2: $t_s = \mathit{new}\, C \quad (C, [X \mapsto U]C_n) \in \Psi$
$\forall m < n.\ \forall T.\ (C, [X \mapsto T]C_m) \notin \Psi \quad t' = [X \mapsto U][x_n \mapsto \mathit{new}\, C]t_n$

By S-Psi, S-Sin and S-Trans, $\Gamma \vdash \{\mathit{new}\, C\} <: [X \mapsto U]C_n$. From D-Psi, $\forall m < n.\ \forall T.\ \Gamma \vdash \mathrm{disj}(\{\mathit{new}\, C\}, [X \mapsto T]C_m)$. By Lemma 3.10 we get $\Gamma \vdash \{\mathit{new}\, C\} <: T_s$. Let T_1 be $\{\mathit{new}\, C\}\ \mathit{match}[X]\{C_i \Rightarrow T_i\}\mathit{or}\ T_d$. From BS-Match1, $\Gamma \vdash [X \mapsto U]T_n <: T_1$. Using BS-Match5, $\Gamma \vdash T_1 <: T$. By S-Trans, $\Gamma \vdash [X \mapsto U]T_n <: T$.

Using T-Class, T-Sub, S-Sin and S-Psi (with $(C, [X \mapsto U]C_n) \in \Psi$) we get $\Gamma \vdash \mathit{new}\, C : [X \mapsto U]C_n$. From the case premises, we have $\Gamma, X <: \mathrm{Top}, x_n : C_n \vdash t_n : T_n$. By Lemma 3.5, we get $\Gamma \vdash [X \mapsto U][x_n \mapsto \mathit{new}\, C]t_n : [X \mapsto U]T_n$. Finally, using T-Sub we get , $\Gamma \vdash [X \mapsto U][x_n \mapsto \mathit{new}\, C]t_n : T$, as required.

Subcase E-Match3: $t_s = \mathit{new}\, C \quad \forall n.\ (C, C_n) \notin \Psi \quad t' = t_d$

The proof for subcase E-Match3 is analogous to the one for E-Match2. From D-Psi, $\forall n.\ \Gamma \vdash \mathrm{disj}(\{\mathit{new}\, C\}, C_n)$. By Lemma 3.10 we get $\Gamma \vdash \{\mathit{new}\, C\} <: T_s$. Let T_1 be $\{\mathit{new}\, C\}\ \mathit{match}[X]\{C_i \Rightarrow T_i\}\mathit{or}\ T_d$. From BS-Match2, $\Gamma \vdash T_d <: T_1$. Using BS-Match5, $\Gamma \vdash T_1 <: T$. By S-Trans $\Gamma \vdash T_d <: T$. Using T-Sub, $\Gamma \vdash t_d : T$, as required.

Subcase E-Match4: $t_s = \lambda x : U.\ u \quad t' = t_d$

By Lemma 3.10, there exists a V such that $\Gamma \vdash U \to V <: T_s$. Using D-Arrow, $\forall n.\ \Gamma \vdash \mathrm{disj}(U \to V, C_n)$. Let T_1 be $U \to V\ \mathit{match}[X]\{C_i \Rightarrow T_i\}\mathit{or}\ T_d$. From BS-Match2, $\Gamma \vdash T_d <:$

T_1. Using BS-Match5, $\Gamma \vdash T_1 <: T$. By S-Trans $\Gamma \vdash T_d <: T$. Using T-Sub, $\Gamma \vdash t_d : T$, as required.
Subcase E-Match5: $t_s = \forall X <: U.\ u \quad t' = t_d$
The proof for subcase E-Match5 is analogous to the one for E-Match4. By Lemma 3.10, there exists a V such that $\Gamma \vdash (\forall X <: U.\ V) <: T_s$. Using D-Arrow, we get $\forall n.\ \Gamma \vdash \text{disj}(\forall X <: U.\ V, C_n)$. Let T_1 be $(\forall X <: U.\ V)$ *match*$[X]\{C_i \Rightarrow T_i\}$ *or* T_d. From BS-Match2, $\Gamma \vdash T_d <: T_1$. Using BS-Match5, $\Gamma \vdash T_1 <: T$. By S-Trans $\Gamma \vdash T_d <: T$. Using T-Sub, $\Gamma \vdash t_d : T$, as required.

□

Theorem 3.12 (Progress).
If t *is a closed, well-typed term, then either* t *is a value or there is some* t' *such that* $t \longrightarrow t'$.

Proof: By induction on a derivation of $\Gamma \vdash t : T$.

- *Case* T-Var: $t = x \quad x : T \in \Gamma$
  This case cannot occure because t is closed.
- *Case* T-Abs, T-TAbs, T-Class:
  In those cases, t is a value and the result is immediate.
- *Case* T-App: $t = t_1 t_2 \quad T = T_{12} \quad \Gamma \vdash t_1 : T_{11} \to T_{12} \quad \Gamma \vdash t_2 : T_{11}$
  By the IH, either t_1 is a value or t_1 can take a step (there is some t_1' such that $t_1 \longrightarrow t_1'$). If t_1 can take a step, then E-App1 applies to t. If t_1 is a value, we use Lemma 3.8 to get that t_1 has the form $\lambda x : S_1.\ t_2$. Therefore, E-AppAbs applies to t, as required.
- *Case* T-TApp: $t = t_1 T_2 \quad T = [X \mapsto T_2]T_{12}$
  $\Gamma \vdash t_1 : (\forall X <: U_1.\ T_{12}) \quad \Gamma \vdash T_2 <: U_1$
  The proof for case T-TApp is analogous to the one for T-App.
  By the IH, either t_1 is a value or t_1 can take a step. If t_1 can take a step, then E-TApp applies to t. If t_1 is a value, we use Lemma 3.8 to get that t_1 has the form $\lambda X <: T_1.\ t_2$. Therefore, E-TAppTAbs applies to t, as required.
- *Case* T-Sub: $\Gamma \vdash t : S \quad \Gamma \vdash S <: T$
  The result follows directly from the IH.
- *Case* BT-Match: $t = t_s$ *match*$[X]\{x_i : C_i \Rightarrow t_i\}$ *or* $t_d \quad T = T_s$ *match*$[X]\{C_i \Rightarrow T_i\}$ *or* T_d
  $\Gamma \vdash t_s : T_s$
  $\Gamma, X <: \text{Top}, x_i : C_i \vdash t_i : T_i$
  $\Gamma \vdash t_d : T_d$
  By the IH, either t_s is a value or t_s can take a step. If t_s can take a step, then E-Match1 applies to t, as required. If t_s is a value, t_s can take 3 different forms:
  Subcase t_s is of the form *new* C:
  If $\forall m.\ \forall T.\ (C, [X \mapsto T]C_m) \notin \Psi$, then E-Match3 applies to t. Otherwise, let C_k be the first class such that $(C, [X \mapsto U]C_k) \in \Psi$ for some type U. By construction we know that $\forall m < k.\ \forall T.\ (C, [X \mapsto T]C_k) \notin \Psi$. Therefore E-Match2 applies to t, as required.
  Subcase t_s is of the form $\lambda x : T_1 t_2$:
  E-Match4 applies to t and the result is immediate.
  Subcase t_s is of the form $\forall X <: U_1.\ T_2$:
  E-Match5 applies to t and the result is immediate.

□